

Table of Contents

1. Abstract.....	3
2. Introduction	4
2.1 Project Objective	4
2.2 Project Overview	4
2.3 Programming Platform	5
2.4 Project Architecture	6
3. System level of our project.....	7
3.1 Overview of the system level of our project.....	7
3.2 DirectShow library	7
3.2.1 What is DirectShow?	7
3.2.2 DirectShow System Overview	8
3.2.3 DirectShow Application Programming	9
3.3 WinSock library	11
3.3.1 What is WinSock?	11
3.3.2 WinSock 2 Architecture	12
3.3.3 The Sockets Programming Paradigm under Windows	13
3.3.4 Asynchronous Notification Using Event Objects.....	14
4. Unix-based BSD UDP chatroom	16
4.1 Introduction	16
4.2 Aim of this program.....	16
4.3 Outcome	16
4.4 The pseudo-code of this program	17
4.4.1 Server side	17
4.4.2 Client side.....	18
5. WinChat.....	19
5.1 Introduction	19
5.2 Aim of writing this program.....	19
5.3 Outcome	20
5.4 The pseudo-code of WinChat	20
5.4.1 Server side	20
5.4.2 Client side.....	21
6. Simple video player.....	22
6.1 Introduction	22
6.2 Aim of writing this program.....	22
6.3 Filter graph of video player.....	23
6.4 Outcome	23

6.5	The pseudo-code of video player	24
7.	MFC Chatting	25
7.1	Introduction	25
7.2	Aim of writing this program.....	25
7.3	New features	26
7.3.1	Server side	26
7.3.2	Client side.....	27
7.4	Outcome	30
7.5	The pseudo-code of MFC Chatting	31
7.5.1	Server side	31
7.5.2	Client side.....	32
8.	WinCap.....	33
8.1	Introduction	33
8.2	Aim of writing this program.....	33
8.3	Filter graph of video capturer	34
8.4	Outcome	34
8.5	Prototype of Class CPlayVideo	35
8.6	Prototype of Class CCaptureVideo	36
8.7	The pseudo-code of video player	37
9.	Integrated program.....	38
9.1	Introduction	38
9.2	Aim of this integrated program	38
9.3	Outcome	39
10.	Problem facing	40
10.1	Problems	40
10.2	Solution	41
10.2.1	Java Media Framework (JAVA JMF)	41
10.2.2	Why using JMF to solve the problem?	41
11.	Conclusion and Future Work.....	44
Appendix A : References		45
Appendix B : JavaPlayer.....		47
Appendix C : JavaCapturer.....		50
Appendix D: Progress Report		53

1. Abstract

Wireless technology provides ubiquitous connection in a concentrated area. When wireless network is established for a campus, teachers and students can enjoy closer interactions and better access to knowledge resources. Not only students will feel more open and comfortable when they ask questions through wireless connections with the teachers, but also teachers can interact with the students more privately and directly, thus creating more personal relationship between teachers and students and improving the quality of education. This close-up interactive environment is hard to be provided with clumsy desk-top computers in a classroom setting, where such a classroom would have to be carefully furnished and wired, thereby lowering the density of participants in the room.

Wireless campus also allows teachers and students to stay in touch with education material. They can conveniently access the library, check their e-mails, chat with others, search information on the Internet, and retrieve or review education material. Combining with thin-client devices, the wireless campus is an effective way for IT ownership in schools with much reduced cost, compared with wired campus. Furthermore, wireless campus encourages portability, connectivity, and communication efficiency. The deployed network is scalable and flexible.

2. Introduction

2.1 Project Objective

Our objective of doing the project is to give a virtual environment for student for learning knowledge without the restriction on time and place. In other words, we want to create a virtual classroom for education aspect.

Virtual Classroom

Virtual Classroom is a teaching and learning environment constructed in software, which supports collaborative learning among students who participate at times and places of their choosing, through computer networks. In our project, we want to provide an inter-active way of communication for teacher and students.

- Discussion on course materials like lecture notes, course projects through chat-room.
- Discussion through video-conferencing
- Students can also attend lessons on-line using computer networks. Besides, they can raise questions through text or video.
- Course can also be made available on video file through video capturing. Students can attend the lecture through streaming on the video file.
- Course material can be distributed through computer network

2.2 Project Overview

In our project, we will implement the whole system, system level and application level.

In system level, there are a server and libraries.

The server works as a central unit of the whole system. It controls every message to pass through the system and controls the usage of resource inside the system. This is the most important part for the system to work properly.

The application level, programs are used by the lecturer/students. They

can invoke one or more activities (message or chat room) in it. The client program will create an instance for each activity locally and every change will be distributed to all the others.

The client program is for received the response from the user and send it to the server. After server received their response, it will calculate the new state of the user and distribute it to all of the users within the system.

The libraries help us to develop a highly interactive network application. The libraries include Winsock and DirectShow library. Based on them, we can design and build the system more efficient. Also, we have tried to develop the software into two approaches. One of the approaches is to have user-friendly interface. The second one is to have a stable system state for client to use.

2.3 Programming Platform

All of the programs are written in Microsoft Visual C++ or Java.

For the server program, it needs to run on Microsoft Windows 95/98/2000/NT platform with Winsock support.

For the client program, it needs to run on Microsoft Windows 95/98/2000 platform with DirectMedia 6.0 or above support.

2.4 Project Architecture

This project Virtual Classroom is mainly divided into two parts.

The first part is the server, which mainly responsible for central control of the whole system. The second part is the client program, which is mainly responsible for display the learning material and interactive activities.

Basically, our work includes building the server and building the client program.

For the server we had built, they will be talked in details at Chapter 3 in this report.

For the client program we had built, detail will be talk in Chapter 4 in this report. Here is the architecture overview:

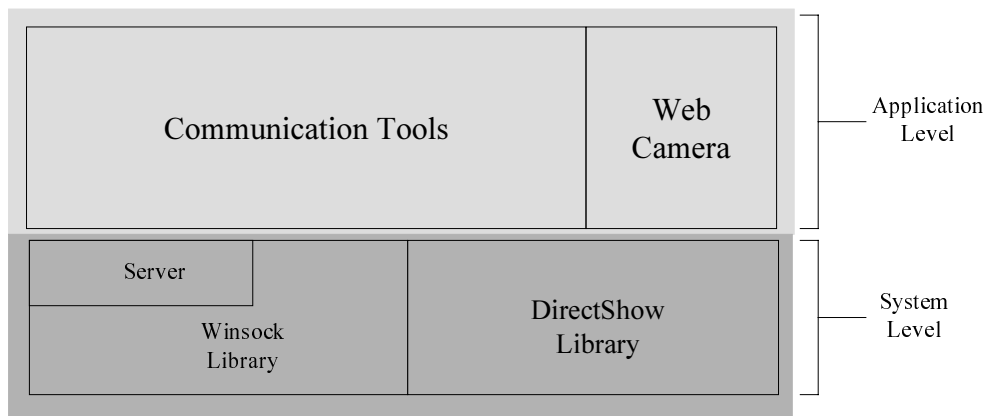


figure2.1 Architecture Overview

3. System level of our project

3.1 Overview of the system level of our project

In our system, there are 3 main components to construct the system level in our project. List as below:

- DirectShow library
- WinSock library
- Server

In the following section, we will introduce some knowledge of each of the components in details, for example: what is it?

3.2 DirectShow library

3.2.1 What is DirectShow?

Microsoft® DirectShow® is an architecture for streaming media on the Microsoft® Windows® platform. It is based on the Component Object Model (COM). DirectShow provides for high-quality capture and playback of multimedia streams. It supports a wide variety of formats, including Advanced Streaming Format (ASF), Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), MPEG Audio Layer-3 (MP3), and WAV files. It supports capture using Windows Driver Model (WDM) devices or older Video for Windows devices..

DirectShow simplifies media playback, format conversion, and capture tasks. At the same time, it provides access to the underlying stream control architecture for applications that require custom solutions.

Examples of the types of applications

- DVD players
- Video editing applications
- AVI to ASF converters
- MP3 players
- Digital video captures applications.

3.2.2 DirectShow System Overview

The following diagram shows the relationship between an application, the DirectShow components, and some of the hardware and software components that DirectShow supports.

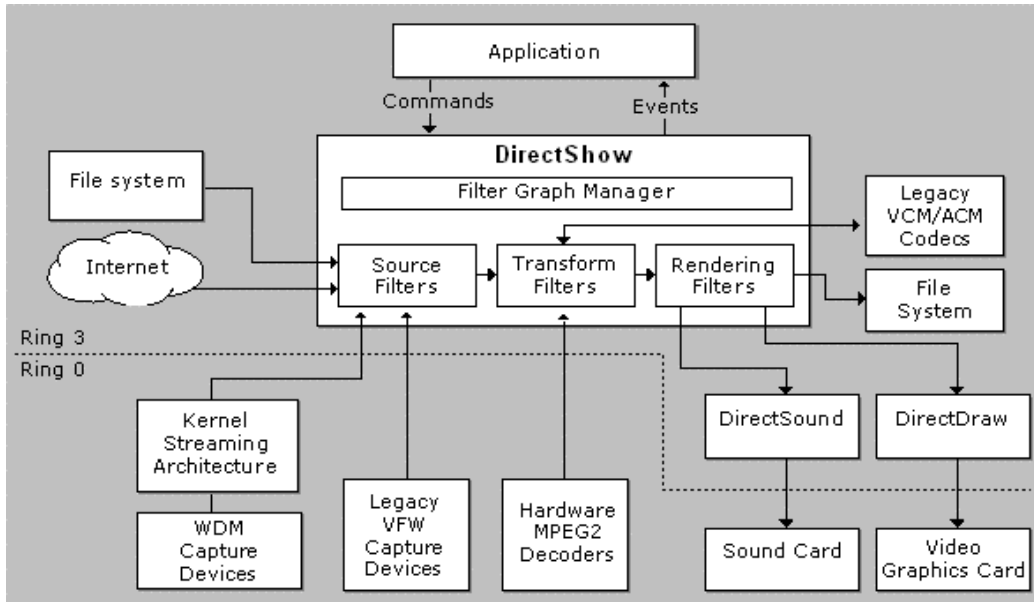


figure 3.1 DirectShow System Overview

As illustrated here, DirectShow enables applications to play files and streams from various sources, including local files, local CD and DVD drives, remote files on a network, newer TV-tuner and video capture cards based on the Windows Driver Model (WDM), and legacy Video For Windows® video capture cards. DirectShow has native compressors and decompressors for some file formats, and many third-party hardware and software decoders are compatible with DirectShow. In addition, DirectShow supports legacy Vfw codecs based on the Video Compression Manager (VCM) and Audio Compression Manager (ACM) interfaces. Playback makes full use of DirectDraw hardware acceleration and DirectSound capabilities when the hardware supports it.

3.2.3 DirectShow Application Programming

At the heart of the DirectShow services is a modular system of pluggable components called filters, arranged in a configuration called a filter graph. A component called the filter graph manager oversees the connection of these filters and controls the stream's data flow.

Filter

The basic building block of DirectShow is a software component called a filter. A filter generally performs a single operation on a multimedia stream. For example, there are DirectShow filters that

- Read files.
- Get video from a video capture device.
- Decode a particular stream format, such as MPEG-1 video.
- Pass data to the graphics or sound card.

Filter Graph

A filter graph is composed of a collection of filters of different types. Most filters can be categorized into one of the following three types.

- A *source filter*, which takes the data from some source, such as a file on disk, a satellite feed, an Internet server, or a VCR, and introduces it into the filter graph.
- A *transform filter*, which takes the data, processes it, and then passes it along.
- A *rendering filter*, which renders the data; typically this is rendered to a hardware device, but could be rendered to any location that accepts media input (such as memory or a disk file).

Filters receive input and produce output. For example, if a filter decodes MPEG-1 video, the input is the MPEG-encoded stream and the output is an uncompressed RGB video stream.

To perform a given task, an application connects several filters so that the output from one filter becomes the input for another. A set of connected filters is called a **filter graph**. As an illustration of this concept, the following diagram shows a filter graph for playing an AVI file.

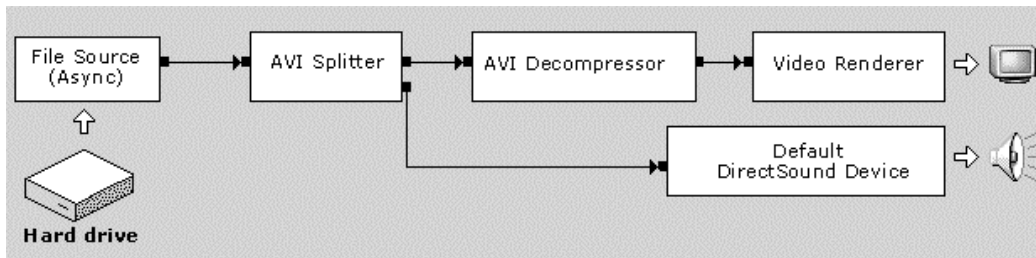


figure 3.2 DirectShow filter graph

Filter Graph Manager

DirectShow provides a high-level component called the Filter Graph Manager. The Filter Graph Manager controls the flow of data through the graph. We can make high-level API calls such as "Run" (to move data through the graph) or "Stop" (to stop the flow of data). We can also access the filters directly through COM interfaces. The Filter Graph Manager also passes event notifications to the application, so that our application can respond to events, such as the end of a stream.

In addition, the Filter Graph Manager simplifies the process of building a filter graph. For example, you can specify a file name, and the Filter Graph Manager will build a graph to play that file.

Writing a DirectShow Application

A typical DirectShow application performs three basic steps, as illustrated in the following diagram.

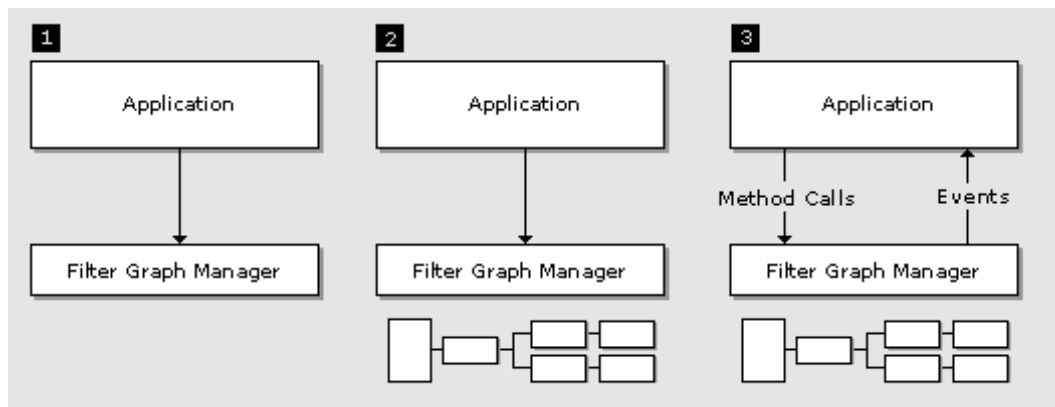


figure 3.3 Process of writing a DirectShow application

1. Creates an instance of the Filter Graph Manager, using the CoCreateInstance function.
2. Uses the Filter Graph Manager to build a filter graph.
3. Controls the filter graph and responds to events.

3.3 WinSock library

3.3.1 What is WinSock?

Windows Sockets (Winsock) specifies a programming interface based on the familiar “socket” interface from the University of California at Berkeley. It includes a set of extensions designed to take advantage of the message-driven nature of Microsoft Windows. Version 1.1 of the Windows Sockets specification was released in January 1993, and version 2.2.0 was published in May of 1996. Windows CE supports Winsock version 1.1.

Sockets are a general-purpose networking API. Winsock is designed to run efficiently on Windows operating systems while maintaining compatibility with the Berkeley Software Distribution (BSD) standard, known as Berkeley Sockets.

Windows Sockets applications can also be used in Windows CE, thus WinSock is suitable for design multi-platform software to provide generic network services.

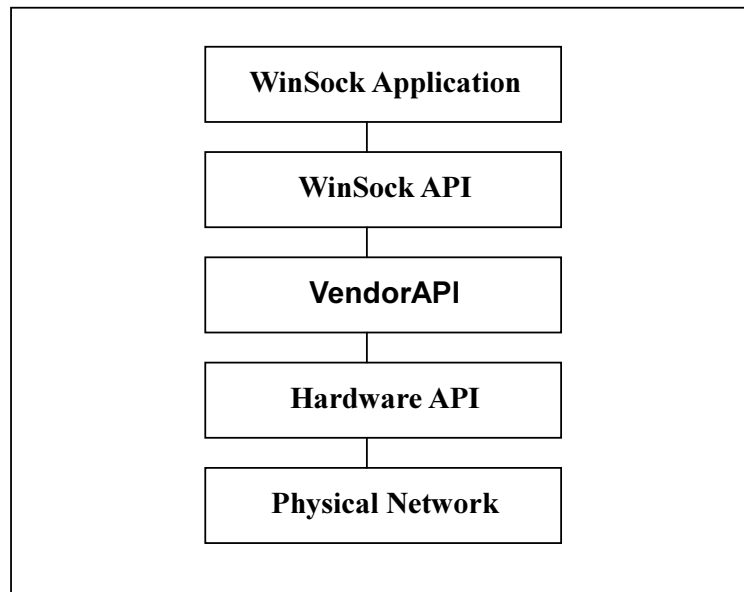


figure3.4 Overviews of how WinSock works

3.3.2 WinSock 2 Architecture

WinSock 2 has an all-new architecture that provides much more flexibility. The new WinSock 2 architecture allows for simultaneous support of multiple protocol stacks, interfaces, and service providers. There is still one DLL on top, but there is another layer below, and a standard service provider interface, both of which add flexibility.

WinSock 2 adopts the Windows Open Systems Architecture (WOSA) model, which separates the API from the protocol service provider. In this model the WinSock DLL provides the standard API, and each vendor installs its own service provider layer underneath. The API layer "talks" to a service provider via a standardized Service Provider Interface (SPI), and it is capable of multiplexing between multiple service providers simultaneously. The following sketch illustrates the WinSock 2 architecture.

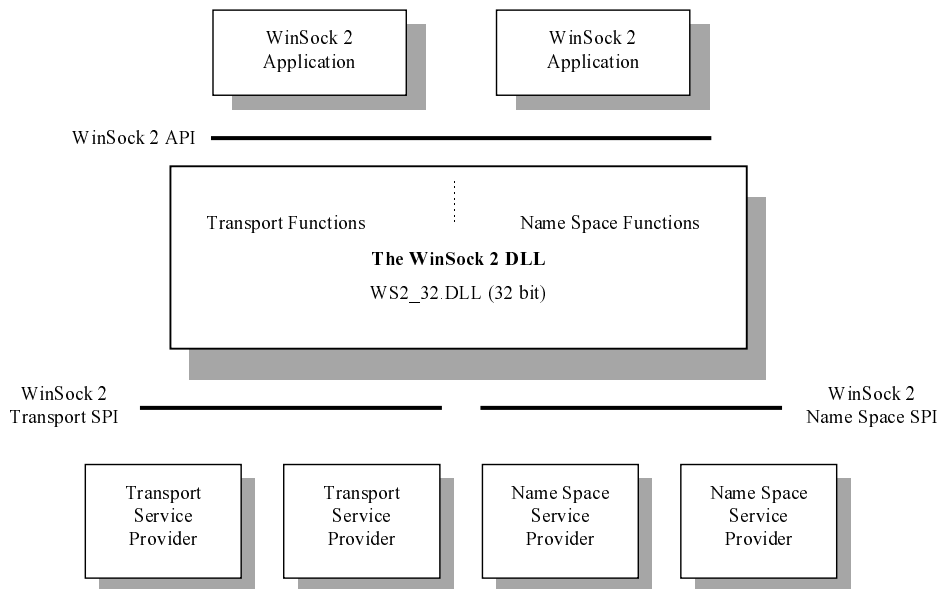


figure 3.5 WinSock 2 Architecture

Note that the WinSock 2 specification has two distinct parts: the API for application developers, and the SPI for protocol stack and namespace service providers. Notice also that the intermediate DLL layers are independent of both the application developers and service providers. These DLLs are provided and maintained by Microsoft and Intel. And lastly, notice that the *Layered Service Providers* would appear in this illustration one or more boxes on top of a transport service provider.

However, WinSock 2 is designed only for 32-bit Windows platforms; it cannot be run using Win32s on 16-bit Windows platforms. Nonetheless, as described earlier, (almost) all 16-bit WinSock 1.1 applications can be used. Windows NT version 4 has been called the "shell-update," since the most obvious change was the addition of the Windows 95 user interface. But there's a whole lot more that was changed within the NT4 kernel design. Significant modifications were done to make WinSock 2 the native network API for Windows NT4. As a result, *WinSock 2 will not be available for NT 3.51 or earlier**.

3.3.3 The Sockets Programming Paradigm under Windows

In WinSock 2, all connectionless protocols use SOCK_DGRAM sockets and all connection-oriented protocols use SOCK_STREAM sockets. Programmers should no longer rely on socket type to describe all of the essential attributes of a transport protocol.

Windows Sockets 2 takes the socket paradigm considerably beyond what it's original designers contemplated. As a consequence, a number of new functions have been added, all of which are assigned names that are prefixed with "WSA". In all but a few instances these new functions are expanded versions of an existing function from BSD sockets. The need to retain backwards compatibility mandates that we retain both the "just plain" BSD functions and the new "WSA" versions.

The usage of functions for using a connection-oriented and connectionless application is shown in figure2.6 and figure2.7 respectively.

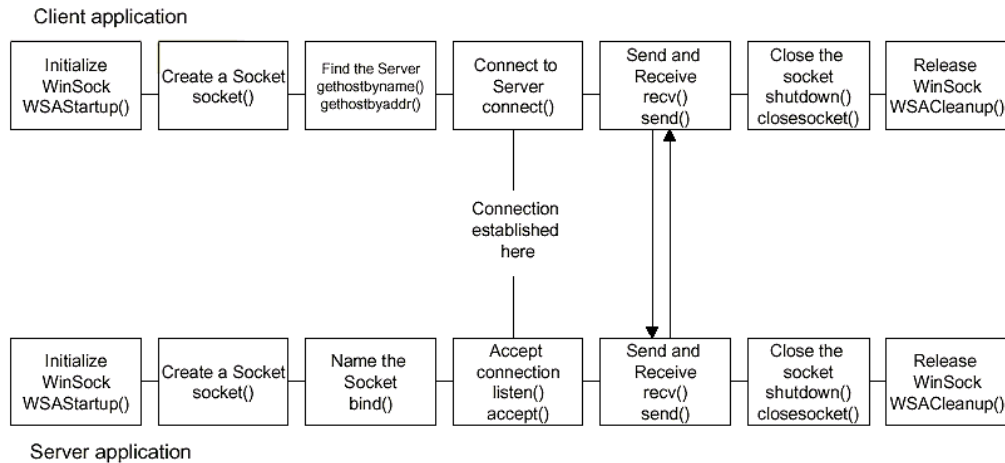


figure 3.6 Overview of Connection-Oriented Application

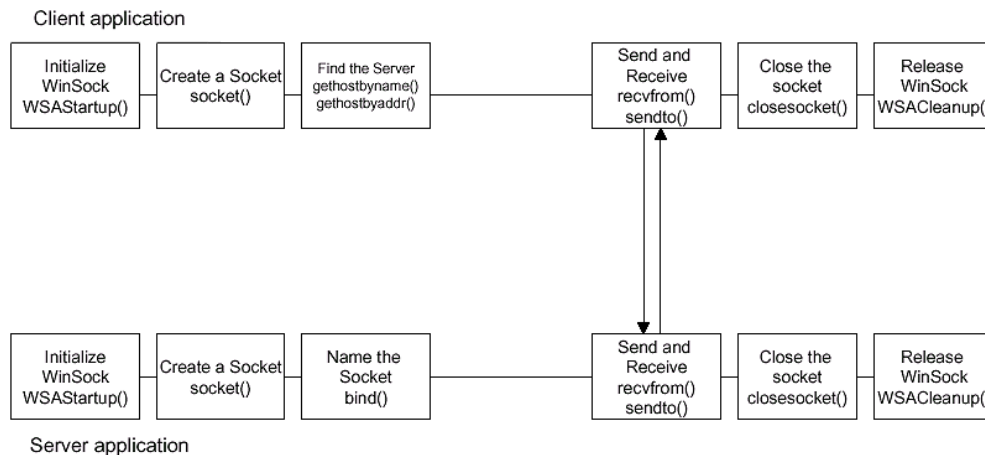


figure 3.7 Overview of Connectionless Application

3.3.4 Asynchronous Notification Using Event Objects

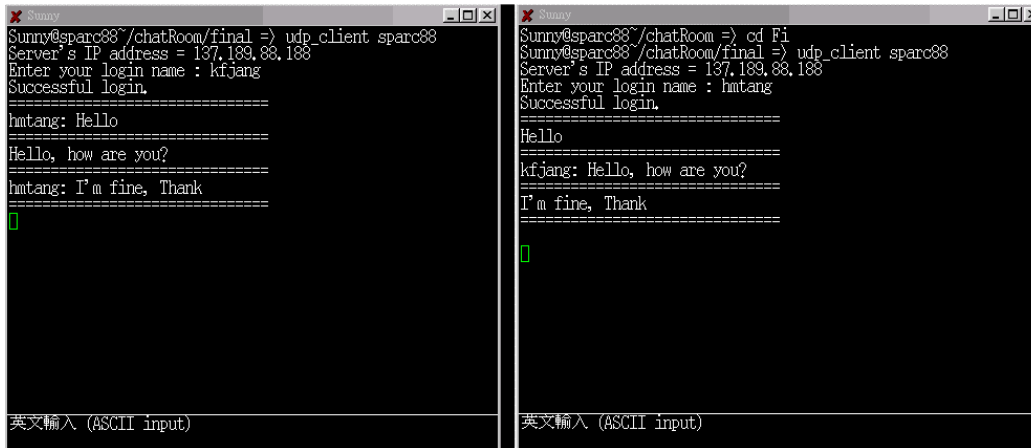
Introducing overlapped I/O requires a mechanism for applications to unambiguously associate send and receive requests with their subsequent completion indications. In WinSock 2 this may be accomplished via event objects that are modeled after Win32 events. WinSock event objects are fairly simple constructs, which can be created and closed, set and cleared, waited upon and polled. Their prime usefulness comes from the ability of an application to block and wait until one or more event objects become set.

Applications use **WSACreateEvent()** to obtain an event object handle which may then be supplied as a required parameter to the overlapped versions of send and receive calls (**WSASend()**, **WSASendTo()**, **WSARecv()**, **WSARecvFrom()**). The event object, which is cleared when first created, is set by the transport providers when the associated overlapped I/O operation has completed (either successfully or with errors). Each event object created by **WSACreateEvent()** should have a matching **WSACloseEvent()** to destroy it.

Event objects are also used in **WSAEventSelect()** to associate one or more FD_XXX network events with an event object.

The **WSAEventSelect()** and **WSAEnumNetworkEvents()** functions are provided. **WSAEventSelect()** behaves exactly like **WSAAsyncSelect()** except that, rather than cause a Windows message to be sent on the occurrence of an FD_XXX network event (e.g.. FD_READ, FD_WRITE, etc.), an application-designated event object is set.

4. Unix-based BSD UDP chatroom



```
Sunny@sparc68~/chatRoom/final => udp_client sparc68
Server's IP address = 137.189.88.188
Enter your login name : kfjang
Successful login.
=====
hmtang: Hello
=====
Hello, how are you?
=====
hmtang: I'm fine, Thank
=====
[ ]
英文輸入 (ASCII input)
```

```
Sunny@sparc68~/chatRoom/final => od Fi
Sunny@sparc68~/chatRoom/final => udp_client sparc68
Server's IP address = 137.189.88.188
Enter your login name : hmtang
Successful login.
=====
Hello
=====
kfjang: Hello, how are you?
=====
I'm fine, Thank
=====
[ ]
英文輸入 (ASCII input)
```

figure4.1 Snapshot of this program

4.1 Introduction

Unix-based BSD UDP chatroom is a simple UDP chatroom. It is a network chatting application using a client/server model. It supports multi-client to join. Also, the program is using text mode for display the chatroom message.

4.2 Aim of this program

- Use Berkeley Socket for network programming. Because this is the first time we learn network programming.
- Using Blocking-mode of communication programming paradigm. It is the basic knowledge for writing network programming.
- Using UDP protocol. UDP protocol is simpler than the TCP protocol, thus we first learn this programming paradigm.

4.3 Outcome

- We have learnt some basic function with Berkeley Socket programming, e.g **socket()** , **recvfrom()**, **sendto()**, **close()**.
- We have used the blocking-mode of communication programming paradigm. The main disadvantage of using the blocking-mode is that need to wait for the reply from the other side for further execute.
- We have success to write the chatroom using UDP protocol. It is quite easy to learn the UDP programming paradigm.

4.4 The pseudo-code of this program

4.4.1 Server side

```
Main()
{
    Create UDP socket
    Bind socket to server address
    Loop
    {
        Receive Message
        If Message is login message
            If new client
                Add client into client list
            Send acknowledgement to client
        Else
            Search client name from client list
            Add client name to the message receive
            Send the received message to all clients
    }
    Close socket
}
```

4.4.2 Client side

```
Main()
{
    Create UDP socket
    Bind socket to client address
    Send login message to server
    Receive acknowledgement
    Initialize select descriptor
    Loop
    {
        If input from standard input
            Get input string
            Send message to server
        If input from socket
            Receive message from server
            Print the message
    }
    Close socket
}
```

5. WinChat

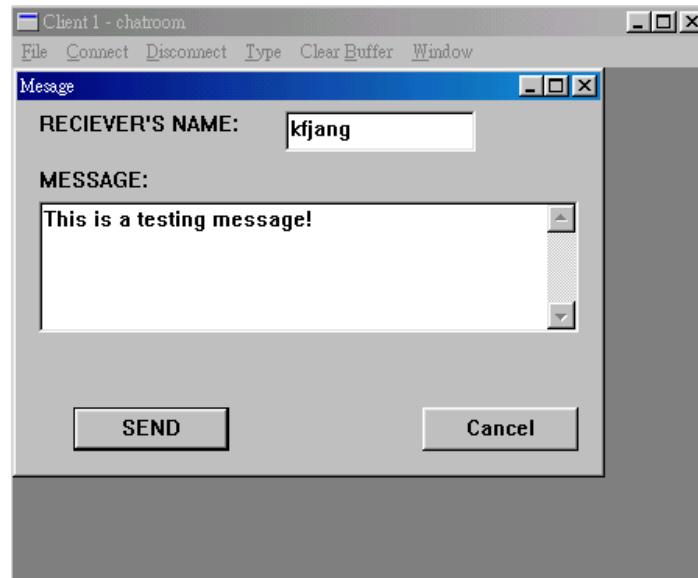


figure5.1 Snapshot of the WinChat

5.1 Introduction

WinChat is a one-way communication program in windows platform. It is a network chatting application using a client/server model with TCP protocol. It allows two users at two different machines to send messages to each other. The requirement for using this application is that client needs to know where the server is, that mean the IP address and the port number of the server.

5.2 Aim of writing this program

- Using Window Socket for network programming. Because we haven't use WinSock for writing network program.
- Use TCP protocol. After learning the UDP protocol, we need to learn the other popular transfer protocol – TCP protocol.
- Use Visual C++. This is the first time we using window-based compiler for writing program.
- Use Asynchronous mode of communication with event-driven programming paradigm. Because this program can help us to learn the asynchronous mode of communication with event-driven programming paradigm more clearly.

5.3 Outcome

- Can get experience in using Window Socket programming.
We find that the main function that we had used is very similar to that of BSD socket programming.
- We had learnt the TCP programming paradigm.
- Can get experience in using Visual C++ compiler.
Visual C++ compiler can help us to manage the source code in a project manner. We can set up a setting to link some library that we had used, that's like the "Makefile" in Unix.
- The flow of the Asynchronous mode of communication with event-driven programming paradigm is clearer now.

5.4 The pseudo-code of WinChat

5.4.1 Server side

```
Main()
{
    Initialize the windows attribute
    Loop
    {
        Dispatch the event
        For event "start listen"
            Initialize WinSock
            Create a socket for listen
        For event "stop listen"
            Close the socket
            Shutdown WinSock
        For event "FD_accept"
            Accept the new connection
            Save the information of the new client
        For event "FD_read"
            When a network message is arrived from a client,
            server starts to analysis the message and forward the
            message to the correspond client
    }
}
```

5.4.2 Client side

Main

```
{  
    Initialize the Windows attribute  
    User input her/his name  
    Loop  
    {  
        Dispatch the event  
        For event "Connect"  
            User input the IP address of the server  
            Initialize WinSock  
            Create a socket for connection  
            Starts to connect with it.  
        For event "Want to send message"  
            User needs to input the receiver's name and the  
            content message. Then send the message through the  
            socket connected with the server  
        For event "FD_read"  
            Receive the message from the network buffer  
            Display the message to the user  
        For event "Disconnect"  
            Disconnect from the server  
            Close socket  
            Shutdown WinSock  
    }  
}
```

6. Simple video player

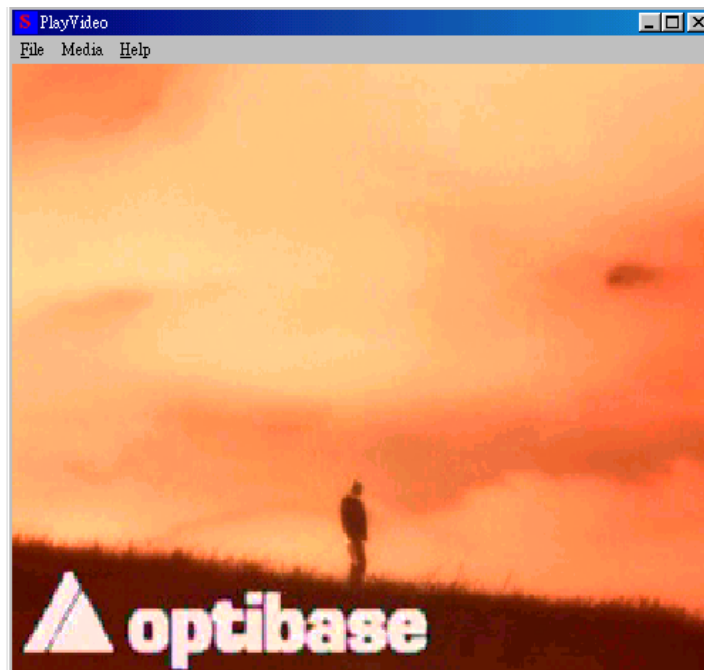


figure 6.1 Snapshot of video player

6.1 Introduction

Video player is a simple application using DirectShow Library. It supports a wide variety of formats, including Advanced Streaming Format (ASF), Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), MPEG Audio Layer-3 (MP3), and WAV files.

The video player supports function of pause and stop a video that is playing. It also supports resize of the window that is playing.

6.2 Aim of writing this program

- Get experience of using DirectShow
- Try to build a windows application using Visual C++
- Try to write a video player that can play different kind of video format

6.3 Filter graph of video player

Here is a filter graph whose purpose is to play back an MPEG-compressed video from a file would use the following filters.

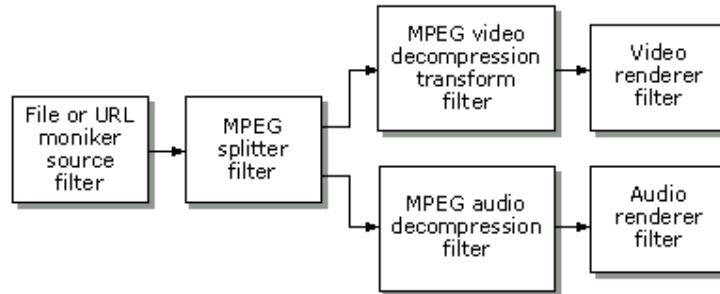


figure 6.2 Filter graph of video player

- A source filter to read the data off the disk.
- An MPEG filter to parse the stream and split the MPEG audio and video data streams.
- A transform filter to decompress the video data.
- A transform filter to decompress the audio data.
- A video renderer filter to display the video data on the screen.
- An audio renderer filter to send the audio to the sound card.

6.4 Outcome

- We get experience of using Visual C++ and DirectShow.
- We have successfully write a simple video player.
- However, we find that writing application using Windows procedure call is not sufficient on further development of our project. So we try to rewrite the video player using Object-Oriented approach and Microsoft Foundation Class (MFC).

6.5 The pseudo-code of video player

```
Main()
{
    Initialize COM
    Loop
    {
        For event (open video)
            Create COM Instance(Filter Graph)
            Query Interface(Media Control)
            Query Interface(Media Event)
            Query Interface(Video Window)
            Filter Graph(Render file)
            Video Window(Put video window)

        For event(play video)
            Media Control(Run)

        For event(pause video)
            Media Control(Pause)

        For event(stop video)
            Media Control(Stop)
            Video Window(Remove video window)

        For event(resize window)
            Video Window(resize)
    }
    UnInitialize COM
}
```


7. MFC Chatting

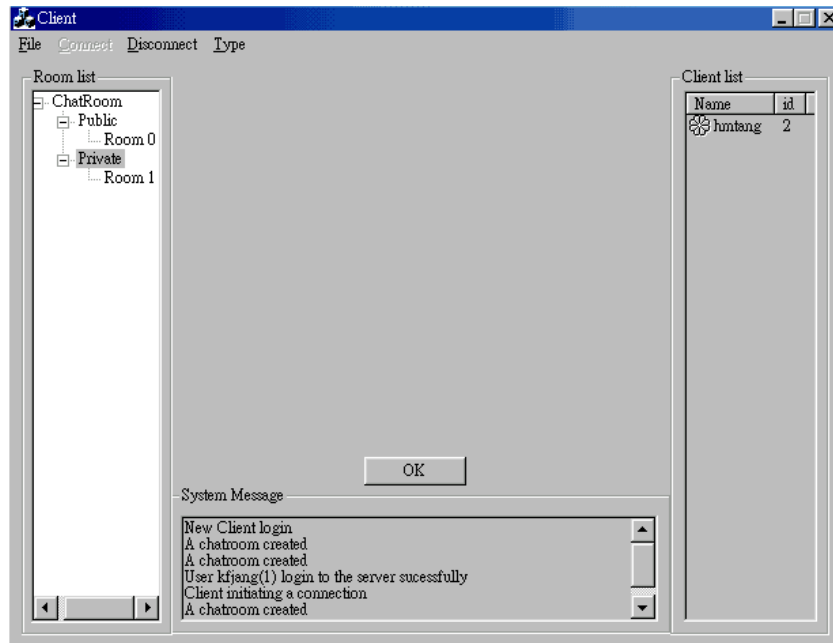


figure7.1 Snapshot

7.1 Introduction

MFC Chatting is a general communication application under Window OS. It works similar to the ICQ application. Client just need to login to the server and then will know who is "online" which mean that connected to the server. Also every client can send message to one of the client and can create chatroom for group discussion. This application is built based on Microsoft Foundation Class (MFC) using Visual C++ compiler.

7.2 Aim of writing this program

- Use Microsoft Foundation Class (MFC). MFC provide many data structure and function library, which can help us in programming.
- Start to write chatroom services. The previous program can only send messages, which is not quite interactive. Chatroom can provide communication with a more interactive way.
- Enhance the user interface. The user interface is not user-friendly for the WinChat. For window application programming, MFC is the most famous library to create a user-friendly interface.

7.3 New features

7.3.1 Server side

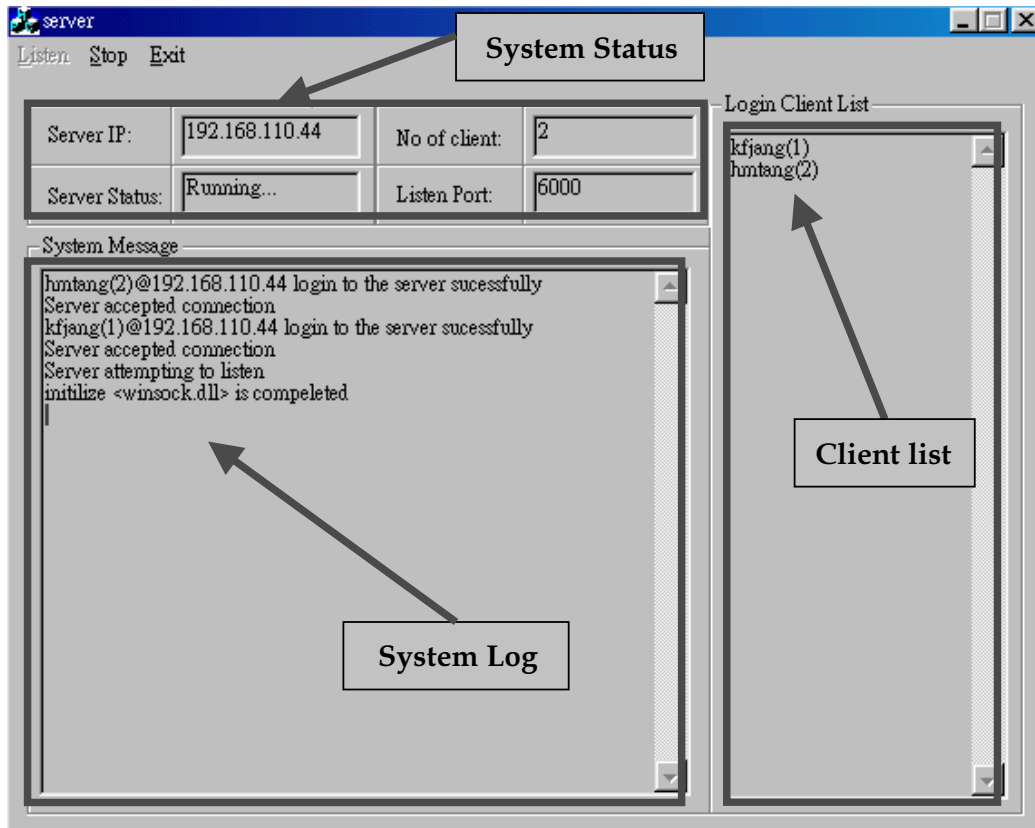


figure7.2 Snapshot of the server program

MFC Chatting server side is built based on Microsoft Foundation Class (MFC) using Visual C++ compiler. It had included more features than the WinChat. The new features are system status, system log and client list.

System status

- Server IP: show the IP address of the server.
- Server Status: show the server's status is running, stop or error.
- No. of client: show the total number of clients which had logged on.
- Listen Port: show the port number that the server is listening at.

Login Client List

A list that show the entire client name and its client id that are logged in the server.

System Message

This box is used to show all system log message, for example, client login/out, the type of error message of the server.

7.3.2 Client side

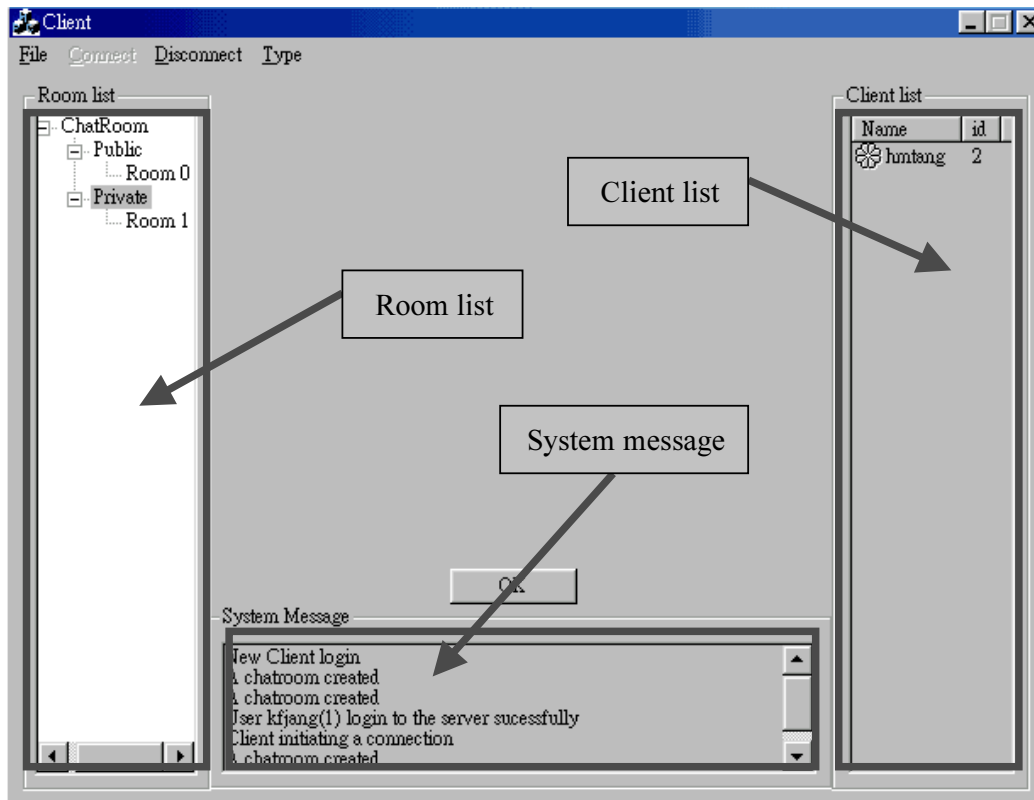


figure7.3 Snapshot of the client program

MFC Chatting client side is built based on Microsoft Foundation Class (MFC) using Visual C++ compiler. It had included more features than the WinChat. The new features are system message, chatroom services, room list and client list.

System Message

This box is used to show all system log message, for example, client login/out, the type of error message of the server.

Login Client List

A list that shows the entire client name and its client id in the system.

Room list

This box shows the entire chatroom that is currently in discussion.

Send Message



figure7.4 Snapshot of the send message box

By double click the mouse button on the client's name, then the send message box will pop up for user to send message to that client. This can be user-friendly.

Receive Message

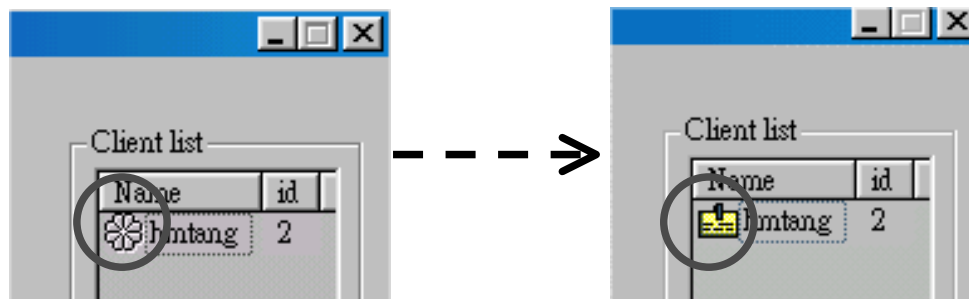


figure7.5 Snapshot of the receive message event

Receive message box will not pop up immediately when message is incoming as before. The status, which is in front of the client's name, will change. User can see the received message by double click the mouse button on the client's name when the user wants to see the message. This is one of the main different with the WinChat.

Chatroom

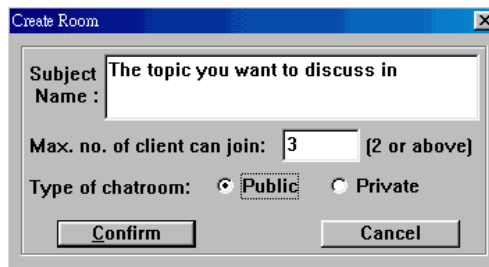


figure7.6 Snapshot of the create chatroom

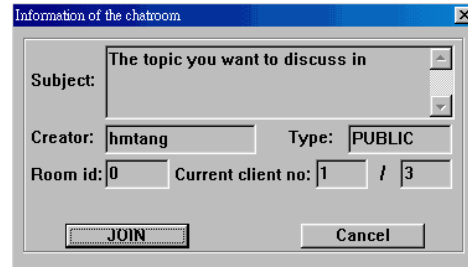


figure7.7 Snapshot of the chatroom information

In the create room box (fig6.6), client can type in the subject of that chatroom, maximum number of client that can join in that chatroom and the type of that chatroom. Other user can see the information of the chatroom (fig6.7) by double click on the room showed on the room list and user can choose to join the chatroom or not.

There are two type of chatroom in our application. One is the public and the other one is private. **Public chatroom** can let the entire client to join in, but **private chatroom** need the authorization of the creator. Client can type the reason for the request (fig6.8).

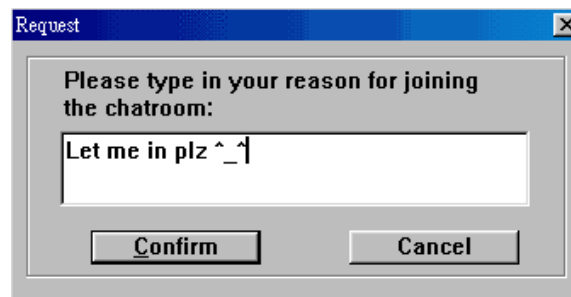


figure7.8 Snapshot of the request to join chatroom

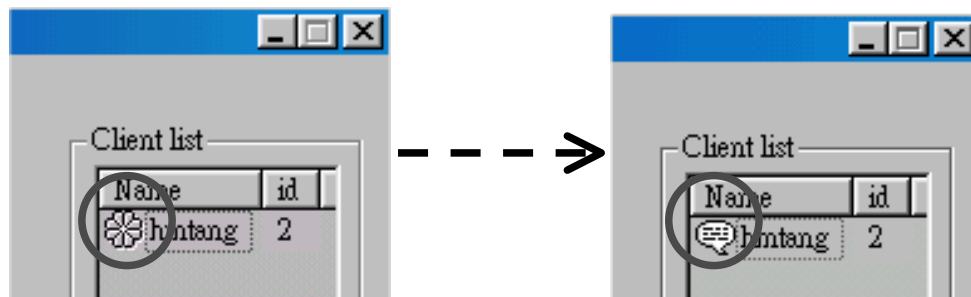


figure7.9 Snapshot of the request to join chatroom event

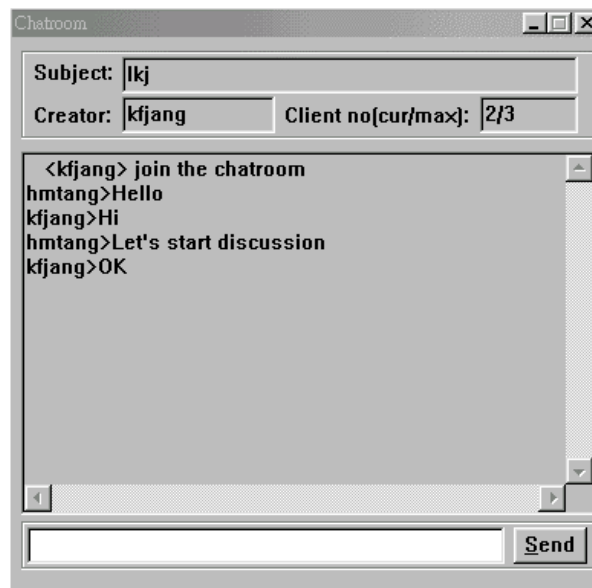


figure7.10 Snapshot of the chatroom

7.4 Outcome

- We have used MFC in our program now and it helps us to reduce the number of code that needed to code.
- Client can have the chatroom services to use. Our program can provide communication with a more interactive way for more than two clients.
- User interface become more attractive and user-friendly. The user can mainly use double mouse click to invoke the function provided.

7.5 The pseudo-code of MFC Chatting

7.5.1 Server side

```
Main()
{
    Initialize the windows attribute
    Loop
    {
        Dispatch the event
        For event "start listen"
            Initialize WinSock
            Create a socket for listen
        For event "stop listen"
            Close the socket
            Shutdown WinSock
        For event "FD_accept"
            Create a new socket and accept the new connection
            Save the information of the new client
            Add the client into the client lis
        For event "FD_read"
            When a network message is arrived from a client,
            server starts to analysis the message and forward the
            message to the correspond client
        For event "FD_close"
            Close the socket that message from
            Mutlicast to all the other client about the disconnection
    }
}
```

7.5.2 Client side

Main

```
{  
    Initialize the Windows attribute  
    User input her/his name  
    Loop  
    {  
        Dispatch the event  
        For event "Connect"  
            User input the IP address of the server  
            Initialize WinSock  
            Create a socket for connection  
            Starts to connect with it.  
        For event "Want to send message"  
            Input the content of message. Then send the message  
            through the socket connected with the server  
        For event " read the incoming message"  
            Load the message from the stored buffer  
            Display the message  
        For event "Create chatroom"  
            User needs to input the information about the chatroom  
            Send the information to the server  
        For event "FD_read"  
            Receive the message from the network buffer  
            Store the message to the buffer  
        For event "Disconnect"  
            Disconnect from the server  
            Close socket  
            Shutdown WinSock  
            Re-initialize the system state  
    }  
}
```


8. WinCap

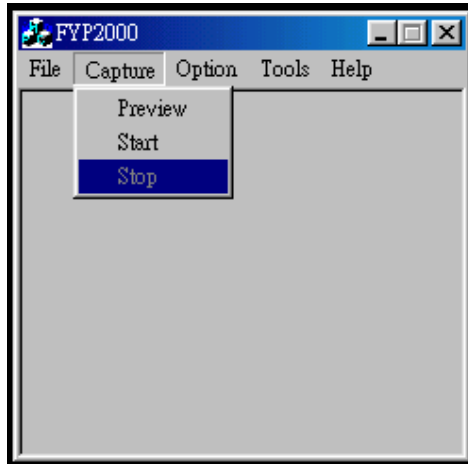


figure 8.1 Snapshot of capturing application

8.1 Introduction

WinCap is an application that can capture video using Windows Driver Model (WDM) devices or older Video for Windows devices. It can also capture audio as well. The video captured can be rendered to display the video data on the screen, or write the data into an AVI file. We have implemented a function to set the frame rate for capturing. So that the capturer can capture video in different frame rate to suits user's need.

WinCap is written using Microsoft Foundation Class (MFC). Besides, we have also rewrite the video player using Object-Oriented approach and MFC. We have embedded the video player into the capturer application so that we can capture video into a file, and then play it in our application.

8.2 Aim of writing this program

- Write a capturer application
- Use MFC to write an application
- Get more experience in using DirectShow
- Try to write software components that can plug into another application.

8.3 Filter graph of video capturer

Here is a filter graph of the video capturer application

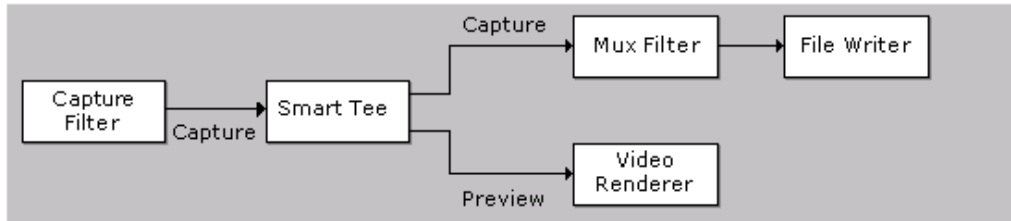


figure 8.2 Filter graph of video capturer

- One or more capture filter to capture audio or video data
- A smart tee filter to split the data into two streams (preview stream and capture stream respectively). The reason for using the smart tee filter is that some capture filter only provides one output only. In order to simultaneously capture and preview the video data, two outputs are required.
- One or more multiplexer filter to combine multiple input streams into one output stream
- One file writer filter to write the output stream into a file
- One video renderer filter to display the video data on screen

8.4 Outcome

- We get experiences in using MFC and using Object-Oriented approach.

This helps us to write application with better user interfaces.

- We have written the capturer and video player in two classes so that it can plug into any other application. This improves the scalability of our application.
- We have successfully written a capture application.

8.5 Prototype of Class CPlayVideo

Class CPlayVideo {

Public Method:

CPlayVideo(CWnd pParent);*
~CPlayVideo(CWnd pParent);*

Private Method

Create();
DestroyWindow();
DoDataExchange(CDataExchange pDX);*
GetVideoFileName(LPSTR szName);
OnInitDialog();
OnButtonOpen();
OnButtonPause();
OnButtonPlay();
OnButtonStop();
OnExit();
OnNotify();
OnSize(UINT nType, int cx, int cy);
}

8.6 Prototype of Class CCaptureVideo

Class CCaptureVideo {

Public Method:

CCaptureVideo();

~CCaptureVideo();

Private Method:

BuildFileWriting();

CaptureStart();

CaptureStop();

CreateComponent();

DisableMenu();

FreeCapFilter();

GetCaptureFileName(LPSTR szName);

*InitCapture(CDialog *mainDlg, RECT grc);*

isCapture();

isPreview();

Preview();

PreviewStart();

PreviewStop();

RenderCaptureStream();

RenderPreviewStream();

SelectDevice();

SetFrameRate();

}

8.7 The pseudo-code of video player

```
Main()
{
    Initialize COM
    Create COM Instance(Filter Graph)
    Create Device Enumerator
    Select Suitable Device
    Loop
    {
        For event "Preview"
            Render Preview Graph
            Start Preview
            For event (Stop Preview)
                Stop Preview
        For event "Capture"
            Check Is Preview
            If (is Preview)
                Stop Preview
            Render Capture Graph
            Get Capture File Name
            Start Capture
        For event "Stop Capture"
            Stop Capture
            Check is Preview Before
            If (is Preview)
                Render Preview Graph
                Start Preview
    }
    UnInitialize COM
}
```

9. Integrated program



figure9.1 Snapshot of the program

9.1 Introduction

Integrated program is the enhanced version of the MFC Chatting. We combine the MFC Chatting and WinCap into this application. Client can use the web camera to preview and capture their video for further usage. This is the first step of our project. It can be further developed to record the lecture into video and store into the video database. Student can get the video for study and for reference. Also, it can help lecturer to improve the supplement material that the lecture haven't been covered.

9.2 Aim of this integrated program

- To integrate what we had done in this semester.
- Try to use the captured video to send to other client by the MFC Chatting.

9.3 Outcome

- We had success to integrate the MFC Chatting and Wincap into this program.
- In this program, we haven't implement streaming the captured video. Because we still study the streaming related information in progress.

10. Problem facing

10.1 Problems

At this stage, we can successfully implement a chat-room, video capturer and player for our virtual classroom. However, this is not enough. We have to real-time capture video and transfer the video to all the clients connected to the server. Therefore, teachers can present their lecture to remote students.

Using DirectShow capture video and play video well in a local machine. But it does not provide a filter that supports video capturing and real-time transmission of the video across the network. To solve this problem, we have to implement our own filters.

Two filters have to implement:

1. Filter which capture real-time video, real-time compress the video and transmit the video to others.
2. Filter which real-time receive the video from remote computer and render the video immediately. The video data should not be saved on the local hard disk.

Difficulties of writing these filters

1. Writing a filter requires understanding of COM programming
2. No example on writing filter for network purpose
3. DirectShow does not provide any documentation on implementing such filter.
4. DirectShow 's documentation is difficult to understand.
5. Compilation of filter using Visual C++ is difficult

We have tried to study the filter development and understand the COM programming. However, we still not understand the filter development process thoroughly. Besides, we found it difficult to compile a typical sample filter either.

We still not finish in writing the DirectShow filters to suit our needs.

10.2 Solution

After thoroughly consideration, we have decided to find another way to achieve our purpose of transmission of video through network.

10.2.1 Java Media Framework (JAVA JMF)

The Java Media Framework (JMF) is an application-programming interface (API) for incorporating time-based media into Java applications and applets. It provides supports for

- Capturing and storing media data
- Controlling the type of processing that is performed during playback
- Performing custom processing on media data streams.
- Streaming and conferencing applications
- Providing access to raw media data
- Enable the development of custom, downloadable demultiplexers, codecs, effects processors, multiplexers, and renderers (JMF *plug-ins*)

10.2.2 Why using JMF to solve the problem?

JMF support **Real-Time Transport Protocol (RTP)** as a solution to send or receive a live media broadcast or construct a videoconference over the Internet or intranet.

Streaming Media

When media content is streamed to a client in real-time, the client can begin to play the stream without having to wait for the complete stream to download. In fact, the stream might not even have a predefined duration downloading the entire stream before playing it would be impossible. The term streaming media is often used to refer to both this technique of delivering content over the network in real-time and the real-time media content that's delivered.

Streaming media is everywhere you look on the web-live radio and

television broadcasts and webcast concerts and events are being offered by a rapidly growing number of web portals, and it's now possible to conduct audio and video conferences over the Internet. By enabling the delivery of dynamic, interactive media content across the network, streaming media is changing the way people communicate and access information.

Real-Time Transport Protocol

RTP provides end-to-end network delivery services for the transmission of real-time data. RTP is network and transport-protocol independent, though it is often used over UDP.

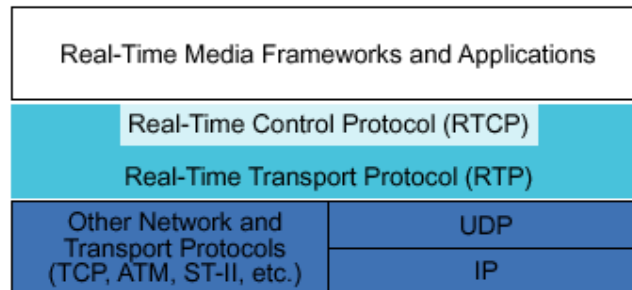


figure10.1 RTP architecture

RTP can be used over both unicast and multicast network services. Over a *unicast* network service, separate copies of the data are sent from the source to each destination. Over a *multicast* network service, the data is sent from the source only once and the network is responsible for transmitting the data to multiple locations. Multicasting is more efficient for many multimedia applications, such as videoconferences. The standard Internet Protocol (IP) supports multicasting.

RTP Applications

RTP applications are often divided into those that need to be able to receive data from the network (RTP Clients) and those that need to be able to transmit data across the network (RTP Servers). Some applications do both -- for example, conferencing applications capture and transmit data at the same time that they're receiving data from the network.

Receiving Media Streams From the Network

Being able to receive RTP streams is necessary for several types of applications. For example:

- Conferencing applications need to be able to receive a media stream from an RTP session and render it on the console.
- A telephone answering machine application needs to be able to receive a media stream from an RTP session and store it in a file.
- An application that records a conversation or conference must be able to receive a media stream from an RTP session and both render it on the console and store it in a file.

Transmitting Media Streams Across the Network

RTP server applications transmit captured or stored media streams across the network.

For example, in a conferencing application, a media stream might be captured from a video camera and sent out on one or more RTP sessions. The media streams might be encoded in multiple media formats and sent out on several RTP sessions for conferencing with heterogeneous receivers. Multiparty conferencing could be implemented without IP multicast by using multiple unicast RTP sessions.

11. Conclusion and Future Work

This final year project hopes to develop an application that enables students to have an alternative way of learning and communication. We want to provide them with a cyber campus so that students can learn and join discussion over Internet

In this semester, we have written applications including chatroom, video player and capturer. We have integrated all applications into one application. However, we face difficulties on further develop our application to include video conferencing.

Up to this moment, we still find ways to solve the problem. One solution we had found is to use Java and Java JMF. And we are now trying to write applications using Java JMF.

In the future, we will try to implement:

1. Video streaming

Students can view the captured video file of a lecture through Internet. In case students miss a lesson, they can still attend the lesson through watching the video. So they will not miss any details of a lecture.

2. One-to-one video conference

Chatting with others may sometimes can't express one's idea thoroughly. If our application includes a one-to-one videoconference, students can communicate with others more directly and can express their idea efficiently.

3. One-to-Many conference

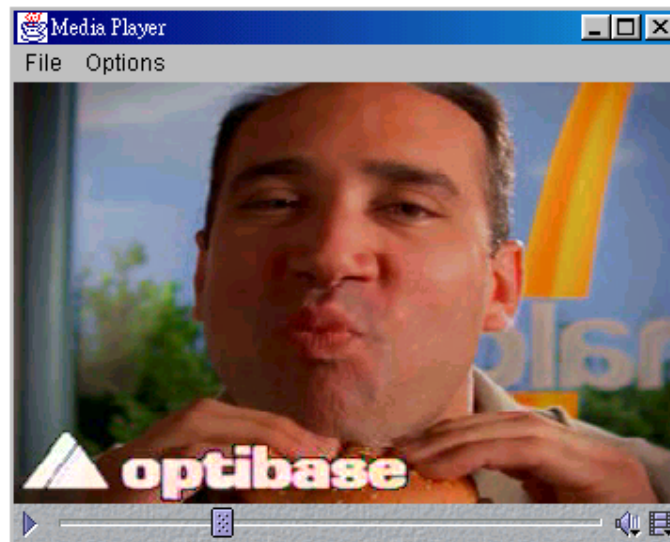
Teachers can present lecture real-time through Internet. Students can no longer attend lessons inside a classroom but in a virtual classroom. Students can learn without restriction of place anymore.

Appendix A : References

1. MSDN Online
<http://msdn.microsoft.com/default.asp>
2. Microsoft DirectX 8.0
<http://www.microsoft.com/directx/>
3. WinSocket
<http://www.sockets.com/>
4. Microsoft NetMeeting Product Home Page
<http://www.microsoft.com/windows/netmeeting/>
5. Microsoft NetMeeting Software Developers' Kit
<http://www.microsoft.com/windows/netmeeting/authors/sdk/default.asp>
6. Home Page fro Windows Media Player
<http://www.microsoft.com/windows/mediaplayer/en/default.asp>
7. Windows Media Format 7 SDK
[http://msdn.microsoft.com/library/psdk/wm_media/wmform/htm/introducin
gwindowsmediaformat.htm](http://msdn.microsoft.com/library/psdk/wm_media/wmform/htm/introducin
gwindowsmediaformat.htm)
8. The Source for the Java™ Technology
<http://www.java.sun.com/>
9. Java JMF
<http://java.sun.com/products/java-media/jmf/index.html>
10. JMF Solutions : Using JMF in Swing
<http://java.sun.com/products/java-media/jmf/2.1/solutions/SwingJMF.html>
11. Video compression, videoconferencing, and image compression
<http://www-nt.e-technik.uni-erlangen.de/%7Ehartung/links.html>
12. Planning and Implementing Wireless LANS
<http://www.networkcomputing.com/netdesign/wlan1.html>
13. Mobile Computing related resources on WWW
<http://www.cs.umd.edu/projects/mcml/mcothers.html>
14. A Short Tutorial on Wireless LANs and IEEE 802.11
Daniel L. Lough, T. Keith Blankenship, Kevin J. Krizman
<http://computer.org/students/looking/summer97/ieee802.htm>
15. Managing Bandwidth: Deploying QOS In Enterprise Networks, 1/e
Alistair A. Croll, Ontario, Canada Eric Packman
http://www.phptr.com/ptrbooks/ptr_0130113913.html
16. RSVP Standards Resources
<http://www.cs.columbia.edu/%7Ehgs/internet/rsvp.html>

17. ISDN Videoconferencing
http://alumni.caltech.edu/%7Edank/isdn/isdn_ai.html - VIDEO
18. The Network Time Protocol (NTP) Distribution
http://www.eecis.udel.edu/%7Entp/ntp_spool/html/index.htm
19. Internet Phone Web Guide
http://www.vocaltec.com/consumer/products/iphone_5/guide/chap4.htm
20. Windows Media Content Development and Deployment
<http://msdsn.microsoft.com/workshop/imedia/windowsmedia/default.asp>
21. Virtual Classroom[®]
http://www.njit.edu/Virtual_Classroom/
22. The Virtual Classroom and Virtual University at New Jersey Institute of Technology
<http://eies.njit.edu/~hiltz/CRProject/sloansum1.html>
23. Designing a Virtual Classroom
http://www.njit.edu/Virtual_Classroom/Papers/Design.html
24. Teaching in the Virtual Classroom
http://www.njit.edu/Virtual_Classroom/Papers/Teaching.html
25. Learning Networks: A Field Guide to Teaching and Learning
<http://www-mitpress.mit.edu/mitp/recent-books/comp/learning-networks.html>
26. The Virtual Classroom: Learning Without Limits Via Computer Network
<http://eies.njit.edu/~hiltz/RBooks/ablex1.html>
27. Bob Quinn, Dave Shute. Windows sockets network programming. Addison Wesley Pub. Co., c1996.
28. TCP/IP Illustrated Volume 1: The Protocols. (Addison Wesley Professional Computing) by W. Richard Stevens. Hardcover
29. TCP/IP Illustrated, Volume 2: The Implementation (Addison-Wesley Professional Computing) by Gary R. Wright(Contributor), et al. Hardcover
30. TCP/IP Illustrated, Volume 3: TCP for Transactions, Http, Nntp, and the Unix Domain Protocols (Addison-Wesley Professional Computing Series) by W. Richard Stevens. Hardcover
31. Understanding ActiveX and OLE by Chappell, Microsoft Press
32. Java in a NutShell by David Flanagan, O'Reilly & Association Inc.
33. Java Network Programming, 2nd Edition by Elliotte Rusty Harold, O'Reilly & Association Inc.
34. The Java Tutorial Second Edition Object-Oriented Programming for the Internet (Java Series) by Mary Campione, Kathy Walrath.

Appendix B : JavaPlayer



figureB.1 Snapshot of JavaPlayer

B.1 Introduction

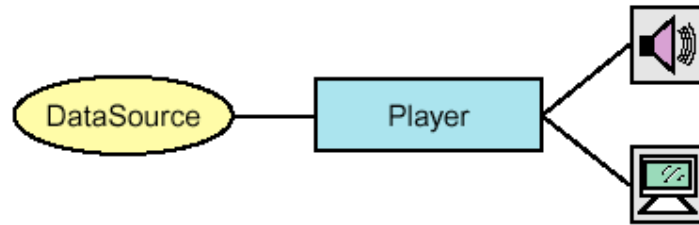
JavaPlayer is a simple video player which is a simple application using Java JMF. It supports a wide variety of formats, including Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI). It also supports function of pause and stop a video that is playing.

B.2 Aim of writing this program

- Learn how to write application with Java as we haven't learnt Java before.
- Understanding underlying working principle of Java Media Framework (JMF)

B.3 JMF Player Model

A Player processes an input stream of media data and renders it at a precise time. A DataSource is used to deliver the input media-stream to the Player. The rendering destination depends on the type of media being presented.

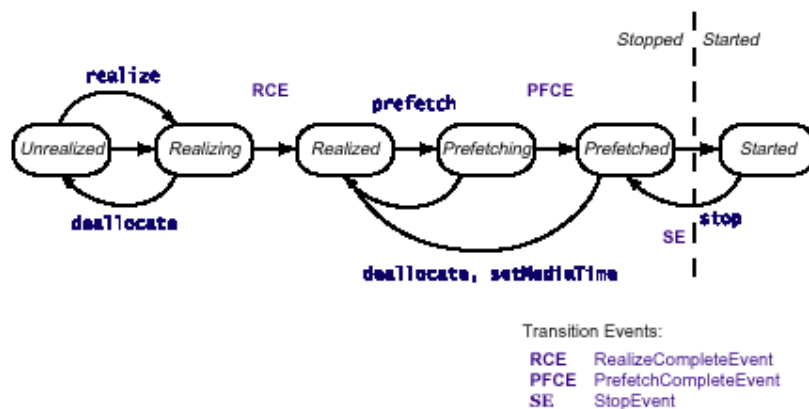


figureB.2 JMF Player Model

We found that JMF's Player model is quite similar to DirectShow's filter graph. Both involve a datasource and a renderer. But JMF's Player Model is much easier to understand as it abstract some details between the source and the renderer such as the transformation from one format to other format, the multiplexer to split the input streams into video and audio stream.

B.4 Player States

A Player can be in one of six states. The Clock interface defines the two primary states: Stopped and Started. To facilitate resource management, Controller breaks the Stopped state down into five standby states: Unrealized, Realizing, Realized, Prefetching, and Prefetched.



figureB.3 Player states

In normal operation, a Player steps through each state until it reaches the *Started* state in order to play a video file.

B.5 OutCome

- We get experience in writing a Java application
- We understand more about JMF's principle, this helps us to further development other application.
- We have successfully written a video player

B.6 The pseudo-code of JavaPlayer

```
Main()
{
    create a Player
    block until the player is realized
    display the visual component
    display the control panel component
    Loop
    {
        For event "Play Video"
            prefetching a player
            block until the player is prefetched
            start the presentation
        For event "Stop Video"
            stop the presentation
        For event "Autoloop"
            Toggle the autoloop states
    }
}
```

Appendix C : JavaCaturer



figure C.1 Snapshot of JavaCaturer

C.1 Introduction

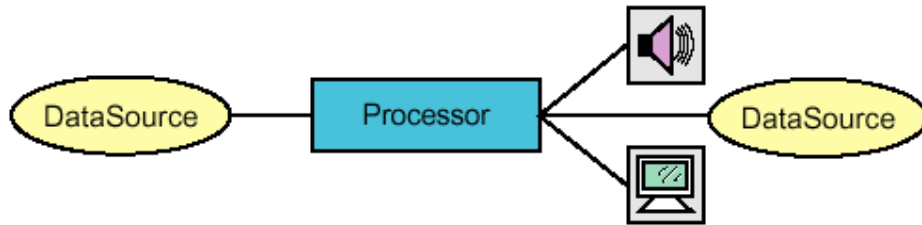
JavaCaturer is an application that can capture the video using Windows Driver Model (WDM) devices. The capturer can only capture video now. The video captured can be rendered to display the video data on the screen, or write the video data into an AVI file.

C.2 Aim of writing this program

- We get more experience in writing a Java application
- Try to use processor rather than player to create an application

C.3 JMF Processor Model

A Processor is a Player that takes a DataSource as input, performs some user-defined processing on the media data, and then outputs the processed media data.

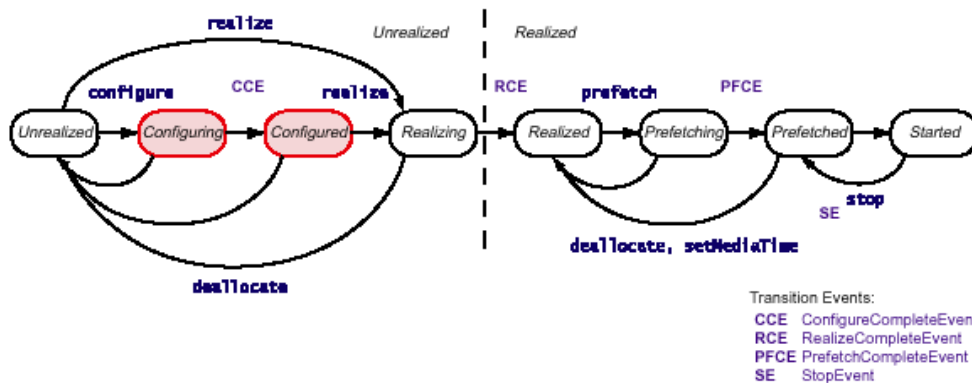


figureC.2 JMF processor model

A Processor can send the output data to a presentation device or to a DataSource. If the data is sent to a DataSource, that DataSource can be used as the input to another Player or Processor, or as the input to a DataSink. (Can write the video into a file) While the processing performed by a Player is predefined by the implementor, a Processor allows the application developer to define the type of processing that is applied to the media data. This enables the application of effects, mixing, and compositing in real-time.

C.4 Processor State

A Processor has two additional standby states, Configuring and Configured, which occur before the Processor enters the Realizing state.



figureC.3 Processor state


C.5 Outcome

- We have successfully used a processor to write an application. The processor can get video from video camera as a data source and can write the video file into an AVI file through a data sink.
- However, the processor now can only capture one data source only. Audio source hasn't been captured. We will further improved the application later.
- And we will also try to compress the video file into another format such as (MPEG)

C.6 The pseudo-code of JavaCapturer

```
Main()
{
    Create List of Device
    Select Device from the list
    Get the output format of the device
    Set output type of the Processor
    Create a new Processor
    Initialize the Processor
Loop()
(
    For event "Capture Video"
        Initialize capture
        If (is Preview)
            Stop Preview
            Reconfigure Processor
        Create DataSink
        Start Capture
    For event "Preview Video"
        Display the visual componenet
        Display the control components
        Start Preview
)
)
```

Appendix D: Progress Report

Date	Description
Start at 12 th June	Studying BSD Trying Netmeeting & Internet Phone
Start at 26 th June 3 rd July	Studying WinSock BSD UDP chatroom finished
17 th July	 WinChat finished
Start at 17 th July	Studying DirectShow
7 th August	 Simple Video Player finished
Start at 7 th August	Studying MFC
11 th September	 MFC Chatting finished
18 th September	 WinCap finished
2 nd October	 Integrated program finished
Start at 2 nd October	Studying DirectShow Filter
Start at 16 th October	Studying Java and Java JMF
30 th October	Java Applet (AVI and MPEG file type supported)
6 th November	 JavaPlayer finished
13 th November	 JavaCap finished