โปรแกรมประยุกต์สำหรับเครือข่าย ATM

APPLICATION FOR ATM NETWORK

โดย

นาย วิชิต วัฒนไพลิน

นาย เสกสรร อัศวานุชิต

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

โปรแกรมประยุกต์สำหรับเครือข่าย ATM

APPLICATION FOR ATM NETWORK

โดย

นาย วิชิต วัฒนไพลิน 38014457

นาย เสกสรร อัศวานุชิต 38014602

อาจารย์ที่ปรึกษา

อาจารย์ ธนา หงษ์สุวรรณ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชา วิศวกรรมศาสตร์คอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมประยุกต์สำหรับเครือข่าย ATM

(APPLICATION FOR ATM NETWORK)

ผู้จัดทำ นาย วิชิต วัฒนไพลิน รหัสประจำตัว 38014457

นาย เสกสรร อัศวานุชิต รหัสประจำตัว 38014602

..... อาจารย์ที่ปรึกษา

(อาจารย์ ธนา หงษ์สุวรรณ)

โปรแกรมประยุกต์สำหรับเครือข่าย ATM

นาย วิชิต วัฒนไพลิน
นาย เสกสรร อัศวานุชิต
อาจารย์ ธนา หงษ์สุวรรณ อาจารย์ที่ปรึกษา
ปีการศึกษา 2541

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้นำเสนอโครงงานวิจัยเพื่อศึกษาการทำงานของระบบเครือข่าย ATM (Asynchronous Transfer Mode) ,การทำ LAN Emulation และ IP over ATM บนเครือข่าย ATM จากนั้นจะทำการพัฒนาแอพพลิเคชั่นที่จะนำมาใช้งานบนระบบเครือข่าย ATM โดยเฉพาะ เพื่อให้สามารถมองเห็นการทำงานของระบบเครือข่าย ATM ได้ชัดเจนยิ่งขึ้น และแอพพลิเคชั่นนี้จะนำเอาคุณลักษณะต่างๆที่มีอยู่ในระบบเครือข่าย ATM มาใช้เพื่อแสดงให้เห็นถึงประสิทธิภาพของระบบเครือข่าย ATM

สำหรับแอพพลิเคชั่นที่ทำการพัฒนาขึ้น จะสามารถรับและส่งสตรีมของวีดีโอได้แบบเรียลไทม์ โดยจะมีเซิร์ฟเวอร์ซึ่งทำหน้าที่ในการส่งสตรีมของวีดีโอไปให้กับไคลเอนต์ ซึ่งผู้ใช้จะสามารถดูภาพวีดีโอได้จากไคลเอนต์ และสามารถที่จะหยุดหรือเล่นภาพวีดีโอได้ตามต้องการ

APPLICATION FOR ATM NETWORK

Mr. Vichit Wattanapailin

Mr. Sekson Asawanuchit

Mr. Thana Hongsuwan Advisor

1998

Abstact

This thesis is presenting the study of ATM Network, LAN Emulation, and IP over ATM on ATM Network. Then, the application, especially develop to be use on ATM Network, intends to emphasize the functioning of ATM Network. The application suggests all the effective aspects of the ATM Network.

The developed application is designed to send and receive video stream in real time mode. This process is carried out by the server-client computer relationship. Server computer send the video stream to client computers, and the users can view or stop playing the video stream from their computer.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี ก็ด้วยความช่วยเหลือ และร่วมมือจากหลายๆฝ่ายด้วยกัน บุคคลแรกที่ต้องขอขอบพระคุณเป็นอย่างมาก คือ อาจารย์ ธนา หงษ์สุวรรณ อาจารย์ที่ปรึกษาในการทำปริญญานิพนธ์ ที่คอยให้ความช่วยเหลือ แนะนำ และคำปรึกษาในการทำโครงงานวิจัยครั้งนี้

ขอขอบคุณ คุณ ธานี ชวศิริกุลฑล จากบริษัท MADGE ประเทศไทย ที่ได้คอยช่วยเหลือ ในการจัดหาอุปกรณ์ และให้คำปรึกษา

ขอขอบคุณ คุณ อัครเดช วัชระภูพงษ์ (พี่ADEK)ที่ได้คอยช่วยเหลือในการจัดหาอุปกรณ์ ให้คำปรึกษา และสนับสนุนในการทำงาน เสมอมา

ขอขอบคุณ คุณ โอฬาร จากบริษัท ATML ที่ได้คอยช่วยเหลือในการจัดหาอุปกรณ์

ขอขอบคุณ พี่กิตติ ที่ช่วยเหลือด้านอุปกรณ์ และช่วยแก้ไขปัญหาเกี่ยวกับอุปกรณ์

ขอขอบคุณ เพื่อนๆที่รักทุกท่าน ที่ได้คอยให้กำลังใจ และความเพลิดเพลินในการทำงาน

ท้ายสุดนี้ขอขอบพระคุณ บิดามารดาและคุณครูบาอาจารย์ทุกท่าน ผู้เป็นที่รักและเคารพอย่างยิ่งที่ได้เลี้ยงดูอบรม และทำให้ผู้เขียนได้มีวันนี้ ผู้เขียนขอระลึกในพระคุณ และขอกราบขอบพระคุณมา ณ ที่นี้

วิชิต วัฒนไพลิน

เสกสรร อัศวานุชิต

สารบัญ

สารบัญภาพ

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ระบบเครือข่าย ATM เป็นเทคโนโลยีที่ใช้โปรโตคอล Asynchronous Transfer mode (ATM) เป็นมาตราฐานในการส่งข้อมูล โดยข้อมูลที่ส่งในระบบเครือข่าย ATM จะถูกแบ่งเป็นเซลล์(cell) ที่มีขนาดคงที่ทำให้สามารถสวิตซ์ (switch) ด้วยความเร็วที่สูงมาก และระบบเครือข่าย ATM มีความสามารถในการส่งข้อมูลได้หลายชนิดทั้งภาพ,เสียง,ข้อมูล โดยที่ระบบเครือข่ายจะมีความสามารถในการกำหนดคุณภาพของการบริการในการส่งข้อมูล (Quality of Service)ได้ แต่เนื่องจาก ATM เป็นเทคโนโลยีที่ใหม่มาก ดังนั้นอุปกรณ์ที่ใช้สำหรับต่อเข้ากับ ATM สวิตซ์ (ATM Switch)และการส่งข้อมูลโดยใช้ ATM โดยตรงนั้นยังมีไม่มาก และราคาแพง ทำให้มีการคิดค้นการทำ LAN Emulation และ IP over ATM ขึ้นมาเพื่อให้ผู้ใช้สามารถนำแอพพลิเคชั่นต่างๆที่ทำงานบนระบบเครือข่าย TCP/IP หรืออีเทอร์เน็ต (Ethernet) สามารถนำมาใช้งานในระบบเครือข่าย ATM ได้

การนำแอพพลิเคชั่นที่ใช้งานบนระบบเครือข่าย TCP/IP หรืออีเทอร์เน็ต มาใช้บนเครือข่าย ATM โดยการทำ IP over ATM หรือ LAN Emulation นั้น จะทำให้เกิดโอเวอร์เฮด(overhead) ขึ้นเพราะต้องมีการแปลง IP หรือ อีเทอร์เน็ตเฟรม (Ethernet Frame) ไปเป็น ATM เซลล์ ซึ่งจะทำให้การทำงานซับซ้อนขึ้นเป็นผลทำให้ประสิทธิภาพที่ได้รับจากระบบเครือข่าย ATM ลดลง และทำให้ไม่สามารถใช้คุณสมบัติบางอย่างของระบบเครือข่าย ATM ได้ ดังนั้นหากแอพพลิเคชั่นที่นำมาใช้สามารถทำงานบนระบบเครือข่าย ATM ได้เลยโดยไม่ต้องผ่านการทำ LAN Emulation หรือ IP over ATM จะทำให้สามารถใช้ระบบเครือข่าย ATM ได้อย่างมีประสิทธิภาพ

1.2 วัตถุประสงค์ของโครงงานวิจัย

1. เพื่อศึกษาการทำงานของระบบเครือข่าย ATM
2. เพื่อศึกษาการนำเอาแอพลิเคชั่นที่ทำงานบนระบบเครือข่ายที่ใช้กันทั่วไปมาใช้บนระบบเครือข่าย ATM โดยการทำ LAN Emulation หรือ IP over ATM
3. เพื่อศึกษาการเขียนแอพพลิเคชั่น ที่สามารถทำงานบนระบบเครือข่าย ATM โดยเฉพาะ (Native ATM application) และสามารถนำเอาคุณสมบัติต่างๆที่มีอยู่ในระบบเครือข่าย ATM มาใช้เพื่อแสดงให้เห็นถึงประสิทธิภาพของระบบเครือข่าย ATM

1.3 ขอบเขตของงานวิจัย

โครงงานวิจัยนี้จะทำการศึกษาการทำงานของระบบเครือข่าย ATM โดยจะมุ่งเน้นศึกษาในส่วนของการส่งผ่านข้อมูลบนเครือข่าย ATM เพื่อใช้ในการพัฒนาแอพพลิเคชั่นในการส่งวีดีโอสตรีมจาก

เซิร์ฟเวอร์ไปยังไคลเอนต์ โดยไคลเอนต์สามารถแสดงผลภาพวีดีโอได้แบบเรียลไทม์ (Real time) ทำให้ไม่จำเป็นที่จะต้องดาวน์โหลดไฟล์วีดีโอไปทั้งหมดก่อนแล้วจึงค่อยทำการแสดงภาพวีดีโอได้ ซึ่งจะเห็นได้ว่าการทำงานเพื่อส่งวีดีโอสตรีมแบบเรียลไทม์นั้น จำเป็นที่จะต้องใช้เครือข่ายที่มีความเร็วสูง ทำให้เหมาะสมที่จะนำเครือข่าย ATM มาใช้ในการพัฒนาแอพพลิเคชั่นประเภทนี้ โดยแอพพลิเคชั่นที่ทำการพัฒนาขึ้นนี้ จะเป็นการแสดงให้เห็นถึงประสิทธิภาพของเครือข่าย ATM ที่มีความเร็วในการส่งข้อมูลสูง และสามารถกำหนดคุณภาพของการบริการในการส่งข้อมูลได้

การพัฒนาแอพพลิเคชั่นในโครงงานวิจัยนี้จะใช้เครื่องมือ (Tool) ช่วยในการพัฒนาดังต่อไปนี้

1. Microsoft Visual C++ 6.0
   เป็นคอมไพเลอร์ที่ใช้ในการเขียนโปรแกรม
2. Winsock 2.0
   เป็นแอพพลิเคชั่นโปรแกรมอินเตอร์เฟส (Application Program Interface) ที่ใช้ช่วยในการเขียนแอพพลิเคชั่นในส่วนสร้างการติดต่า
3. DirectShow SDK
   เป็นแอพพลิเคชั่นโปรแกรมอินเตอร์เฟส ที่ใช้ช่วยในการเขียนส่วนรับสตรีมของวีดีโอ และแสดงผลเป็นภาพวีดีโอ

สำหรับรายละเอียดของ Winsock 2 0 และ DirectShow SDK จะมีอธิบายเพิ่มเติมในบทที่ 4 และ บทที่ 5

1.4 วิธีการดำเนินงาน

งานวิจัยในโครงงานนี้จะเริ่มต้นด้วยการศึกษาทฤษฎีต่างๆ ที่เกี่ยวข้องและจำเป็นต่องานวิจัย ซึ่งก็มีเรื่องหลักๆอยู่ 3 เรื่อง ดังต่อไปนี้

1. การส่งข้อมูลบนเครือข่าย ATM
2. การใช้งาน Winsock 2.0 ในการเขียนโปรแกรมติดต่อกับเครือข่าย ATM
3. การใช้งาน DirectShow SDK ในการเขียนโปรแกรมรับสตรีมของวีดีโอ และแสดงผลเป็นภาพวีดีโอ

เมื่อทำการศึกษารายละเอียด และการใช้งานต่างๆครบแล้ว จึงนำเอาความรู้ที่ได้มาออกแบบ และสร้างแอพพลิเคชั่นในการส่งวีดีโอสตรีมของเครื่องเซิร์ฟเวอร์ไปยังไคลเอนต์ โดยไคลเอนต์สามารถแสดงผลภาพวีดีโอได้แบบเรียลไทม์ โดยสามารถหยุดหรือเล่นภาพวีดีโอได้ตามต้องการ

บทที่ 2
ความรู้พื้นฐานเกี่ยวกับระบบเครือข่าย ATM

ระบบเครือข่าย ATM จะใช้โปรโตคอล ATM เป็นมาตรฐานการส่งข้อมูลความเร็วสูง โดย ATM ถูกพัฒนามาเพื่อให้ใช้กับงานที่มีลักษณะข้อมูลหลายรูปแบบ และต้องการความเร็วในการส่งข้อมูลสูงมากๆ สื่อที่ใช้ในเครือข่ายมีได้ตั้งแต่สายไฟเบอร์ออปติก (Fiber optic) สายโคแอกเชียล (Coaxial) หรือสายคู่ตีเกลียว (Twisted pair) มีความเร็วในการส่งข้อมูลได้ตั้งแต่ 25 เมกะบิต ไปจนถึง 622 เมกะบิต ATM ถูกพัฒนามาจากเครือข่ายแพคเก็ตสวิตซ์ชิ่ง (Packet-switching) ซึ่งจะแบ่งข้อมูลที่จะส่งออกเป็นหน่วยย่อยๆเรียกว่าแพคเก็ต(packet) ที่มีขนาดเล็กและคงที่แล้วจึงส่งแต่ละแพคเก็ตออกไป แล้วนำมาประกอบรวมกันเป็นข้อมูลเดิมอีกครั้งที่ปลายทาง ข้อดีของ ATM คือสามารถใช้กับข้อมูลได้หลายรูปแบบ เช่น เสียง, ภาพเคลื่อนไหว หรือข้อมูลคอมพิวเตอร์ ได้อย่างมีประสิทธิภาพ, มีความเร็วของข้อมูลสูง และสามารถกำหนดคุณภาพของบริการในการส่งข้อมูล

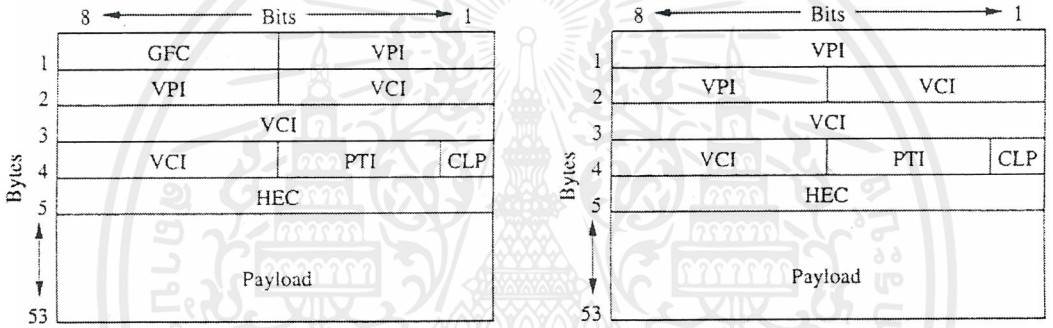
2.1 ลักษณะการทำงานของ ATM

ระบบเครือข่าย ATM มีรูปแบบมาตรฐานการส่งข้อมูลความเร็วสูง ซึ่งถูกพัฒนามาเพื่อรองรับงานที่ต้องการความเร็วในการส่งข้อมูลสูงมากๆ ข้อมูลที่ส่งในระบบเครือข่าย ATM จะถูกแบ่งเป็นกลุ่มย่อยเล็ก ๆ เรียกว่า เซลล์ (Cell) ซึ่งมีขนาดคงที่ 53 ไบต์ ประกอบด้วยส่วนข้อมูล (payload) ขนาด 48 ไบต์ และส่วนหัว(header) ขนาด 5 ไบต์ โดยรูปแบบของส่วนหัวจะมี 2 แบบ คือ user-to-network-interface (UNI) ซึ่งจะเป็นส่วนที่ใช้เชื่อมต่อผู้ใช้(user) กับ ATM สวิตซ์และ network-to-network-interface (NNI) ซึ่งจะเป็นส่วนที่ใช้เชื่อมต่อ ATM สวิตซ์กับ ATM สวิตซ์ ดังแสดงในรูปที่ 2.1 รายละเอียดส่วนหัวจะเก็บข้อมูลที่จำเป็นต่างๆที่ใช้ในการควบคุมการส่งดังนี้

- **GFC** (Generic Flow Control) ใช้สำหรับควบการไหลของข้อมูล สำหรับการเชื่อมต่อแบบ UNI
- **VPI/VCI** (Virtual Path Identifier และ Virtual Channel Identifier) ทำหน้าที่กำหนดวงจรเสมือน (virtual circuit) ในการเดินทางให้กับเซลล์นั้น
- **PT** (Payload Type) เป็นส่วนที่ใช้บอกว่าเซลล์บรรจุข้อมูลของผู้ใช้ หรือ รายละเอียดของ การจัดการบริหารข้อมูล(Traffic management)
- **CLP** (Cell loss priority) เป็นบิตที่ใช้บอกระดับความสำคัญของเซลล์นั้น เช่น CLP=1 เซลล์จะถูกทิ้งได้ เมื่อมีการคับคั่งของข้อมูลในเครือข่าย
- **HEC** (Header Error Control) ทำหน้าที่ตรวจสอบเซลล์ที่ไม่สอดคล้องตามที่ระบุไว้ในส่วนหัว ถ้าพบว่าส่วนหัวผิดพลาด เซลล์ที่ได้รับจะถูกทิ้งไป

ระบบเครือข่าย ATM เป็นระบบแบบสวิตซ์ กล่าวคือในเครือข่าย ATM นั้น แต่ละเส้นทางการติดต่อ (Connection) สามารถส่งข้อมูลถึงกันได้ทันที ไม่ต้องรอให้อีกคนหนึ่งส่งเสร็จก่อน โดย ATM สวิตซ์จะทำหน้าที่ในการมัลติเพล็กซ์(Multiplex) และจัดการส่งข้อมูลนั้นตามที่กำหนดไว้ในส่วนหัวไปสู่ปลาย

ทาง เมื่อข้อมูลของผู้ใช้เข้ามา จะถูกตัดแบ่งย่อยเป็นกลุ่ม กลุ่มละ 48 ไบต์ และเติมส่วนหัวเข้าไป อีก 5 ไบต์ แล้วจึงส่งไปตามเส้นทางต่างๆในเครือข่าย ATM ซึ่งระบุไว้โดยส่วนหัว เมื่อถึงปลายทางแล้วก็จะเอาส่วนหัวออก แล้วประกอบกันเป็นข้อมูลชิ้นใหญ่เหมือนเดิม ลักษณะของ ATM นี้จะคล้ายกับระบบเครือข่ายแพคเก็ตสวิตซ์ชิ่งอื่นๆ ที่มีอยู่ เช่น x.25 หรือเฟรมรีเลย์ (Frame relay) แต่ต่างกันที่ ATM จะมีขนาดแพคเก็ตเล็กและคงที่ โดยรูปแบบการส่งข้อมูลของ ATM เป็นแบบคอนเน็คชันโอเรียนเต็ท (Connection-oriented) กล่าวคือจะมีการสร้างเส้นทางการติดต่อจากต้นทางถึงปลายทางเป็นเส้นทางที่แน่นอนก่อน แล้วจึงเริ่มส่งข้อมูล เมื่อส่งข้อมูลเสร็จก็ปิดเส้นทางการติดต่อ นอกจากนี้ ATM ยังมีลักษณะอีกอย่างหนึ่งที่มีความสำคัญมาก คือ ATM จะมีคุณภาพของบริการในการส่งข้อมูล ซึ่งสามารถรับประกันคุณภาพของการส่งข้อมูลในแต่ละเส้นทางการติดต่อได้ นั่นคือเมื่อมีการเริ่มสร้างเส้นทางการติดต่อจะมีการตกลงระดับคุณภาพของบริการที่ต้องการใช้ก่อน เพื่อให้เราสามารถส่งข้อมูลโดยได้รับคุณภาพของการส่งข้อมูลตามที่กำหนดไว้นั่นเอง



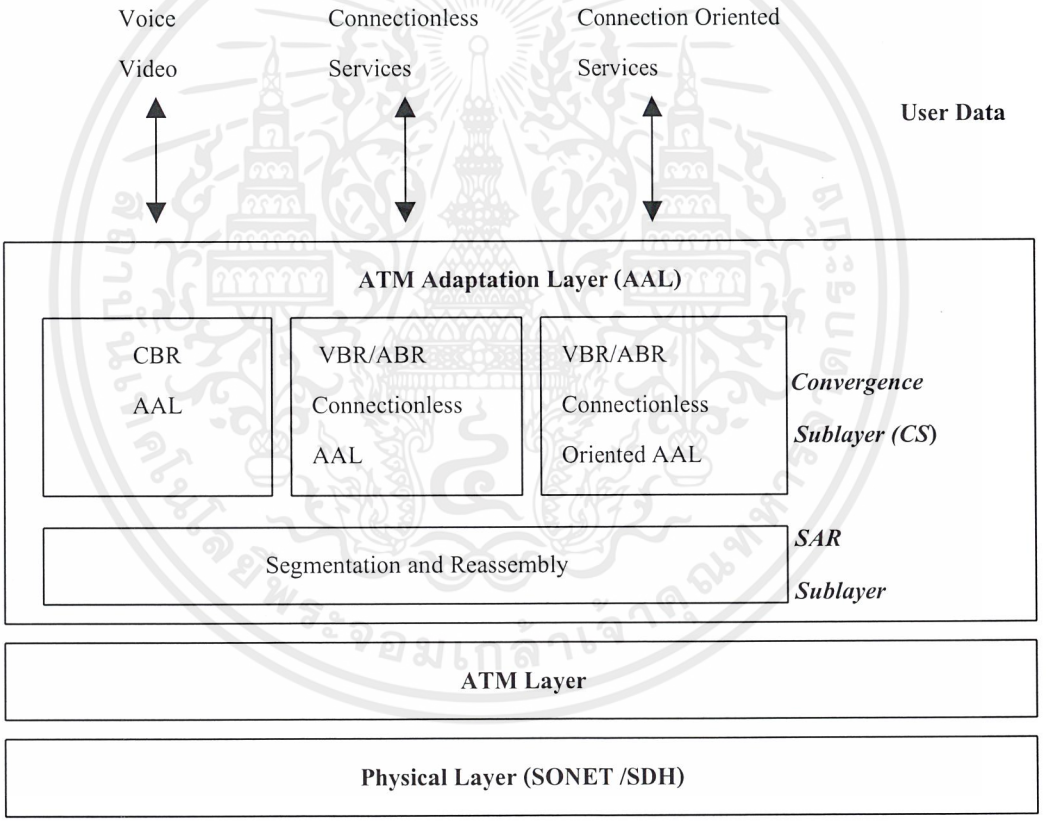
รูปที่2.1 แสดงรูปแบบ ATM เซลล์ ของ UNI (ซ้าย) และ NNI (ขวา)

2.2 โครงสร้างโปรโตคอลของ ATM

โครงสร้างโปรโตคอลของ ATM จะแบ่งการทำงานออกเป็น 3 ชั้น ดังต่อไปนี้

1. **ระดับกายภาพ (Physical Layer)** เป็นข้อกำหนดเกี่ยวกับตัวนำสัญญาณที่ใช้ในการส่งสัญญาณดิจิตอล การนำ ATM มาใช้ในเครือข่ายโทรคมนาคมนั้น จะนำมาใช้ร่วมกับ SONET (Synchronous Optical Network)/SDH (Synchronous Digital Hierarchy) โดยมีเส้นใยแก้วนำแสงเป็นตัวนำสัญญาณ
2. **ATM Layer** ทำหน้าที่สร้างส่วนหัวของเซลล์ และประมวลผลส่วนหัวของเซลล์ที่รับเข้ามา โดยอ่านค่า VCI/VPI ของเซลล์ และหาเส้นทางที่จะส่งเซลล์ออกไป แล้วจึงกำหนด VCI/VPI ใหม่ให้กับส่วนหัวของเซลล์นั้น
3. **ATM Adaptation Layer (AAL)** ทำหน้าที่ปรับบริการที่ได้รับจาก ATM Layer ให้สอดคล้องกับความต้องการของโปรโตคอลและแอพพลิเคชั่น ในระดับชั้นที่สูงกว่า (Higher Layer) โดยแบ่งเป็น 4 ชนิดด้วยกันเพื่อใช้กับแอพพลิเคชั่นที่ต่างกัน ดังต่อไปนี้

- **AAL1** เป็นวิธีการกำหนดให้มีการส่งและรับข้อมูลด้วยอัตราคงที่ (Constant Bit Rate :CBR) โดยการจำลองวงจรการเชื่อมโยงระหว่างตัวรับตัวส่งข้อมูลที่ส่งมีลักษณะเป็นสตรีม (Stream) เพื่อใช้กับแอพพลิเคชั่นที่มีการส่งสัญญาณแบบจุดต่อจุดอย่างต่อเนื่อง
- **AAL2** เป็นวิธีการรับส่งข้อมูลแบบปรับค่าความเร็วของการรับส่งได้ตามที่ต้องการ (Variable Bit Rate :VBR) โดยเน้นการใช้อัตราความเร็วตามที่ต้องการ จึงนำมาใช้กับการรับส่งสัญญาณเสียงและภาพได้
- **AAL3/4** เป็นวิธีการรับส่งข้อมูลแบบปรับค่าความเร็วของการรับส่งได้ตามที่ต้องการเช่นเดียวกับ AAL2 แต่ต่างกันที่สามารถรับส่งข้อมูลแบบอะซิงโครนัส (Asynchronous) ได้ กล่าวคือเวลาในการส่งและรับข้อมูลไม่จำเป็นต้องสัมพันธ์กัน
- **AAL5** มีวิธีการรับส่งข้อมูลเช่นเดียวกับ AAL3/4 สามารถใช้กับการสื่อสารข้อมูลซึ่งมีเชื่อมต่อแบบคอนเน็กชันเลส (Connectionless)ได้ และมีส่วนหัวของข้อมูลสั้นกว่า AAL3/4



รูปที่ 2.2 แสดงโครงสร้างโปรโตคอลของ ATM

โปรโตคอลในชั้น AAL นี้จะควบคุมการติดต่อสื่อสารจากต้นทางถึงปลายทาง และจะถูกประมวลผลโดยผู้ส่งและผู้รับข้อความ (message) เท่านั้น ชั้น AAL แบ่งออกเป็นชั้นย่อย 2 ชั้น คือ ชั้น **Convergence Sublayer (CS)** ช่วยในการเชื่อมต่ออุปกรณ์ (interface) ที่ไม่ใช่ ATM เข้ากับ ATM และชั้น **Segmentation and Reassembly Sublayer (SAR)** ทำหน้าที่ตัดข้อความที่โปรโต

คอลหรือแอพพลิเคชั่นต้องการส่งออกเป็นส่วนย่อยๆ เพื่อนำไปสร้างเซลล์หรือนำส่วนข้อมูลของเซลล์มาต่อกันเป็นข้อความ

ATM จะแบ่งบริการออกเป็น 4 คลาส(Class) ดังแสดงในรูปที่ 2.3

<table>
<tr><th>Attribute</th><th>Class A</th><th>Class B</th><th>Class C</th><th>Class D</th></tr>
<tr><td>Connection mode</td><td colspan="3">Connection-oriented</td><td>Connectionless</td></tr>
<tr><td>Bit rate</td><td>คงที่</td><td colspan="3">ปรับค่าได้</td></tr>
<tr><td>ความสัมพันธ์ของเวลาของต้นทางและปลายทาง</td><td colspan="2">ต้องสัมพันธ์</td><td colspan="2">ไม่ต้องสัมพันธ์</td></tr>
<tr><td>ตัวอย่างการใช้งาน</td><td>Circuit emulation</td><td>Variable bit rate ภาพ หรือ เสียง</td><td>Connection - oriented data</td><td>Connectionless data tranfer</td></tr>
<tr><td rowspan="2">ATM Adaptation Layer (AAL)</td><td rowspan="2">1</td><td rowspan="2">2</td><td>3</td><td>4</td></tr>
<tr><td colspan="2">5</td></tr>
</table>

ตารางที่ 2.1 แสดงบริการ 4 คลาสของ ATM

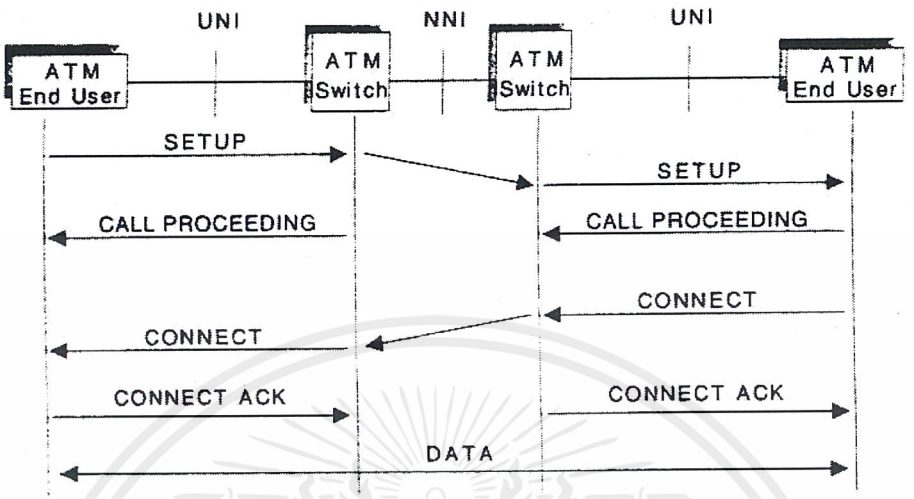
2.3 การเชื่อมต่อของ ATM

การเชื่อมต่อของ ATM จะเป็นแบบจุดต่อจุด (Point-to-point) โดยแต่ละการเชื่อมต่อที่สร้างขึ้นระหว่างผู้ส่งและผู้รับ จะเรียกว่าวงจรเสมือน โดยจะแบ่งออกเป็น 2 ชนิด ดังต่อไปนี้

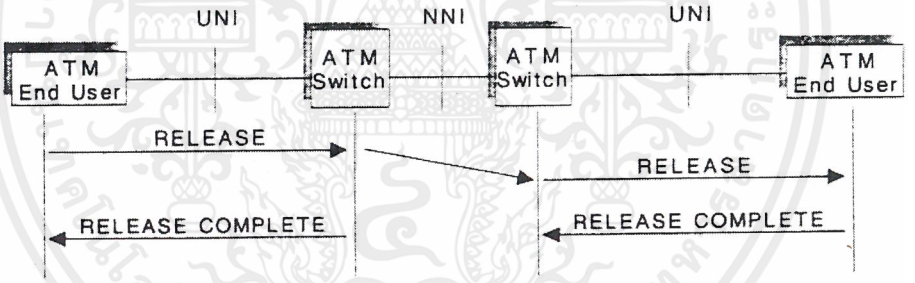
1. **Switched Virtual Circuit (SVC)** เป็นวงจรเสมือนที่จะเกิดขึ้นเมื่อต้นทางต้องการติดต่อกับปลายทาง จำเป็นต้องมี Signaling Protocol เพื่อทำการสร้างตลอดจนลบเส้นทางการเชื่อมต่อ โดยการสร้างเส้นทางเชื่อมต่อไปยังโฮสต์(Host) ที่แยกกันด้วยฮอบ(Hop) ในสวิตช์ ATM คนละตัวกัน จำเป็นต้องมีการส่งข้อความแจ้งโดยโฮสต์จะเริ่มต้นจากการส่งข้อความที่เรียกว่า SETUP ไปยังสวิตช์ตัวแรก ข้อความ SETUP จะส่งพารามิเตอร์เกี่ยวกับ AAL และคุณภาพของบริการในการส่งข้อมูลระหว่างการจัดตั้งเส้นทางเชื่อมต่อนั้น เมื่อสวิตช์ตอบรับแล้วจะส่งข้อมูลเกี่ยวกับ VCI/VPI กลับมายังฮอบที่อยู่ก่อนหน้านั้น โดยใช้ข้อความที่เรียกว่า CALL PROCEEDING กระบวนการเช่นนี้จะดำเนิน ต่อไปเรื่อยๆ จนกระทั่งไปถึงปลายทาง ทางปลายทางจะส่งข้อความที่เรียกว่า CONNECT กลับมายังต้นทาง จากนั้นต้นทางก็จะส่งข้อความ CONNECT ACK เพื่อรับการเชื่อมต่อไปยังปลายทาง รูปที่2.4 แสดงการสร้างการเชื่อมต่อของ ATM สำหรับการลบเส้นทางเชื่อมต่อนั้น ก็ทำได้ในทำนองเดียวกันโดยใช้ข้อ

ความที่เรียกว่า RELEASE และ RELEASE COMPLETE รูปที่ 2.5 แสดงการลบการเชื่อมต่อของ

ATM



รูปที่ 2.3 แสดงการสร้างการเชื่อมต่อของ ATM



รูปที่ 2.4 แสดงการลบการเชื่อมต่อของ ATM

2. **Permanent Virtual Circuit (PVC)** เป็นการสร้างเส้นทางเชื่อมต่อระหว่างแต่ละโฮสต์ตั้งแต่ตอนเพิ่มโฮสต์เข้ามา วิธีนี้จะช่วยลดขนาดของโอเวอร์เฮดในการจัดตั้งเส้นทางการเชื่อมต่อ แต่อย่างไรก็ตาม วิธีการ PVC นี้ไม่เหมาะที่จะใช้กับเครือข่ายที่มีโฮสต์จำนวนมาก

2.4 การจัดการเกี่ยวกับแอดเดรสของ ATM

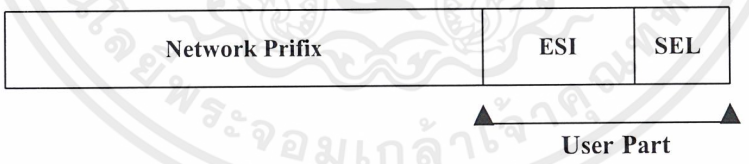
ก่อนที่จะมีการสร้างการติดต่อระหว่างผู้ใช้ ATM 2 คน จะต้องมีการแลกเปลี่ยนข้อมูลเกี่ยวกับพารามิเตอร์ (Parameter) ต่างๆเช่น ค่าแบนด์วิธ (Bandwidth),คุณภาพของบริการในการส่งข้อมูลก่อน ดังนั้นฝ่ายส่งจะต้องทราบแอดเดรสของฝ่ายรับก่อน

2.4.1 ลักษณะของ ATM แอดเดรส

1. จะไม่เกี่ยวข้องกับแอดเดรสของโปรโตคอลในระดับอื่น เช่น ไม่เกี่ยวข้องหรือมีความสัมพันธ์กับ IP แอดเดรส
2. มีความแตกต่างกันระหว่าง ATM แอดเดรสที่ใช้ ใน ATM Public network กับ ATM private network
3. มีโครงสร้างแบบลำดับขั้น (Hierarchical structure)
4. มีขนาด 20 ไบต์และจะต้องไม่ซ้ำกับแอดเดรสของ ATM โฮสต์อื่นๆ

ATM แอดเดรส จะแบ่งเป็น 2 ส่วนคือ

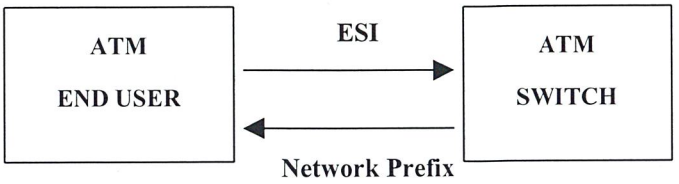
1. Network prefix มีขนาด 13 ไบต์
2. User part มีขนาด 7 ไบต์จะประกอบด้วย 2 ส่วนย่อยคือ ESI กับ SEL
   - ESL จะมีขนาดเท่ากับ 48 บิตซึ่งทำหน้าที่คล้ายกับ MAC แอดเดรสในระบบเครือข่ายอีเทอร์เน็ต
   - SEL จะมีขนาด 1ไบต์ใช้ในกำหนดแอดเดรสตำแหน่งไบต์สุดท้ายของ ATM แอดเดรสของโฮสต์



รูปที่ 2.5 แสดง ATM แอดเดรส

2.4.2 Address Registration

ในการที่จะจัดการกับ ATM แอดเดรสซึ่งมีขนาดความยาว 20 ไบต์นั้นเป็นสิ่งที่ทำได้ยากลำบากและอาจเกิดข้อผิดพลาดขึ้นได้ง่าย ดังนั้น ATM Forum จึงได้กำหนดวิธีการลงทะเบียน ATM แอดเดรสแบบอัตโนมัติโดยใช้โปรโตคอล Interim Local Management Interface (ILMI) ในการแลกเปลี่ยนข้อมูลกันระหว่างผู้ใช้ ATM ปลายทาง และATM สวิตช์ ทำให้ผู้ใช้ ATM ปลายทางสามารถส่ง ESI แอดเดรสของตนไปให้ ATM สวิตช์ และในขณะเดียวกันก็จะได้รับ Network Prefix จาก ATM สวิตช์กลับมาประกอบกับ ESI ของตนเพื่อประกอบกันเป็น ATM แอดเดรสของโฮสต์นั้นๆ



รูป 2.6 แสดงการแลกเปลี่ยนข้อมูลในกระบวนการ Address Registration

บทที่ 3
การนำแอพพลิเคชั่นบนระบบเครือข่าย LAN มาใช้งาน บนเครือข่าย ATM

วิธีการในการทำให้แอพพลิเคชั่นที่ทำงานบนระบบเครือข่าย LAN สามารถนำมาใช้งานบนระบบเครือข่าย ATM ได้จะมีอยู่ 2 วิธี คือ

1 การทำ LAN Emulation

2 การทำ Classical IP over ATM

3.1 LAN Emulation

LAN Emulation เป็นการทำให้แอพพลิเคชั่นที่ทำงานบนระบบเครือข่าย LAN ในปัจจุบันสามารถที่จะทำงานบนระบบเครือข่าย ATM ได้ ซึ่งจะต้องมีการจำลองการลักษณะการทำงานหลายอย่างของระบบเครือข่าย LAN เช่น จำลองการทำงานของระบบเครือข่าย LAN ซึ่งทำงานแบบคอนเน็กชันเลส โดยสถานีบนระบบเครือข่าย LAN สามารถส่งข้อมูลได้โดยไม่ต้องมีการสร้างการติดต่อระหว่างต้นทางและปลายทางก่อน,และต้องสนับสนุนการทำบรอดแคสท์(Broadcast) และมัลติแคสท์(Multicast) ,การติดต่อระหว่างระบบเครือข่าย LAN แบบเดิม (Legacy LAN) และ LAN Emulation จะต้องสามารถทำได้ และที่สำคัญที่สุดคือแอพพลิเคชั่นต่างๆที่ทำงานบนระบบเครือข่าย LAN จะต้องสามารถทำงานได้บนระบบเครือข่าย ATM โดยไม่ต้องมีการเปลี่ยนแปลงใดๆทั้งสิ้น

3.1.1 โครงสร้างของ LAN Emulation

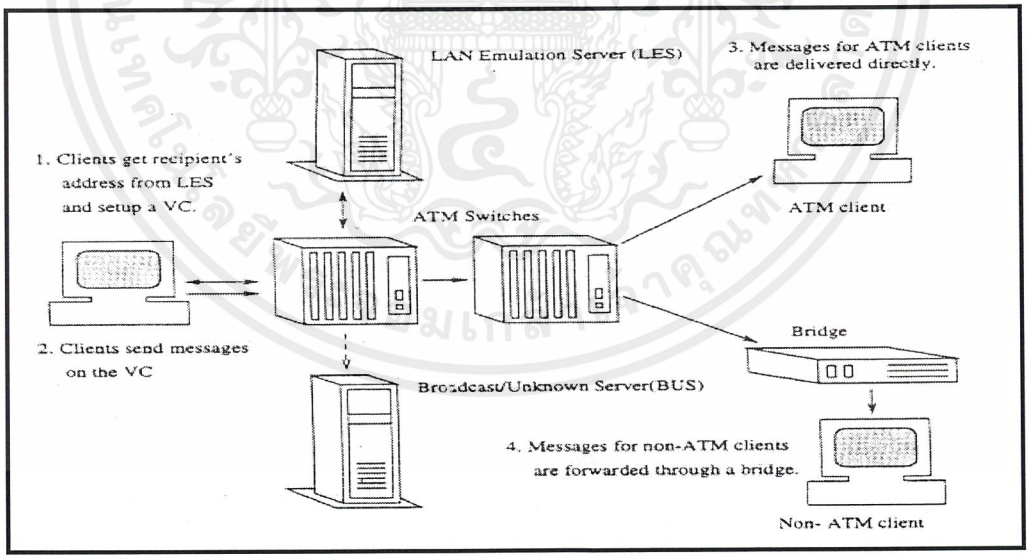
การที่เราจะมองโครงสร้างของ LAN Emulation นั้นเราอาจมองได้ 2 ลักษณะคือ

- **มองในรูปของระดับชั้น(Layer)** ถ้าเรามองในลักษณะของระดับชั้นนั้น LAN Emulation จะเป็นเหมือนอีกระดับชั้นหนึ่งที่เพิ่มขึ้นมาโดยอยู่ระหว่างระดับชั้นสูงกว่าซึ่งเป็นพวก อีเทอร์เน็ตหรือโทเคนริง(Token ring) กับระดับชั้นที่ต่ำกว่า(Lower layer) ซึ่งเป็นระดับชั้นของ ATM โดย LAN Emulation จะทำการติดต่อกับระดับชั้นสูงกว่า ทำให้ระดับชั้นสูงกว่า ดูเหมือนว่ามันทำงานอยู่บนระบบเครือข่าย LAN ตามปกติ
- **มองในรูปของโปรโตคอล** LAN Emulation ไคลเอนต์กับเซิร์ฟเวอร์ จะติดต่อกันโดยใช้ LAN Emulation user-to-network interface (LUNI)

3.1.2 ส่วนประกอบของ LAN Emulation

จะประกอบด้วยส่วนต่างๆต่อไปนี้คือ

1. **LAN Emulation Client (LEC)** อาจจะเป็นสถานีปลายทาง,บริดจ์หรือเราท์เตอร์ก็ได้ LEC สามารถทำแอดเดรสรีโซลูชัน (Address Resolution) ,ส่งข้อมูลและฟังก์ชันควบคุมต่างๆ โดย LEC จะติดต่อกับส่วนอื่นๆบน LAN Emulation โดยใช้ LAN Emulation user-to-network-interface (LUNI)
2. **LAN Emulation Server (LES)** ทำหน้าที่คือเป็นตัวรับลงทะเบียน (registry) ค่า MAC แอดเดรสจาก LEC และเป็นแอดเดรสรีโซลูชันเซิร์ฟเวอร์ (Address Resolution Server) ให้กับ LEC ต่างๆภายใน LAN Emulation
3. **LAN Emulation Configuration Server (LECS)** จะทำหน้าที่ส่งข้อมูล,ข้อกำหนดต่างๆ รวมทั้งค่าพารามิเตอร์ที่ใช้ใน LAN Emulation ให้กับ LEC ที่จะเข้าเป็นสมาชิกของ LAN Emulation โดย LECS จะส่งค่า ATM แอดเดรสของ LES ให้กับ LEC
4. **Broadcast and Unknown Server (BUS)** จะทำหน้าที่ในการจัดการกับบรอดแคสท์แอดเดรส (Broadcast address) ,มัลติแคสท์แอดเดรส (Multicast address) โดยทุก LEC ใน LAN Emulation จะรักษาเส้นทางการติดต่อกับ BUS โดยมีลักษณะเป็น point to multipoint โดยมี BUS เป็นโหนดราก(Root Node) ซึ่งทำให้ LEC สามารถส่งข้อมูลแบบไม่ต้องมีการสร้างการติดต่อก่อนได้(Connectionless)



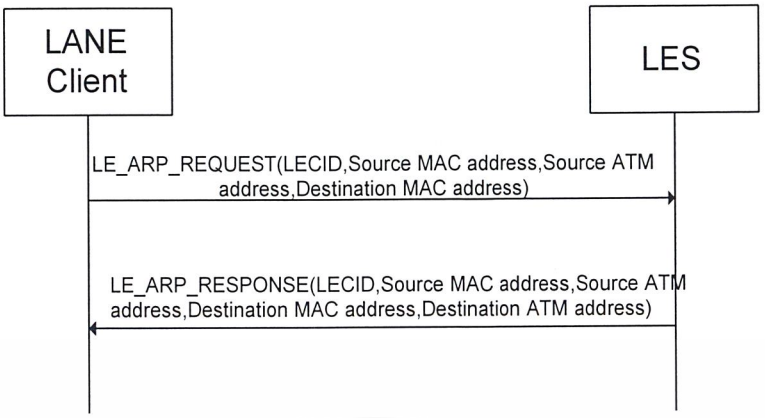
รูปที่ 3.1 แสดงส่วนประกอบของ LAN Emulation

3.1.3 การทำงานของ LAN Emulation

การทำงานของ LAN Emulation พื้นฐานจะประกอบด้วยขั้นตอนต่อไปนี้

1. **INITIALIZATION** เป็นขั้นตอนที่ LEC ได้รับ ATM แอดเดรสของ LES และสร้างการติดต่อกับ LES และ BUS จะแบ่งเป็น 4 ขั้นตอนย่อยคือ
   1.1 **LECS connect phase** ในขั้นตอนนี้เมื่อ LEC จะเข้ามาเป็นสมาชิกใน LAN Emulation นั้น LEC จะต้องสร้างเส้นทางการติดต่อกับ LECS
   1.2 **Configuration phase** ในขั้นตอนนี้ LEC จะได้รับ ATM แอดเดรสของ LES และค่าพารามิเตอร์รวมทั้งข้อกำหนดต่างๆ จาก LECS
   1.3 **Join phase** ในขั้นตอนนี้ LEC จะสร้างการติดต่อกับ LES และพยายามเข้าร่วมเป็นสมาชิกของ LAN Emulation โดยในขั้นตอนนี้ LEC สามารถที่จะลงทะเบียนค่า MAC แอดเดรส ของตนกับ ATM แอดเดรสที่ใช้ได้
   1.4 **BUS connect phase** ในขั้นตอนนี้ LEC จะสร้างการติดต่อกับ BUS โดย LEC จะต้องส่ง REQUEST ไปยัง LES โดยระบุ MAC แอดเดรสเป็นบรอดแคสท์แอดเดรส จากนั้น LES จะส่งค่า ATM แอดเดรสของ BUS กลับไปให้ LEC โดย LEC จะใช้ค่าที่ได้รับมานี้มาสร้างเส้นทางการติดต่อเพื่อส่งข้อมูลไปยัง BUS ได้จากนั้น BUS เพิ่ม LEC นั้นลงใน point-to-multipoint VCC เพื่อใช้ในการส่งข้อมูลแบบมัลติแคสท์ไปยังทุก LEC ที่มี แอดเดรสนั้น
2. **REGISTRATION** ขั้นตอนการลงทะเบียนแอดเดรสนั้น LAN Emulation อนุญาตให้ LEC ลงทะเบียนคู่ของ MAC แอดเดรสกับ ATM แอดเดรสเพิ่มเติมได้หลังจากไม่ได้ลงทะเบียนในขั้นตอนของ Join phase
3. **ADDRESS RESOLUTION** เป็นขั้นตอนที่ LEC ต้นทางทำการหา ATM แอดเดรสของ LEC ปลายทาง เพื่อสร้างเส้นทางการติดต่อไปยังปลายทาง

   เมื่อ LEC ต้นทางต้องการที่จะส่งข้อมูลไปยัง LEC ปลายทางนั้น LEC ต้นทางจะต้องทำการหา ATM แอดเดรสของ LEC ปลายทางก่อนเพราะ LEC ต้นทางจะรู้แต่ MAC แอดเดรสของ LEC ปลายทาง โดยก่อนอื่น LEC ต้นทางจะตรวจสอบตารางที่จับคู่ของ MAC แอดเดรสกับ ATM แอดเดรสภายในหน่วยความจำแคช(Cache) ของมันก่อนซึ่งถ้ามีก็สามารถสร้างเส้นทางการติดต่อได้ทันที แต่ถ้าไม่มี LEC ต้นทางจะต้องทำการส่ง REQUEST ไปยัง LES เพื่อขอ ATM แอดเดรสของ LEC ปลายทาง โดย LES จะตรวจสอบค่า MAC แอดเดรสของ LEC ปลายทางซึ่งถ้า LES รู้จัก MAC แอดเดรสของ LEC ปลายทางนั้น LES ก็จะส่ง ATM แอดเดรสของ LEC ปลายทางนั้นกลับไปให้ LEC ต้นทาง ถ้าไม่รู้จัก LES ก็จะส่ง REQUEST นั้นต่อไปยัง LEC ตัวอื่น ซึ่งก็คือกรณีที่มีการส่งข้อมูลข้ามเครือข่าย LAN นั่นเอง



รูปที่ 3.2 แสดงการทำแอดเดรสรีโซลูชันในกรณีที่ LES รู้จัก MAC แอดเดรสของ LEC ปลายทาง

4. **DATA TRANSFER** เป็นการส่งข้อมูลระหว่างต้นทางกับปลายทาง โดยข้อมูลที่ส่งระหว่าง LEC ต้นทางและปลายทางนั้นอาจส่งด้วยการสร้างเส้นทางการติดต่อระหว่างต้นทางกับปลายทางโดยตรงหรือส่งผ่าน BUS โดยส่งข้อมูลให้กับ BUS แล้ว BUS ก็จะส่งข้อมูลนั้นไปให้ปลายทางซึ่งเป็นการส่งข้อมูลแบบไม่ต้องมีการสร้างการติดต่อก่อนก็ได้

3.2 IP Over ATM

จากการทำงานร่วมกันระหว่าง Internet Engineering Task Force (IETF) และผู้ร่วมงานอีกหลายฝ่าย ทำให้ได้ข้อกำหนดมาตราฐาน RFC เพื่อให้อินเตอร์เน็ตโปรโตคอล(Intermet Protocol (IP)) สามารถนำมาใช้ในเครือข่าย ATM ได้ เนื่องจากโครงสร้างการทำงานของ ATM เป็นแบบคอนเน็กชันโอเรียนเต็ท คือ มีการสร้างเส้นทางการส่งข้อมูลที่แน่นอนก่อนที่จะทำการส่งข้อมูล และเป็นแบบจุดต่อจุด (point to point) ทำให้การทำงานแบบบรอดแคสท์ ซึ่งเป็นการทำงานของระบบเครือข่าย LAN ไม่สามารถนำมาใช้ได้ในเครือข่าย ATM ได้ ดังนั้นจึงได้มีการคิดค้นวิธีการทำ IP Over ATM โดยจะเรียกว่า " Classical IP and ARP Over ATM " ขึ้นมาเพื่อช่วยแก้ปัญหานี้

3.2.1 RFC 1577 "Classical IP and ARP Over ATM"

RFC 1577 เป็นข้อกำหนดมาตราฐาน เพื่อใช้ในการทำ Classical IP and ARP Over ATM โดยมีข้อกำหนดดังต่อไปนี้

- สำหรับทุกวงจรเสมือนในซับเน็ต(Subnet) จะมี Maximium Transmit Unit (MTU) 9180 ไบต์ กับ LLC/SNAP Header 8 ไบต์ เป็นค่ามาตราฐาน ดังนั้นขนาดของเฟรมในชั้น AAL5 จะเท่ากับ 9188 ไบต์
- การใส่ LLC/SNAP กับแพคเก็ตของ IP ใน AAL5 เซลล์ อยู่ในข้อกำหนด RFC1483

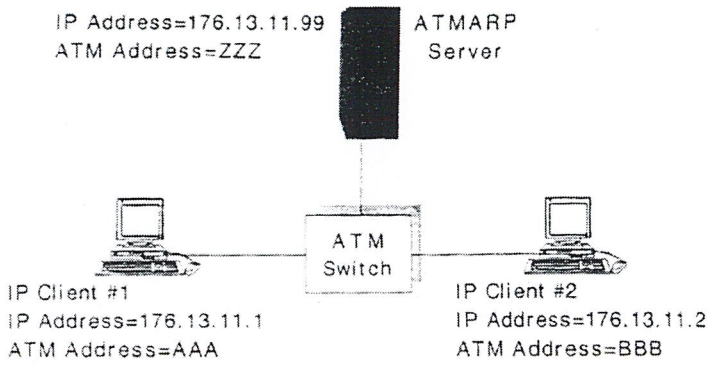
- IP แอดเดรสจะถูกเปลี่ยนเป็น ATM แอดเดรสโดยบริการของ ATMARP ภายใน Logical IP Subnet (LIS) เดียวกัน
- ทุกสมาชิกใน LIS จะต้องใช้หมายเลขเน็ตเวิร์คเดียวกัน(IP network), Subnetwork และ Subnet mask เดียวกัน
- ทุกสมาชิกใน LIS จะต้องเชื่อมต่อโดยตรงกับ ATM เน็ตเวิร์คเดียวกัน
- IP โฮสต์ภายนอก LIS จะเข้าถึง LIS โดยผ่านเราท์เตอร์
- ถ้าการเชื่อมต่อเป็นแบบ SVC สมาชิกใน LIS จะต้องใช้ ATMARP และ InATMARP ในการติดต่อกับ ATMARP เซิร์ฟเวอร์เพื่อเปลี่ยนค่า IP แอดเดรสและATM แอดเดรส
- ถ้าการเชื่อมต่อเป็นแบบ PVC สมาชิกใน LIS จะต้องใช้ InATMARP เพื่อเปลี่ยนค่า VC เป็น IP แอดเดรสโดยไม่จำเป็นต้องใช้ ATMARP เซิร์ฟเวอร์
- สมาชิกใน LIS ต้องสามารถติดต่อกับ สมาชิกอื่นๆ ใน LIS โดยใช้ SVC หรือ PVC

สำหรับค่าต่อไปนี้จะถูกกำหนดสำหรับแต่ละสมาชิกใน LIS

- **ATM hardware address** เป็นค่า ATM แอดเดรสสำหรับแต่ละ IP โฮสต์
- **ATMARP request address** เป็นค่าแอดเดรสของ ATMARP เซิร์ฟเวอร์สำหรับ LIS แต่ถ้าใน LIS มีการเชื่อมต่อแบบ PVC เท่านั้น ส่วนนี้ก็ไม่จำเป็น

3.2.1.1 ATMARP

การทำงานของ ATMARP จะทำงานบนพื้นฐานของ ARP แต่จะมีการเพิ่มเติมบางส่วนเข้าไปเพื่อให้สามารถทำงานได้บนเครือข่าย ATM ซึ่งเป็นการทำงานแบบจุดต่อจุดได้ โดยองค์ประกอบสำคัญสำหรับการทำ Classical IP ก็คือ ATMARP เซิร์ฟเวอร์ซึ่งจะเป็นสมาชิกใน LIS ที่ถูกเลือกมาเพื่อใช้เก็บตารางสำหรับการจับคู่เปรียบเทียบระหว่าง IP แอดเดรสและ ATM แอดเดรสโดยแต่ละ LIS นั้นอย่างน้อยที่สุดจะต้องมีหนึ่ง ATMARP เซิร์ฟเวอร์ที่ได้กำหนด IP แอดเดรสและATM แอดเดรสไว้โดยเฉพาะ และ ATMARP เซิร์ฟเวอร์สามารถที่จะให้บริการมากกว่าหนึ่ง LIS ได้ แต่จะต้องมี IP แอดเดรสและ ATM แอดเดรสอยู่ใน LIS นั้น ATMARP เซิร์ฟเวอร์จะใช้การส่งข้อความ ATMARP และ InATMARP แลกเปลี่ยนระหว่าง ATMARP เซิร์ฟเวอร์กับสมาชิกใน LIS เพื่อสอบถาม IP แอดเดรสและ ATM แอดเดรส โดย ATMARP เซิร์ฟเวอร์นั้นสามารถจะเป็น IP โฮสต์หรือเราท์เตอร์ก็ได้ รูปที่ 3.3 แสดง LIS ที่มี IP ไคลเอนต์ 2 ตัว กับ ATMARP เซิร์ฟเวอร์ 1 ตัว



รูปที่ 3.3 แสดง LIS ที่มี IP ไคลเอนต์ 2 ตัว กับ ATMARP เซิร์ฟเวอร์ 1 ตัว

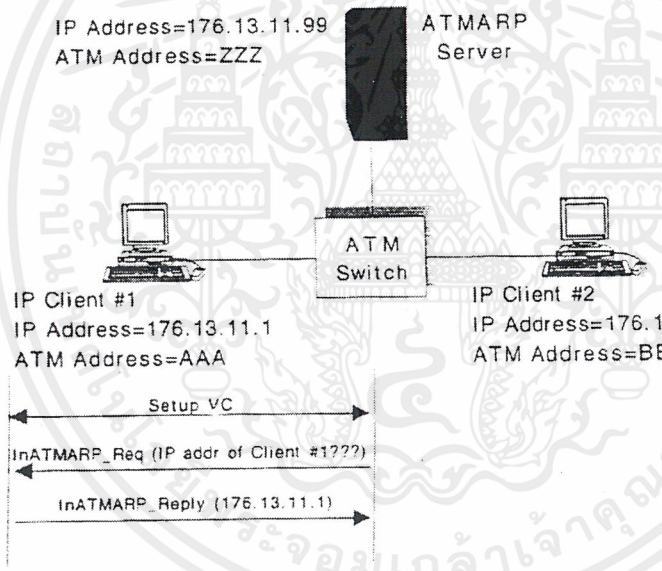
ATMARP โปรโตคอลประกอบด้วยข้อความ 5 ชนิด ดังตารางที่ 3.1

ATMARP message	**คำอธิบาย**
ATMARP request	ส่งจาก IP ไคลเอนต์ไปยังเซิร์ฟเวอร์เพื่อร้องขอ ATM แอดเดรสโดยจะบรรจุ IP แอดเดรส,ATM แอดเดรสของไคลเอนต์ที่ร้องขอ และ IP แอดเดรสของปลายทาง
ATMARP reply	ตอบกลับจากเซิร์ฟเวอร์ไปยัง IP ไคลเอนต์กับ ATM แอดเดรสปลายทาง โดยจะบรรจุ IP แอดเดรสและ ATM แอดเดรสของ client และ ปลายทาง
InATMARP request	ส่งจากเซิร์ฟเวอร์ไปยัง IP ไคลเอนต์ผ่าน VC เพื่อร้องขอ IP แอดเดรสโดยจะบรรจุ ATM แอดเดรสของไคลเอนต์,IP แอดเดรสและ ATM แอดเดรสของ ATMARP server
InATMARP reply	ตอบกลับจาก IP ไคลเอนต์ไปยังเซิร์ฟเวอร์ผ่าน VC โดยจะบรรจุ IP แอดเดรส และ ATM แอดเดรส ของไคลเอนต์และเซิร์ฟเวอร์
ATMARP NAK	ปฏิเสธการตอบกลับ ATMARP request ที่ส่งจากเซิร์ฟเวอร์ไปยัง IP ไคลเอนต์

ตารางที่ 3.1 ATMARP message

3.2.1.2 Registration

IP ไคลเอนต์จะต้องทำการลงทะเบียน IP แอดเดรสและ ATM แอดเดรสกับ ATMARP เซิร์ฟเวอร์ โดยจะกระทำเมื่อ IP ไคลเอนต์ได้เริ่มสร้าง SVC กับ ATMARP เซิร์ฟเวอร์ การที่ IP ไคลเอนต์สามารถทำเช่นนี้ได้เพราะได้ถูกติดตั้ง ATM แอดเดรสของ ATMARP เซิร์ฟเวอร์ไว้แล้ว ขั้นตอนต่อไป ATMARP เซิร์ฟเวอร์จะส่ง InATMARP request ไปยังไคลเอนต์เพื่อร้องขอ IP แอดเดรสของไคลเอนต์ จากนั้นไคลเอนต์ก็จะส่ง InATMARP reply ตอบกลับ โดยจะบรรจุ IP แอดเดรสและ ATM แอดเดรสของไคลเอนต์ เมื่อ ATMARP เซิร์ฟเวอร์ได้รับ IP แอดเดรสและ ATM แอดเดรสของไคลเอนต์ ก็จะตรวจสอบค่าที่ได้กับตารางที่ใช้เก็บ IP แอดเดรสและ ATM แอดเดรสว่ามีอยู่แล้วหรือไม่ ถ้าไม่มีก็จะบันทึกเวลาและเก็บค่าที่ได้ลงในตาราง สำหรับค่านี้จะสามารถใช้งานได้อย่างน้อยที่สุด 20 นาที IP ไคลเอนต์เองก็สามารถที่จะส่ง InATMARP request ไปยังเซิร์ฟเวอร์เพื่อร้องขอ IP แอดเดรสของเซิร์ฟเวอร์ ตัวอย่างขั้นตอนการลงทะเบียนโดย IP client #1 แสดงดังรูปที่ 3.4 สำหรับIP client #2 ก็จำเป็นต้องทำการลงทะเบียนกับ ATMARP เซิร์ฟเวอร์ด้วยในตอนเริ่มต้น



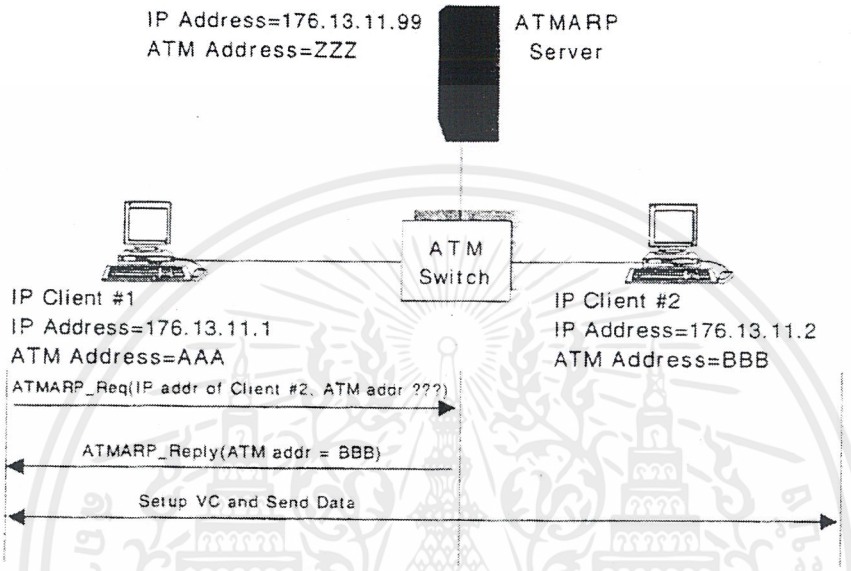
รูปที่ 3.4 แสดงขั้นตอนการลงทะเบียนแอดเดรส ของ IP client #1 (RFC1577)

3.2.1.3 Address resolution

ถ้า IP client #1 ต้องการติดต่อกับ IP client #2 และการเชื่อมต่อมีอยู่แล้วแพคเก็ตข้อมูลก็สามารถส่งถึงกันโดยผ่านการเชื่อมต่อนั้น โดยIP client #1 อาจจะมี ATM แอดเดรสของ IP client #2 ใน ARP cache ของมันเอง ดังนั้นมันก็สามารถสร้าง SVC กับ IP client #2 ได้ทันที อย่างไรก็ตามถ้าการเชื่อมต่อยังไม่เกิดขึ้น และ IP client #1 ยังไม่รู้ ATM แอดเดรส ของ IP client #2 ขบวนการ ATMARP ก็จะถูกกระทำ

เริ่มต้น IP client #1 จะส่ง ATMARP request ไปยัง ATMARP เซิร์ฟเวอร์โดยจะบรรจุ IP แอดเดรส ,ATM แอดเดรสของ client #1 และ IP แอดเดรสของ client #2 ถ้า ATMARP เซิร์ฟเวอร์มี IP

แอดเดรส,ATM แอดเดรสของ client #2 อยู่ มันก็จะส่งค่า IP แอดเดรส,ATM แอดเดรสของ client #2 ไปใน ATMARP reply เมื่อ client 1# ได้รับ ATMARP reply มันก็จะรู้ ATM แอดเดรสของ client #2 ทำให้สามารถสร้าง SVC เชื่อมต่อกับ client #2 ได้ แต่ถ้า ATMARP เซิร์ฟเวอร์ไม่มี IP แอดเดรส,ATM แอดเดรสของ client #2 มันจะตอบกลับด้วย ATMARP NAK ตัวอย่างขบวนการ ATMARP address resolution แสดงดังรูปที่ 3.5



รูปที่ 3.5 แสดงขบวนการ ATMARP address resolution (RFC1577)

3.3 ข้อเปรียบเทียบระหว่าง LAN Emulation และ IP over ATM

การทำ LAN Emulation ทำให้สามารถนำแอพพลิเคชั่นต่างๆที่ทำงานบนระบบเครือข่าย LAN มาใช้งานบนเครือข่าย ATM ได้ทันทีไม่ว่าแอพพลิเคชั่นนั้นจะใช้โปรโตคอลในระดับชั้น Network layer อะไรก็ตามเช่น IP หรือ IPX เป็นต้น ส่วนการทำ IP over ATM บนเครือข่าย ATM นั้นจะสามารถใช้งานแอพพลิเคชั่น ที่ทำงานกับโปรโตคอล IP เท่านั้น

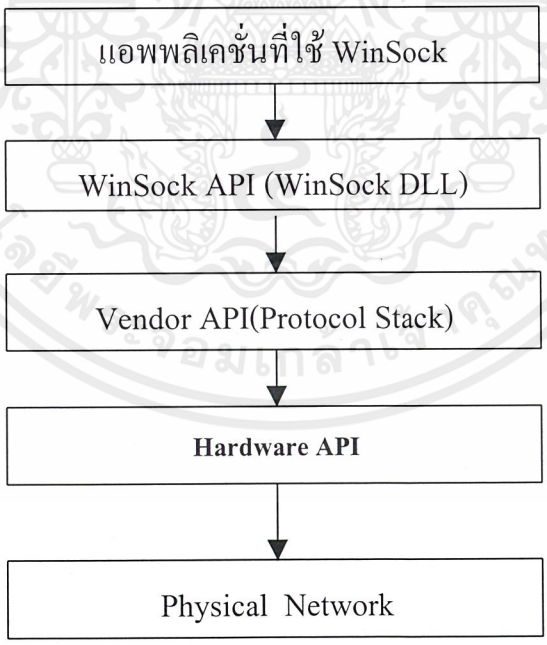
การทำ IP over ATM จะสามารถใช้ประสิทธิภาพของระบบเครือข่าย ATM ได้ดีกว่าการทำ LAN Emulation เพราะจะมีโอเวอร์เฮดในส่วนของการทำงานน้อยกว่า และระดับชั้นที่ใช้ในการเปลี่ยนข้อมูลให้อยู่ในรูปของ ATM เซลล์นั้น น้อยกว่า การทำ LAN Emulation

บทที่ 4
แนะนำ WinSock

4.1 WinSock คืออะไร

WinSock เป็นแอพพลิเคชั่นโปรแกรมอินเตอร์เฟส (Application Program Interface (API)) ที่ทำงานบนระบบปฏิบัติการไมโครซอฟท์วินโดวส์ (Microsoft Windows Operating System) ซึ่งช่วยให้โปรแกรมเมอร์ (Programmer) สามารถเขียนโปรแกรมติดต่อกับเน็ตเวิร์คโปรโตคอล (Network Protocol) ในระดับล่าง (Low Level) ได้ง่ายขึ้น โดยมี กลุ่มบริษัทผู้ขาย (Vendor) เป็นจำนวนมากได้ให้การยอมรับให้ WinSock เป็นแอพพลิเคชั่นโปรแกรมอินเตอร์เฟสมาตราฐาน เพื่อให้การเขียนโปรแกรมติดต่อกับเน็ตเวิร์คเป็นมาตราฐานเปิด (Open Standard) ซึ่งจะทำให้การเขียนและการพัฒนาโปรแกรมเน็ตเวิร์คเป็นไปได้อย่างรวดเร็วและง่ายขึ้น

โดย WinSock จะเป็นชุดของโครงสร้างข้อมูล (Data Structures) ที่ถูกกำหนดไว้และเป็นฟังก์ชันคอล (Function Call) ที่ทำเป็นไดนามิกลิงค์ไลบรารี (Dynamic Link Library (DLL)) ไว้ให้ผู้ใช้เรียกใช้ฟังก์ชันเหล่านี้ในการติดต่อกับส่วนเน็ตเวิร์ค โดยตัว WinSock จะทำการแปลฟังก์ชันที่ผู้ใช้เรียกใช้ไปเป็นคำสั่งในการติดต่อกับส่วนเน็ตเวิร์คให้เอง



รูปที่ 4.1 แสดงการใช้ WinSock ช่วยในการเขียนโปรแกรมติดต่อกับส่วนเน็ตเวิร์ค

4.2 WinSock 2.0

WinSock ได้ถูกพัฒนามาหลายเวอร์ชั่น (Version) ตั้งแต่ WinSock 1.0 มาเป็น WinSock 1.1 จนกระทั่งมาเป็น WinSock 2.0 ในปัจจุบัน โดย WinSock 2.0 ได้มีการเพิ่มคุณลักษณะเด่น (Feature) และความสามารถหลายประการจาก WinSock 1.1 ดังนี้

- **สนับสนุนหลายโปรโตคอล (Multiple Protocol Support)** WinSock 2.0 ทำให้สามารถใช้โปรโตคอลได้หลายโปรโตคอล ซึ่งทำให้ WinSock ไม่ถูกจำกัดการทำงานอยู่กับโปรโตคอล TCP/IP เพียงอย่างเดียวดังเช่น WinSock1.1 ที่ผ่านมา

- **โอเวอร์แลปอินพุทเอาท์พุท (Overlapped I/O)** WinSock2.0 จะใช้ Overlapped input/output Model ที่สร้างใน Win32 ซึ่งอนุญาตให้แอพพลิเคชั่นสามารถประมวลผลงานอื่นในขณะที่กำลังรอให้อินพุทเอาท์พุททำงานเสร็จ ทำให้ประสิทธิภาพการทำงานของระบบดีขึ้น

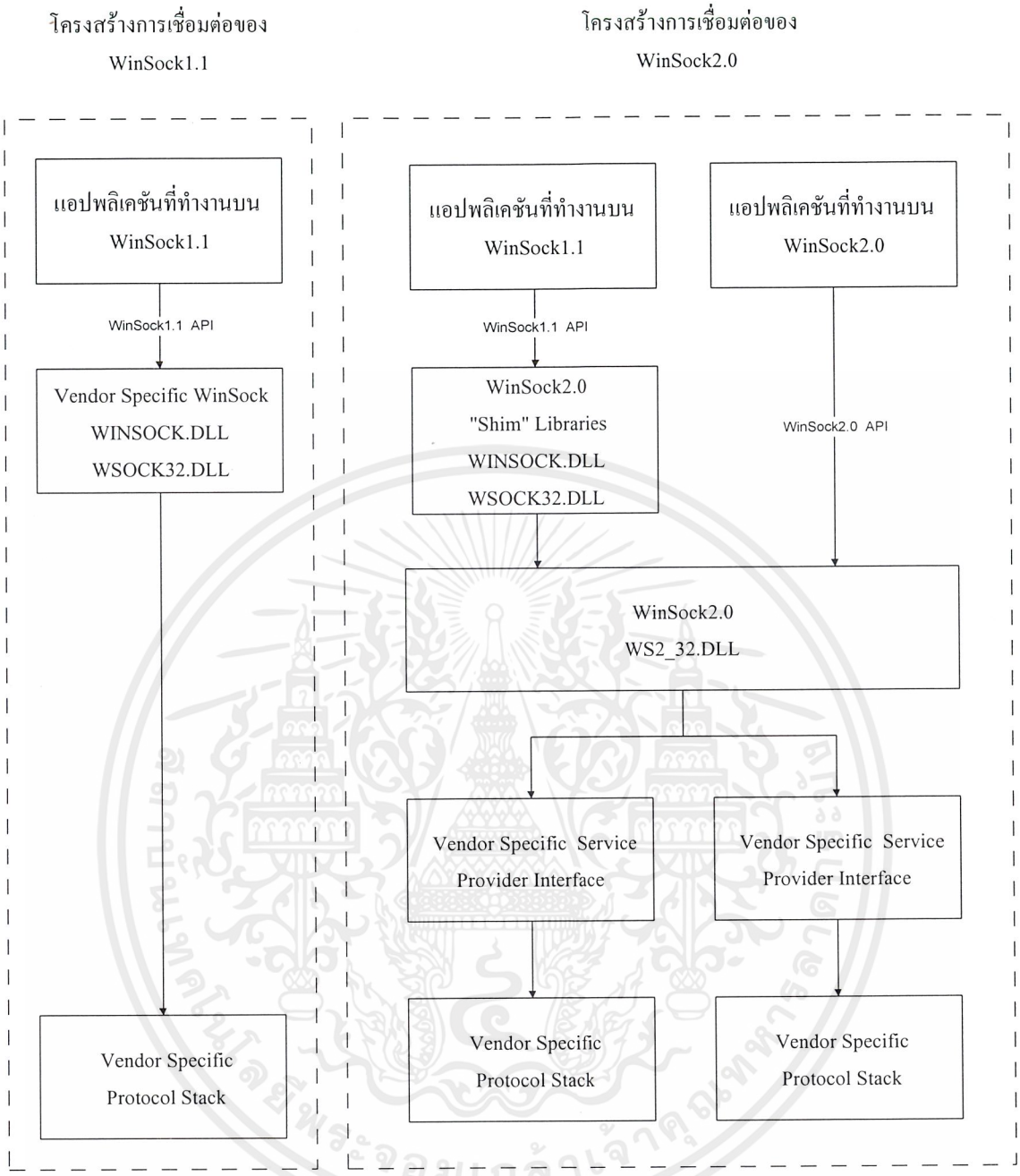
- **อีเวนต์ออปเจ็กต์ (Event Object)** ใน WinSock2.0 นั้นการสังเกต (Notify) เหตุการณ์ (Event) ของเน็ตเวิร์คโดยใช้อีเวนต์ออปเจ็กต์ของ Win32 แทนการใช้เมสเสจ (Messages) ของวินโดว์ ทำให้สามารถเขียนโปรแกรมที่ไม่มีการสร้างวินโดว์ได้เช่นโปรแกรมเซิร์ฟเวอร์ที่ไม่จำเป็นต้องใช้วินโดว์

- **คุณภาพของบริการ (Quality of Service)** WinSock 2.0 สามารถกำหนดคุณภาพของบริการในการส่งข้อมูลได้ ทำให้สามารถส่งข้อมูลประเภทที่ความหน่วงเวลา (Delay) มีผลต่อคุณภาพของข้อมูลเช่นข้อมูลประเภทวีดีโอสตรีม เป็นต้นได้

WinSock 2.0 จะถูกออกแบบโครงสร้างของการเชื่อมต่อที่แตกต่างจาก WinSock1.1 คือ WinSock 2.0 จะประกอบด้วยแอพพลิเคชั่นโปรแกรมอินเตอร์เฟสและเซอร์วิสโพวายเดอร์อินเตอร์เฟส (Service Provider Inerface (SPI)) โดยแต่ละส่วนจะทำหน้าที่ดังนี้

1. **แอพพลิเคชั่นโปรแกรมอินเตอร์เฟส** จะเป็นส่วนที่กำหนดถึงฟังก์ชันและโครงสร้างข้อมูลที่เตรียมเอาไว้ให้โปรแกรมเมอร์เรียกใช้
2. **เซอร์วิสโพวายเดอร์อินเตอร์เฟส** เป็นส่วนที่ใช้เชื่อมต่อกับโปรโตคอลสแตกของบริษัทผู้ขายเข้ากับ WinSock2.0 DLL เพื่อให้สามารถใช้งานได้

การที่ WinSock2.0 DLLไม่ทำการเชื่อมต่อกับโปรโตคอลสแตกของบริษัทผู้ขายโดยตรง แต่จะให้บริษัทผู้ขายสร้างเซอร์วิสโพวายเดอร์อินเตอร์เฟสที่สนับสนุนโปรโตคอลสแตกของตนออกมาเพื่อเชื่อมต่อเข้ากับ WinSock2.0 DLL การที่ใช้การเชื่อมต่อในลักษณะนี้ทำให้แอพพลิเคชั่นที่ทำงานบน WinSock2.0 สามารถเข้าใช้บริการได้จากหลายๆโปรโตคอล ซึ่งแตกต่างจาก WinSock1.1 ที่สามารถใช้งานโปรโตคอล TCP/IP เพียงโปรโตคอลเดียว

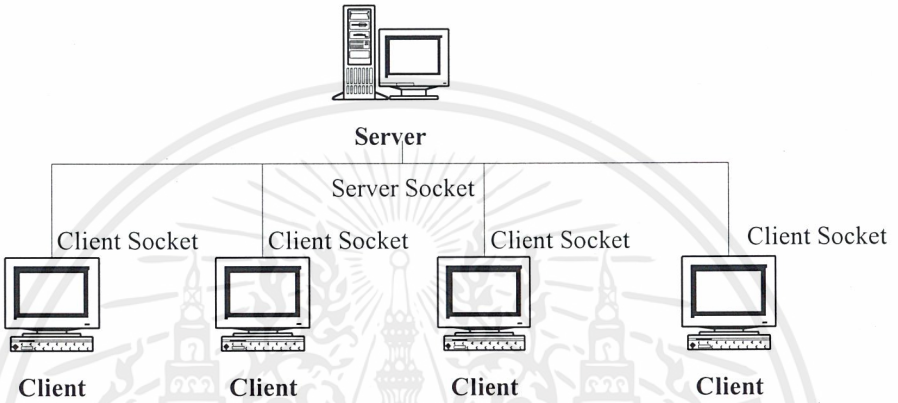


รูปที่4.2 แสดงการเปรียบเทียบโครงสร้างการเชื่อมต่อระหว่าง WinSock1.1 กับ WinSock2.0

4.3 ลักษณะการทำงานของ WinSock

WinSock ถูกออกแบบการทำงานเป็นแบบ**ซอกเก็ตโมเดล (Socket Model)** โดยการทำงานของซอกเก็ตโมเดลนั้น การติดต่อสื่อสารจะเกิดขึ้นระหว่างจุดปลาย (Endpoint) 2 จุด โดยจุดปลายนี้จะเรียกว่าซอกเก็ต โดยซอกเก็ตนี้จะใช้กำหนดจุดปลายในการติดต่อสื่อสาร ดังนั้นก่อนที่การติดต่อสื่อสารจะเกิดขึ้นซอกเก็ตของทั้งสองฝ่ายที่จะติดต่อสื่อสารกันจะต้องถูกสร้างขึ้นก่อน WinSock จะใช้**ไคลเอนต์เซิร์ฟเวอร์โมเดล (Client/Server Model)** เป็นรูปแบบในการติดต่อสื่อสารคือ จะมีแอพพลิเคชั่นตัวหนึ่งทำหน้าที่

เป็นเซิร์ฟเวอร์ (Server) คอยให้บริการกับไคลเอนต์ (Client) ที่เข้ามาขอใช้บริการ โดยตัวเซิร์ฟเวอร์จะต้องทำการสร้างซอกเก็ตก่อน และต้องทำการตั้งชื่อให้กับซอกเก็ตนั้นเพื่อให้ตัวไคลเอนต์สามารถตรวจสอบและค้นหาซอกเก็ตนี้ได้ จากนั้นตัวเซิร์ฟเวอร์ก็จะทำการคอยฟัง (listen) การขอบริการจากตัวไคลเอนต์ ส่วนตัวไคลเอนต์นั้นก็จะต้องสร้างซอกเก็ตขึ้นมาก่อนแล้วจากนั้นจึงทำการค้นหาซอกเก็ตของตัวเซิร์ฟเวอร์โดยใช้ชื่อหรือแอดเดรส (Address) ที่ตัวเซิร์ฟเวอร์ตั้ง จากนั้นจึงทำการสร้างการติดต่อสื่อสาร เมื่อการติดต่อสื่อสารถูกสร้างขึ้นข้อมูลจะสามารถส่งได้ทั้งสองทิศทางคือ เซิร์ฟเวอร์และไคลเอนต์สามารถส่งและรับข้อมูลจากกันและกันได้

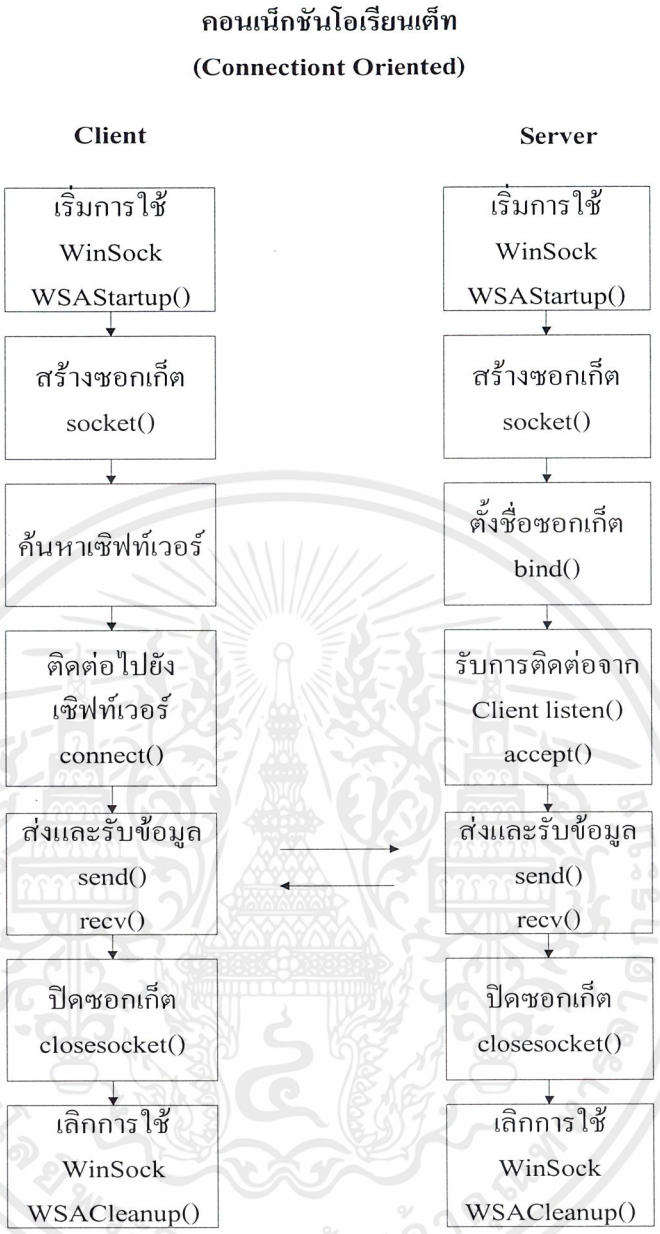


รูปที่4.3 แสดงไคลเอนต์เซิร์ฟเวอร์โมเดล

4.3.1 ชนิดของการติดต่อใน WinSock

จะแบ่งเป็น 2 ชนิดคือ

1. การติดต่อแบบคอนเน็กชันโอเรียนเต็ท (Connection Oriented Connection) ในการติดต่อแบบนี้จะต้องมีการสร้างการติดต่อขึ้นมาก่อนจึงจะสามารถส่งข้อมูลได้ นั่นคือซอกเก็ตของทั้งสองฝ่ายจะต้องถูกสร้างขึ้นก่อนในขั้นตอนการสร้างการติดต่อเท่านั้น จากนั้นจึงส่งข้อมูลโดยใช้ซอกเก็ตที่สร้างขึ้น ตัวอย่างของโปรโตคอลที่ใช้การติดต่อแบบนี้นั้นก็เช่น TCP,ATM เป็นต้น



รูปที่4.4 ขั้นตอนในการติดต่อแบบคอนเน็กชันโอเรียนเต็ท

จากรูปที่ 4.4 จะเห็นว่าเริ่มต้นแอพพลิเคชั่นทางฝั่งเซิร์ฟเวอร์จะเริ่มต้นขึ้นก่อน โดยจะสร้างซอกเก็ตด้วยฟังก์ชัน socket() จากนั้นจะทำการระบุแอดเดรส,พอร์ตและโปรโตคอลที่ใช้ให้กับซอกเก็ตด้วยฟังก์ชัน bind() จากนั้นเซิร์ฟเวอร์ก็จะรอการติดต่อมาจากไคลเอนต์ด้วยฟังก์ชัน listen() และ accept()

ส่วนในฝั่งของไคลเอนต์นั้นจะเริ่มต้นจากการสร้างซอกเก็ตเช่นกันจากนั้นจะทำการหาที่อยู่ของเซิร์ฟเวอร์โดยชื่อหรือแอดเดรสของเซิร์ฟเวอร์ก็ได้ (ถ้าทราบแอดเดรสของเซิร์ฟเวอร์ก็สามารถใช้ได้ทันทีได้ไม่ต้องทำการค้นหา) จากนั้นจึงทำการติดต่อกับเซิร์ฟเวอร์ด้วยฟังก์ชัน connect() จากนั้นก็สามารถที่จะทำการส่งข้อมูลกับเซิร์ฟเวอร์ได้

สุดท้ายเมื่อไม่ต้องการใช้การติดต่อนี้แล้วก็ทำการปิดซอกเก็ตโดยใช้ฟังก์ชัน closesocket()

2. การติดต่อแบบคอนเน็กชันเลส (Connectionless Connection) ในการติดต่อแบบนี้ตัวไคลเอนต์สามารถส่งข้อมูลไปยังเซิร์ฟเวอร์ได้ทันทีโดยไม่ต้องสร้างการติดต่อกับเซิร์ฟเวอร์ขึ้นมาก่อน มักจะใช้กับการส่งข้อมูลจำนวนไม่มากนัก ตัวอย่างของโปรโตคอลที่ทำงานแบบนี้เช่น UDP

คอนเน็กชันเลส (Connectiont less)

รูปที่4.5 แสดงขั้นตอนในการติดต่อแบบคอนเน็กชันเลส

จากรูปที่ 4.5 จะเห็นว่าแอพพลิเคชั่นทางฝั่งเซิร์ฟเวอร์จะเริ่มต้นขึ้นก่อน โดยตัวเซิร์ฟเวอร์จะทำการสร้างซอกเก็ตขึ้นโดยใช้ฟังก์ชัน socket() จากนั้นจะทำการระบุแอดเดรส,พอร์ตและโปรโตคอลที่ใช้ให้กับซอกเก็ตด้วยฟังก์ชัน bind() จากนั้นเซิร์ฟเวอร์จะรอรับการขอใช้บริการจากไคลเอนต์ โดยใช้ฟังก์ชัน recvfrom() สังเกตุว่าการติดต่อแบบนี้ตัวเซิร์ฟเวอร์จะไม่ใช้ฟังก์ชัน listen() และฟังก์ชัน accept()

แต่ใช้ฟังก์ชัน recvfrom() แทนโดยฟังก์ชัน recvfrom() ซึ่งจะทำงานเหมือนกับฟังก์ชัน recv() แต่มันจะบอกตัวเซิร์ฟเวอร์ด้วยว่าข้อมูลนั้นมาจากไหนทำให้เซิร์ฟเวอร์สามารถส่งข้อมูลกลับไปได้โดยใช้ฟังก์ชัน sendto()

ส่วนในฝั่งของไคลเอนต์นั้นจะเริ่มต้นจากการสร้างซอกเก็ตเช่นกันจากนั้นจะทำการหาที่อยู่ของเซิร์ฟเวอร์โดยชื่อหรือแอดเดรสของเซิร์ฟเวอร์ก็ได้ (ถ้าทราบแอดเดรสของเซิร์ฟเวอร์ก็สามารถใช้ได้ทันทีได้ไม่ต้องค้นหา) แต่ต่อจากนี้ตัวไคลเอนต์สามารถส่งข้อมูลถึงตัวเซิร์ฟเวอร์ได้ทันทีโดยไม่ต้องใช้ฟังก์ชัน connect() แต่จะใช้ฟังก์ชัน sendto() ส่งข้อมูลไปได้ทันที

สุดท้ายเมื่อไม่ต้องการใช้การติดต่อนี้แล้วก็ทำการปิดซอกเก็ตโดยใช้ฟังก์ชัน closesocket()

4.3.2 ประเภทของซอกเก็ตใน WinSock

ซอกเก็ตของ WinSock สามารถแบ่งได้เป็น 2 ประเภทดังนี้คือ

1. ซอกเก็ตแบบบล็อก (Blocking Socket) เมื่อมีการใช้ซอกเก็ตประเภทนี้ฟังก์ชันที่เรียกใช้จะไม่มีการคืนค่าจนกว่าฟังก์ชันนั้นจะทำงานเสร็จ คือเมื่อมีการเรียกฟังก์ชันที่ไม่สามารถที่จะทำงานได้เสร็จในทันทีก็จะต้องรอให้ฟังก์ชันทำงานเสร็จก่อนจึงจะไปทำงานอื่นได้ การใช้งานซอกเก็ตประเภทนี้จะทำให้ระบบมีค่าทรูพุต (Throughput) ต่ำ แต่การสร้างและเขียนโปรแกรมกับซอกเก็ตประเภทนี้ทำได้ง่ายและอาจนำเอาเทคนิคอื่นมาช่วยให้ระบบมีค่าทรูพุตสูงขึ้นได้เช่นการทำมัลติเทรด (Multithread) โดยถ้าทำการสร้างซอกเก็ตขึ้นมาด้วยฟังก์ชัน socket() ก็จะได้ซอกเก็ตแบบบล็อก

2.ซอกเก็ตแบบไม่บล็อก (Nonblocking Mode) เมื่อมีการใช้ซอกเก็ตแบบนี้ฟังก์ชันที่เรียกใช้จะคืนค่าทันทีไม่ว่าฟังก์ชันนั้นจะทำงานเสร็จหรือไม่ก็ตาม โดยถ้าฟังก์ชันนั้นทำงานเสร็จทันทีก็ไม่มีอะไรแต่ถ้าฟังก์ชันนั้นไม่สามารถทำงานให้เสร็จได้ทันทีก็จะมีการคืนค่ามาเป็นค่าผิดพลาด (Error) มาแทนโดยถ้าเรียกฟังก์ชัน WSAGetLastError() จะได้ค่าผิดพลาดเป็น WSAEWOULDBLOCK และจะถือว่าการทำงานนั้นไม่สำเร็จทั้งหมด

4.4 การโปรแกรมด้วย WinSock พื้นฐาน

ในหัวข้อนี้จะอธิบายถึงการเขียนโปรแกรมด้วย WinSock พื้นฐานโดยจะใช้ฟังก์ชันใน WinSock2.0 ซึ่งจะเป็นฟังก์ชันที่เพิ่มเติมความสามารถขึ้นมาจากฟังก์ชันเพราะเราต้องการที่จะใช้ความสามารถใหม่ที่เพิ่มขึ้นมาใน WinSock2.0 ในการทำเขียนแอพพลิเคชั่น

4.4.1 การเริ่มต้นใช้ WinSock

จะมีฟังก์ชันที่สำคัญอยู่ 3 ฟังก์ชันคือ

1. **WSAStartup()** จะเป็นฟังก์ชันที่ทุกแอพพลิเคชันที่ทำงานด้วย WinSock จะต้องเรียกใช้ก่อนที่จะใช้งานฟังก์ชันอื่นของ WinSock โดยถ้าทำการเรียกใช้ฟังก์ชันอื่นโดยไม่ทำการเรียกฟังก์ชัน WSAStartup() ก่อน ฟังก์ชันนั้นจะคืนค่าผิดพลาดกลับมาเป็นค่า WSAENOTINITIALISED การเรียกฟังก์ชัน WSAStartup() จะเป็นการเตรียมให้ WinSock DLL สามารถจัดสรรทรัพยากรต่างๆเพื่อให้บริการได้ หรืออาจปฏิเสธการให้บริการถ้าทรัพยากรไม่เพียงพอต่อความต้องการ

2. **WSACleanup()** เป็นฟังก์ชันที่เรียกเมื่อต้องการเลิกใช้ WinSock จะจัดการปลดปล่อย(Free) ทรัพยากรที่จองไว้

3. **WSAGetLastError()** เป็นฟังก์ชันที่เรียกใช้เพื่อตรวจสอบค่าผิดพลาดที่เกิดขึ้นว่าเกิดจากสาเหตุอะไร

4.4.2 ฟังก์ชันที่สำคัญที่ใช้ในการสร้างการติดต่อ

จะมีฟังก์ชันที่สำคัญดังนี้

1. **WSASocket(int af, int type, int protocol , LPWSAPROTOCOL_INFOA lpProtocolInfo, GROUP g, DWORD dwFlags)** เป็นฟังก์ชันที่ใช้สร้างซอกเก็ตที่ใช้ในการติดต่อ

2. **closesocket(SOCKET s)** เป็นฟังก์ชันที่ใช้ในการปิดซอกเก็ตที่ไม่ต้องการใช้อีกต่อไป

3. **bind((SOCKET s, const struct sockaddr FAR * name,int namelen)** เป็นฟังก์ชันที่เซิร์ฟเวอร์ใช้ในการตั้งชื่อให้ซอกเก็ตเพื่อให้ไคลเอนต์สามารถค้นหาที่อยู่ของเซิร์ฟเวอร์ได้

4. **listen(SOCKET s, int backlog)** เป็นฟังก์ชันที่เซิร์ฟเวอร์ใช้เซตให้ซอกเก็ตรอการขอสร้างการติดต่อจากไคลเอนต์โดยสามารถกำหนดได้ว่าจะให้มีไคลเอนต์รอการตอบรับได้กี่ตัว

5. **WSAAccept(SOCKET s, struct sockaddr FAR * addr, LPINT addrlen, LPCONDETIONPROC lpfnConditon ,DWORD dwCallbackData)** เป็นฟังก์ชันที่เซิร์ฟเวอร์ใช้ยอมรับการขอสร้างการติดต่อที่ไคลเอนต์ขอเข้ามา

6. **WSARecv(SOCKET s, LPWSABUF lpBuffers, DWORD dwBufferCount, LPDWORD lpNumverOfBytesRecvd, LPDWORD lpFlags, LPWSAOVERLAPPED lpOverlapped,LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine)** เป็นฟังก์ชันที่ใช้รับข้อมูลที่อีกฝ่ายหนึ่งส่งมา

7. **WSASend(SOCKET s, LPWSABUF lpBuffers, DWORD dwBufferCount, LPDWORD lpNumverOfBytesSent, LPDWORD dwFlags, LPWSAOVERLAPPED lpOverlapped,LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine)** เป็นฟังก์ชันที่ใช้ส่งข้อมูลไปให้อีกฝ่ายหนึ่ง

8. **WSAConnect(SOCKET s,const struct sockaddr FAR* name, int namelen , LPWSABUF lpCallerData,LPWSABUF lpCalleeData, LPQOS lpSQOS,LPQOS lpGQOS)** เป็นฟังก์ชันที่ไคลเอนต์ใช้ในการขอสร้างการติดต่อกับเซิฟท์โดยสามารถกำหนดคุณภาพของการบริการได้

บทที่ 5
แนะนำ DirectShow SDK

DirectShow SDK เป็นชุดพัฒนาโปรแกรมบนวินโดว์สของไมโครซอฟต์ เพื่อช่วยในการสร้างโปรแกรมจับสตรีมของมัลติมีเดีย (Capture Multimedia Streams) จากอุปกรณ์ต่างๆ และทำการแสดงผลสตรีมของมัลติมีเดีย (Playback Multimedia Streams) บนคอมพิวเตอร์ โดยตัว DirectShow SDK จะมีเครื่องมือช่วยทำหน้าที่เหล่านี้ ทำให้ผู้ใช้ไม่ต้องเขียนโปรแกรมขึ้นมาเองทั้งหมด ช่วยให้การพัฒนาโปรแกรมบนวินโดว์ส มีความเป็นมาตราฐาน สะดวก รวดเร็ว และมีประสิทธิภาพมากขึ้น สำหรับเนื้อหาในบทนี้จะมีดังต่อนี้

- DirectShow คืออะไร ?
- สถาปัตยกรรมของ DirectShow
- ตัวจัดการฟิลเตอร์กราฟ (Filter Graph Manager) และฟิลเตอร์กราฟ (Filter Graph)
- ฟิลเตอร์ (Filter) และพิน (Pin)
- การใช้งานส่วนประกอบ (Component) ต่างๆ ใน DirectShow
- การเขียนโปรแกรมด้วย DirectShow เบื้องต้น

5.1 DirectShow คืออะไร?

DirectShow SDK เป็นชุดพัฒนาโปรแกรมบนวินโดวส์ของไมโครซอฟต์ โดยจะประกอบอยู่ในชุดพัฒนาโปรแกรมบนวินโดวส์ของไมโครซอฟต์ที่ชื่อว่า Microsoft DirectX Media 6.0 SDK ซึ่งสามารถดาวน์โหลดได้จากที่เวปไซต์ของไมโครซอฟต์ ที่ http://www.microsoft.com รวมทั้งข้อมูลรายละเอียดเกี่ยวกับส่วนประกอบต่างๆก็สามารถหาได้จากเวปไซต์ดังกล่าว

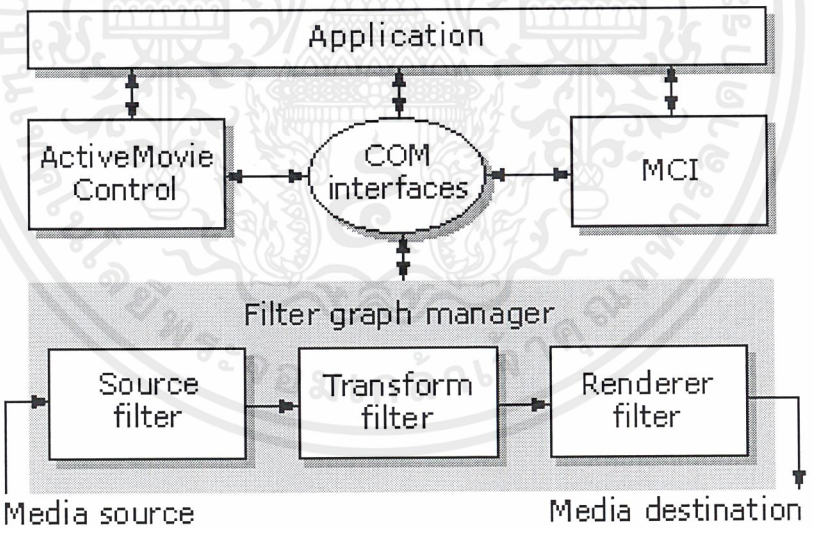
สำหรับ DirectShow SDK นั้นจะเป็นชุดพัฒนาโปรแกรม เพื่อช่วยนักพัฒนาในการสร้างโปรแกรมเพื่อจัดการกับมัลติมีเดียสตรีม โดยจะสามารถจับสตรีมของมัลติมีเดียจากอุปกรณ์ต่างๆ และทำการแสดงผลสตรีมของมัลติมีเดียบนคอมพิวเตอร์ได้ตามต้องการ สำหรับสตรีมของมัลติมีเดียที่จะแสดงผลสามารถมีได้หลายรูปแบบ เช่น MPEG ,Apple QuickTime(MOV) ,audio-video interleaved(AVI) และ WAV โดยสตรีมของมัลติมีเดียสามารถจะนำมาจากข้อมูลในคอมพิวเตอร์ หรือจะนำมาจากอินเตอร์เน็ตก็ได้

หัวใจของ DirectShow ก็คือ ส่วนประกอบที่เรียกว่า ฟิลเตอร์ซึ่งจะถูกจัดการโดย ตัวจัดการฟิลเตอร์กราฟเพื่อคอยควบคุมดูแลสตรีมของมัลติมีเดีย ให้สามารถทำงานได้ตามต้องการ

5.2 สถาปัตยกรรมของ DirectShow

สถาปัตยกรรมของ DirectShow ประกอบด้วยส่วนประกอบที่เรียกว่า ฟิลเตอร์ซึ่งจะทำหน้าที่ในการควบคุมและประมวลผลมัลติมีเดียสตรีม โดยฟิลเตอร์จะประกอบด้วยส่วนอินพุทพิน (input pin) หรือส่วนเอาท์พุทพิน(output pin) หรืออาจจะประกอบด้วยทั้งสองส่วน แล้วแต่หน้าที่การทำงานของแต่ละฟิลเตอร์ สำหรับการทำงานฟิลเตอร์จะถูกนำมาเชื่อมต่อกัน โดยตัวจัดการฟิลเตอร์กราฟ(filter graph manager) เพื่อให้ได้การทำงานตามต้องการ ซึ่งโดยปกติตัวจัดการฟิลเตอร์จะทำการสร้างฟิลเตอร์ที่จำเป็นให้อย่างอัตโนมัติในการแสดงผลมัลติมีเดียสตรีม ทำให้ผู้ใช้ไม่ต้องระบุชนิดของฟิลเตอร์ที่จำเป็นในการเขียนโปรแกรม แต่ก็สามารถที่จะเปลี่ยนแปลงชนิดของฟิลเตอร์แต่ละชนิดได้ตามต้องการ โดยแต่ละฟิลเตอร์ที่เลือกใช้จะต้องสนับสนุนซึ่งกันและกัน

การเรียกใช้ส่วนประกอบต่างๆของ DirectShow ผู้ใช้จะต้องใช้เรียกใช้ผ่านส่วนเชื่อมต่อของ Component Object Model (COM) เพื่อให้แอพพลิเคชั่นที่เขียนขึ้นสามารถเข้าถึงฟิลเตอร์กราฟได้ โดยแอพพลิเคชั่นสามารถที่จะเรียกใช้ส่วนเชื่อมต่อไปยังตัวจัดการฟิลเตอร์เพื่อให้ควบคุมดูแลมัลติมีเดียสตรีมทั้งหมดโดยอัตโนมัติ หรือ อาจจะเรียกใช้เฉพาะส่วนเชื่อมต่อไปยังฟิลเตอร์ที่ต้องการเท่านั้นก็ได้



รูปที่ 5.1 แสดงการเชื่อมต่อของแอพพลิเคชั่นไปยังตัวจัดการฟิลเตอร์ (Filter Graph Manager)

จากรูปแอพพลิเคชั่นจะเรียกใช้ส่วนเชื่อมต่อไปยังตัวจัดการฟิลเตอร์เพื่อให้ควบคุมดูแลมัลติมีเดียสตรีมโดยตัวจัดการฟิลเตอร์จะทำการสร้างส่วนเชื่อมต่อไปยังฟิลเตอร์ที่จำเป็นให้อย่างอัตโนมัติ ซึ่งจากรูปจะเป็น Source filter ,Transform filter และRenderer filter เพื่อให้สามารถประมวลผลมัลติมีเดียสตรีมและแสดงผลได้อย่างถูกต้อง สำหรับรายละเอียดของแต่ละฟิลเตอร์จะกล่าวในหัวข้อต่อไป

5.3 ตัวจัดการฟิลเตอร์กราฟ (Filter Graph Manager) และฟิลเตอร์กราฟ (Filter Graph)

การใช้งานตัวจัดการฟิลเตอร์กราฟ นั้นผู้ใช้ไม่จำเป็นต้องเข้าใจการทำงานของฟิลเตอร์แต่ละฟิลเตอร์อย่างละเอียด เพราะตัวจัดการฟิลเตอร์กราฟ จะทำการสร้างส่วนเชื่อมต่อไปยังฟิลเตอร์ที่จำเป็นให้อย่างอัตโนมัติ โดยผู้ใช้ไม่ต้องระบุชนิดของฟิลเตอร์ที่ต้องการใช้ในการจัดการกับมัลติมีเดียสตรีม ตัวจัดการฟิลเตอร์กราฟจะทำการตรวจสอบ ,ประมวลผล และกำหนดชนิดของฟิลเตอร์ต่างๆที่ต้องใช้ให้อย่างอัตโนมัติ แต่อย่างไรก็ตามหากผู้ใช้ต้องการเพิ่มส่วนเชื่อมต่อไปยังฟิลเตอร์อื่นเพื่อให้ได้ความสามารถที่ต้องการก็สามารถทำได้ แต่ผู้ใช้จำเป็นที่จะต้องเข้าใจการทำงานของฟิลเตอร์นั้นอย่างละเอียดก่อน

ฟิลเตอร์กราฟ คือ ฟิลเตอร์หลายชนิดที่มีการทำงานแต่ละหน้าที่ต่างกัน แต่ถูกนำมาเชื่อมต่อกันเพื่อให้ได้ความสามารถในการควบคุม และประมวลผลมัลติมีเดียสตรีม ให้ได้ผลลัพธ์ตามต้องการ

สำหรับฟิลเตอร์นั้นสามารถแบ่งออกได้เป็น 3 ประเภทใหญ่ๆ ดังต่อไปนี้

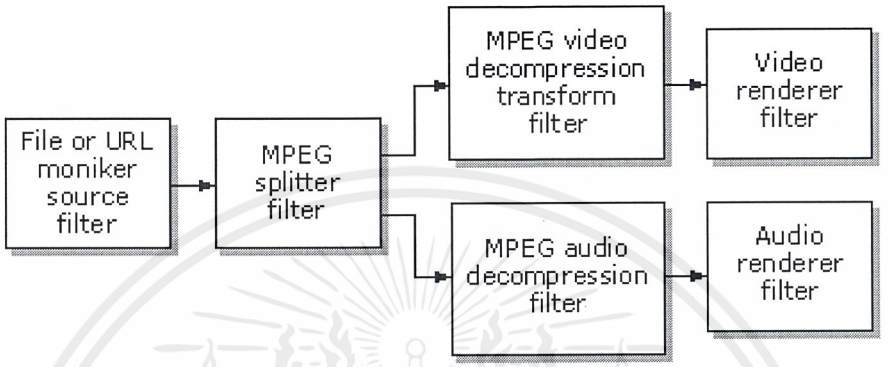
- **ซอร์สฟิลเตอร์** (Source filter) เป็นฟิลเตอร์ที่ทำหน้าที่ในการรับข้อมูลจากแหล่งข้อมูลต่างๆ เช่น ข้อมูลจากดิสก์,ข้อมูลจากอินเตอร์เน็ต ,ข้อมูลจากวีซีอาร์(VCR) และส่งต่อให้ฟิลเตอร์กราฟหรือฟิลเตอร์ที่เชื่อมต่อถัดไปผ่านเอาท์พุทพิน (Output Pin)
- **ทรานฟอร์มฟิลเตอร์** (Transform filter) เป็นฟิลเตอร์ที่ทำหน้าที่ในการนำเอาข้อมูลที่ได้มาประมวลผลให้ได้ผลลัพธ์จากนั้นก็ส่งต่อไปยังฟิลเตอร์ถัดไปผ่านเอาท์พุทพิน
- **เรนเดอร์ฟิลเตอร์** (Render filter) เป็นฟิลเตอร์ที่ทำหน้าที่ในการนำเอาข้อมูลที่ได้มาแสดงผลไปยังอุปกรณ์ฮาร์ดแวร์ต่างๆ ที่สนับสนุนมัลติมีเดียสตรีมที่ได้รับ

จากฟิลเตอร์ 3 ประเภทที่ได้กล่าวไปก็ยังมีฟิลเตอร์อื่นๆอีก เช่น เอฟเฟ็คฟิลเตอร์(Effect filter) ซึ่งจะใช้การสร้างเอฟเฟ็คเพิ่มเติมให้กับข้อมูลที่ได้รับ และยังมีอีกหลายฟิลเตอร์ที่มีประโยชน์ในการใช้งานเพื่อเพิ่มเติมความสามารถให้กับโปรแกรมที่ทำการพัฒนา โดยจะสามารถดูรายละเอียดได้จากเอกสารของ DirectShow SDK

ต่อไปนี้จะเป็นตัวอย่างของฟิลเตอร์กราฟที่ใช้ในการแสดงภาพวีดีโอ MPEG จากไฟล์ที่อยู่ในดิสก์ โดยจะประกอบด้วยฟิลเตอร์ที่จำเป็นในการใช้งานดังต่อไปนี้

- ซอร์สฟิลเตอร์ทำหน้าที่ในการอ่านข้อมูลจากไฟล์ในดิสก์ (file or URL moniker source filter)
- เอ็มเพคสปริทเตอร์ฟิลเตอร์ (MPEG Splitter filter) ทำหน้าที่ในการรับสตรีมของวีดีโอ จากนั้นทำการแยกสตรีมออกเป็น 2 ส่วน คือ วีดีโอสตรีม และออดิโอสตรีม
- ทรานฟอร์มฟิลเตอร์ที่ทำหน้าที่ในการถอดรหัสข้อมูลของวีดีโอ (MPEG video decompression tranform filter)

- ทรานฟอร์มฟิลเตอร์ที่ทำหน้าที่ในการถอดรหัสข้อมูลของออดิโอ (MPEG audio decompression tranfrom filter)
- เรนเดอร์ฟิลเตอร์ที่ทำหน้าที่ในการแสดงภาพวีดีโอบนจอภาพคอมพิวเตอร์(Video renderer filter)
- เรนเดอร์ฟิลเตอร์ที่ทำหน้าที่ในการส่งสัญญาณเสียงไปยังซาวน์การ์ด (Audio renderer filter)



รูปที่ 5.2 แสดงฟิลเตอร์กราฟที่ใช้ในการแสดงภาพวีดีโอ MPEG จากไฟล์ที่อยู่ในดิสก์

จากรูปจะแสดงให้เห็นการเชื่อมต่อกันของฟิลเตอร์แต่ละชนิด โดยลำดับของการเชื่อมต่อจะต้องเป็นไปดังรูปการทำงานของฟิลเตอร์จึงจะสามารถทำงานได้ถูกต้องตามหน้าที่ ฟิลเตอร์แต่ละชนิดที่ถูกเลือกใช้ดังรูปนั้นผู้ใช้อาจจะเป็นผู้กำหนดเอง และสร้างลำดับการเชื่อมต่อเองก็ได้ แต่จากที่กล่าวไปแล้วผู้ใช้สามารถให้ตัวจัดการฟิลเตอร์กราฟเป็นผู้กำหนดฟิลเตอร์แต่ละชนิดเองโดยอัตโนมัติก็ได้ ซึ่งจะทำให้สะดวก รวดเร็ว ถูกต้อง และมีประสิทธิภาพ สำหรับรายละเอียดการเรียกใช้ส่วนเชื่อมต่อไปยังตัวจัดการฟิลเตอร์กราฟจะกล่าวถึงในบทต่อๆไป

5.4 ฟิลเตอร์ (Filter) และพิน (Pin)

ส่วนประกอบที่จะนำมาใช้ในการส่งข้อมูลแบบสตรีม คือ ฟิลเตอร์และพิน โดยฟิลเตอร์นี้จะเป็นออปเจกต์ (Object) ของ COM (Componet Object Model) ที่มีหน้าที่ต่างๆตามที่กำหนดไว้ตามแต่ละชนิดของฟิลเตอร์ สำหรับพินก็จะเป็นออปเจกต์ของ COM เหมือนกันแต่จะถูกสร้างขึ้นตามชนิดของแต่ละฟิลเตอร์ ซึ่งพินจะเป็นจุดที่ใช้เชื่อมต่อฟิลเตอร์เข้าด้วยกัน ดังรูป

รูปที่ 5.3 แสดงการเชื่อมต่อฟิลเตอร์แต่ละชนิดเข้าด้วยกันโดยใช้พินเป็นจุดเชื่อมต่อ

การเชื่อมต่อแต่ละฟิลเตอร์เข้าด้วยกันโดยใช้พินเป็นจุดเชื่อมต่อนั้น จะเป็นไปในทิศทางเดียว กล่าวคือ เอาท์พุทพินจะต้องเชื่อมต่อกับอินพุทพิน ส่วนอินพุทก็ต้องเชื่อมต่อกับเอาท์พุทพิน เพื่อให้สตรีมของมัลติมีเดียไหลผ่านฟิลเตอร์แต่ละฟิลเตอร์อย่างถูกต้อง

อินพุทพิน จะทำหน้าในการรับข้อมูลเข้าไปในฟิลเตอร์ ส่วนเอาท์พุทจะทำหน้าที่ในการส่งข้อมูลออกจากฟิลเตอร์ สำหรับซอร์สฟิลเตอร์นั้นจะประกอบด้วย 1 เอาท์พุทพินเท่านั้นเพื่อใช้ในการส่งสตรีมของมัลติมีเดียจากแหล่งข้อมูล ส่วนทรานฟอร์มฟิลเตอร์นั้นจะประกอบด้วย 1 อินพุทพิน และ 1 เอาท์พุทพิน โดยจะรับข้อมูลมาจากอินพุทพิน และทำการประมวลผลให้ได้ผลลัพธ์ จากนั้นจึงส่งผลลัพธ์ที่ได้ไปยังเอาท์พุทพิน เพื่อส่งข้อมูลต่อไปยังอินพุทพินของเรนเดอร์ฟิลเตอร์ จากนั้นเรนเดอร์ฟิลเตอร์ก็จะทำการแสดงผลที่ได้ออกไปยังอุปกรณ์ที่ได้กำหนดไว้ตามชนิดของเรนเดอร์ฟิลเตอร์

สำหรับการเชื่อมต่อฟิลเตอร์แต่ละชนิดเข้าด้วยกันนั้น ผู้ใช้จะต้องตรวจสอบชนิดของอินพุทพินและเอาท์พุทพิน ของแต่ละฟิลเตอร์ก่อนด้วยว่าสนับสนุนการเชื่อมต่อกันหรือไม่ เพราะถ้าเชื่อมต่อพินแต่ละชนิดไม่ถูกต้องแล้วฟิลเตอร์จะไม่สามารถทำงานได้ เช่นการเชื่อมต่อซอร์สฟิลเตอร์เข้ากับเรนเดอร์ฟิลเตอร์โดยตรงโดยไม่ผ่าน ทรานฟอร์มฟิลเตอร์ นั้นจะทำให้ฟิลเตอร์ไม่สามารถทำงานได้ สำหรับรายละเอียดการทำงานของแต่ละฟิลเตอร์สามารถศึกษาได้จากเอกสารของ DirectShow SDK

5.5 การใช้งานส่วนประกอบ (Component) ต่างๆ ใน DirectShow

ฟิลเตอร์ ,ฟิลเตอร์กราฟ ,ตัวจัดการฟิลเตอร์ และส่วนประกอบของ DirectShow ทั้งหมดนั้นถูกออกแบบโดยมีพื้นฐานมาจาก COM (Componet Object Model) ดังนั้นในการที่จะเรียกใช้งานส่วนประกอบเหล่านี้ จึงจำเป็นที่จะต้องเข้าใจพื้นฐานของ COM เบื้องต้นก่อน

COM เป็นมาตราฐานของไบนารีที่ได้กำหนดวิธีการสร้าง และทำลายออปเจกต์ไว้อย่างเป็นมาตราฐานดังนั้นหากต้องการนำเอา COM ไปใช้ก็เพียงแต่ทำตามมาตราฐานที่ได้กำหนดไว้ก็สามารถใช้งานได้ โดยแอพพลิเคชั่นที่ใช้งานไม่จำเป็นต้องเขียนด้วยภาษา C++ เท่านั้นจะเป็นภาษาอะไรก็ได้ แต่ต้องสนับสนุนการเรียกใช้ตารางของพอยท์เตอร์ไปยังฟังก์ชันต่างๆ

COM อินเตอร์เฟส (Interface) คือ ฟังก์ชันที่ทำหน้าต่างๆตามที่ได้กำหนดไว้ โดยใน DirectShow ก็มีอินเตอร์เฟชมากมายให้เรียกใช้แล้วแต่ความต้องการของผู้ใช้ ตัวอย่างเช่น IAsyncReader เป็นอินเตอร์เฟสที่ใช้ในการอ่าน สตรีมของข้อมูล เข้ามายังฟิลเตอร์ สำหรับแต่ละอินเตอร์เฟสนั้นจะนำหน้าด้วย globally unique interface identifier (IID)

COM ออปเจกต์ คือ อินสแตนซ์ (Instance) ของ COM คลาส (Class) โดย COM ออปเจกต์นั้นสามารถที่จะเรียกใช้ COM อินเตอร์เฟสได้ตามต้องการ แต่ COM อินเตอร์เฟสที่ถูกเรียกใช้จะต้องสนับสนุนการใช้งานกับ COM ออปเจกต์นั้นๆด้วย สำหรับ COM คลาสแต่ละชนิดจะนำหน้าด้วย Globally unique class identifier (CLSID) การสร้าง COM ออปเจกต์สามารถทำได้หลายวิธี ดังคำสั่งต่อไปนี้ CoCreateInstance หรือ IclassFactory::CreateInstance หรือ อาจใช้คำสั่ง new ในภาษา C++ ก็ได้ เมื่อทำการสร้างอินสแตนซ์ของ COM คลาสแล้วจะได้พอยท์เตอร์ที่ชี้ไปยังชี้ไปยังอินเตอร์เฟสของออปเจกต์นั้นๆ

ตัวอย่างที่ 5-1 การสร้างอินสแตนซ์ของคลาส CLSID_FilterGraph และเรียกใช้อินเตอร์เฟส IID_IGraphBulider

```
HRESULT Hr;        //ประกาศตัวแปรที่ใช้เก็บผลลัพธ์ของคำสั่ง CoCreateInstance(…)
IGraphBuider  *pigb = NULL;     // ประกาศตัวแปรพอยท์เตอร์เพื่อชี้ไปยัง IGraphBuilder อินเตอร์เฟส
Hr = CoCreateInstance(CLSID_FilterGraph,NULL,CLSCTX_INPROC_SERVER,IID_IGraphBuilder,
      (void**)&pigb);
```

จากตัวอย่างที่ 5-1 Hr คือตัวแปรที่ใช้เก็บผลลัพธ์จากการเรียกใช้คำสั่ง CoCreateInstance(…) ถ้าเป็นผลสำเร็จ Hr จะมีค่าเท่ากับ S_OK และตัวแปร pigb ก็จะเก็บพอยท์เตอร์ที่ชี้ไปยังอินเตอร์เฟชส IID_IGraphBuider ถ้าไม่สำเร็จก็จะมีค่าเท่ากับ NULL

ตัวอย่างที่ 5-2 การเรียกใช้อินเตอร์เฟส IID_IMediaContral จากพอยท์เตอร์ของ IGraphBuilder อินเตอร์เฟส

```
HRESULT Hr;
IMediaControl  *pimc = NULL;    // ประกาศตัวแปรพอยท์เตอร์เพื่อชี้ไปยัง IMediaControl อินเตอร์เฟส
Hr = pigb->QueryInterface(IID_IMediaControl, (void**)&pimc);
```

จากตัวอย่างที่ 5-2 ถ้าเป็นผลสำเร็จ Hr จะมีค่าเท่ากับ S_OK และตัวแปร pimc ก็จะเก็บพอยท์เตอร์ที่ชี้ไปยังอินเตอร์เฟส IID_IMediaControl ถ้าไม่สำเร็จก็จะมีค่าเท่ากับ NULL สำหรับการเรียกใช้คำสั่ง QueryInterface() นั้นจำเป็นที่จะต้องสร้างอินเตอร์เฟสหลักก่อนโดยใช้คำสั่ง CoCreateInstance() จึงจะสามารถใช้งานคำสั่ง QueryInterface() ได้ และการใช้งานคำสั่ง QueryInterface() นั้นจะเป็นผลสำเร็จก็ต่อเมื่อ อินเตอร์เฟสที่เรียกใช้สนับสนุนการใช้งานกับคลาสที่ระบุในคำสั่ง CoCreateInstance()

5.6 การเขียนโปรแกรมด้วย DirectShow เบื้องต้น

ในหัวข้อนี้จะแสดงตัวอย่างของการใช้งาน DirectShow SDK เพื่อสร้างโปรแกรมในการแสดงภาพมัลติมีเดียเบื้องต้น สำหรับการใช้งานนั้นก่อนอื่นจะต้องทำการติดตั้งชุดพัฒนาโปรแกรม DirectX Media 6.0 SDK ลงบนเครื่องคอมพิวเตอร์ก่อน และการเขียนโปรแกรมจะใช้ Microsoft Visual C++ 6.0 เป็นคอมไพเลอร์ในการเขียนโปรแกรม

ตัวอย่างการเขียนโปรแกรมที่จะแสดงดังต่อไปนี้ จะเป็นเพียงพื้นฐานการใช้งานอินเตอร์เฟสต่างๆของ DirectShow ในการแสดงภาพวีดีโอ สำหรับฟิลเตอร์ต่างที่จำเป็นต้องใช้ในการจัดการกับไฟล์วีดีโอที่ต้องการแสดงผลนั้น จะใช้ตัวจัดการฟิลเตอร์กราฟเป็นผู้กำหนดให้ทั้งหมด โดยที่ผู้ใช้ไม่ต้องระบุหรือเข้าใจการทำงานในส่วนนี้เลย เพียงแต่เรียกใช้ฟังก์ชั่นที่มีอยู่ในตัวจัดการฟิลเตอร์กราฟให้ถูกต้องเท่านั้น ก็จะสามารถสร้างโปรแกรมได้อย่างง่ายดาย สำหรับการสร้างโปรแกรมจะมีขั้นตอนดังต่อไปนี้

1. กำหนดไฟล์เฮดเดอร์ที่จำเป็น ซึ่งจะใช้เพื่อให้สามารถเรียกใช้อินเตอร์เฟส และคลาสต่างๆของ DirectShow ได้

```
#include <streams.h>
#include <mmsystem.h>
```

2. กำหนดตัวแปรพอยท์เตอร์ให้กับอินเตอร์เฟสแต่ละชนิดต้องใช้ในการแสดงผลวีดีโอ

```
IGraphBuilder   *pigb  = NULL;
IMediaControl   *pimc  = NULL;
IMediaEventEx  *pimex  = NULL;
IVideoWindow   *pivw   = NULL;
```

IGraphBuider เป็นอินเตอร์เฟสที่ทำหน้าที่เป็นตัวจัดการฟิลเตอร์กราฟ โดยจะทำหน้าที่ในการสร้างฟิลเตอร์กราฟที่จำเป็นในการแสดงภาพวีดีโอให้อย่างอัตโนมัติ

ImediaControl เป็นอินเตอร์เฟสที่ทำหน้าที่ในการควบคุมการแสดงภาพวีดีโอบนจอภาพคอมพิวเตอร์ โดยผู้ใช้สามารถสั่งให้ เล่นภาพ หรือ หยุดภาพได้ผ่านทางอินเตอร์เฟสนี้

IMediaEventEx เป็นอินเตอร์เฟสที่ทำหน้าที่ในการควบคุมเหตุการ์รต่างที่เกิดขึ้นขณะแสดงภาพวีดีโอ เช่น เมื่อเล่นภาพจบแล้วผู้ใช้ต้องการให้ทำอะไรต่อ สามารถกำหนดได้

IVideoWindow เป็นอินเตอร์เฟสที่ทำหน้าที่ในการควบคุมการแสดงภาพบนจอภาพคอมพิวเตอร์ โดยผู้สามารถกำหนดตำแหน่งของการแสดงภาพ หรือขนาดของการแสดงภาพได้ตามต้องการ

3. กำหนดชื่อของไฟล์วีดีโอที่ต้องการแสดงภาพ

```
TCHAR *szFilename = "c:\\dxmedia\\movie\\movie.avi"; //ชื่อไฟล์ที่ต้องการแสดงภาพวีดีโอ
WCHAR wFile[MAX_PATH];
MultiByteToWideChar (CP_ACP,0,szFilename,-1,wFile,MAX_PATH);
```

การกำหนดชื่อที่ต้องการแสดงภาพวีดีโอนั้น จะต้องทำดังที่ได้แสดงไว้ โดยจะต้องทำการเปลี่ยนชื่อไฟล์ที่กำหนดให้เป็น Unicode (wide character) โดยใช้ฟังก์ชั่น MultiByteToWideChar() ในการเปลี่ยนชื่อไฟล์ และจะเก็บผลลัพธ์ที่ได้ไว้ที่ตัวแปร wFile ซึ่งจะต้องถูกใช้ต่อไปโดย IGraphBuilder อินเตอร์เฟส

4. สร้างอินสแตนซ์ของตัวจัดการฟิลเตอร์กราฟ โดยจะต้องระบุอินเตอร์เฟสเป็น IGraphBuilder

```
HRESULT Hr;
Hr = CoCreateInstance(CLSID_FilterGraph,
                      NULL,
                      CLSCTX_INPROC_SERVER,
                      IID_IGraphBuilder,
                      (void**)&pigb);
```

Hr คือตัวแปรที่ใช้เก็บผลลัพธ์จากการเรียกใช้คำสั่ง CoCreateInstance(…) ถ้าเป็นผลสำเร็จ Hr จะมีค่าเท่ากับ S_OK และตัวแปร pigb ก็จะเก็บพอยท์เตอร์ที่ชี้ไปยังอินเตอร์เฟช IID_IGraphBuider ถ้าไม่สำเร็จก็จะมีค่าเท่ากับ NULL พอยท์เตอร์ pigb จะเป็นอินเตอร์เฟสหลักในการเรียกใช้อินเตอร์เฟสอื่นๆอีกต่อไป

5. เรียกใช้อินเตอร์เฟสต่างๆที่จำเป็นต้องใช้ตามตัวแปรที่ได้กำหนดไว้ในตอนแรก

```
pigb->QueryInterface(IID_IMediaControl,(void**)&pimc);
pigb->QueryInterface(IID_IMediaEventEx,(void**)&pime);
pigb->QueryInterface(IID_IVideoWindow,(void**)&pivw);
```

ถ้าการเรียกใช้อินเตอร์เฟสเป็นผลสำเร็จตัวแปรพอยท์เตอร์ pime, pime, pivw ก็จะชี้ไปยังอินเตอร์เฟสที่ต้องการ ทำให้สามารถเรียกใช้ฟังก์ชันต่างๆของแต่ละอินเตอร์เฟสได้ตามต้องการ สำหรับรายละเอียดของฟังก์ชันต่างๆ สำหรับแต่ละฟิลเตอร์สามารถดูได้จากเอกสารของ DirectShow SDK

6. เรียกใช้ฟังก์ชัน RenderFile() ของตัวจัดการฟิลเตอร์กราฟเพื่อให้สร้างฟิลเตอร์กราฟ ซึ่งประกอบด้วยฟิลเตอร์ที่จำเป็นในการแสดงภาพวีดีโอของไฟล์ที่กำหนดให้อย่างอัตโนมัติ

```
HRESULT hr ;
hr = pigb->RenderFile(wFile,NULL);
```

ถ้าเป็นผลสำเร็จ hr จะมีค่าเท่ากับ S_OK ซึ่งแสดงว่าตัวจัดการฟิลเตอร์กราฟสามารถสร้างฟิลเตอร์ต่างๆที่จำเป็นในการแสดงภาพวีดีโอได้เสร็จเรียบร้อย

7. เรียกใช้ฟังก์ชัน SetNotifyWindow() จาก IMediaEventEx อินเตอร์เฟส

```
#define WM_GRAPHNOTIFY WM_USER+13      //กำหนดค่าคงที่ให้กับวินโดวส์เมสเสจ
HWND ghApp;      //กำหนดค่าตัวแปรที่ใช้เก็บแฮนเดิลวินโดวส์ (Handle window) ที่ใช้รับเมสเสจ
Hr = pimex->SetNotifyWindow((OAHWND)ghApp, WM_GRAPHNOTIFY, 0);
```

ถ้าคำสั่งเป็นผลสำเร็จ วินโดวส์ที่ถูกกำหนดโดยตัวแปร ghApp จะเป็นผู้รับเหตุการณ์ต่างๆที่เกิดขึ้นจากฟิลเตอร์กราฟ โดยถ้ามีเหตุการณ์เกิดขึ้น DirectShow จะส่งเมสเสจไปยังวินโดวส์ที่กำหนด โดยค่าของเมสเสจที่เกิดขึ้นจะถูกเก็บอยู่ใน WM_GRAPHNOTIFY

8. เริ่มแสดงภาพวีดีโอ โดยใช้ฟังก์ชั่น Run() ของ IMediaControl อินเตอร์เฟส

```
HRESULT hr;
hr = pimc->Run();
```

ถ้าคำสั่งเป็นผลสำเร็จ ภาพของวีดีโอจะถูกแสดงบนหน้าจอคอมพิวเตอร์ สำหรับ IMediaControl ยังมีฟังก์ชั่นที่ใช้ในการหยุดภาพวีดีโอชั่วคราว หรือหยุดภาพวีดีโอได้ทุกเวลาตามต้องการ โดยใช้คำสั่งต่อไปนี้

```
hr = pimc->Pause();
hr = pimc->Stop();
```

9. การทำลายพอยท์เตอร์ที่ชี้ไปยังอินเตอร์เฟสต่างๆ

```
pigb->Release();    //ทำโดยระบุชื่อของพอยท์เตอร์ และตามด้วยฟังก์ชั่น Release();
```

จากขั้นตอนทั้ง 8 ขั้นที่ได้แสดงไปจะทำให้ผู้ใช้สามารถสร้างโปรแกรมที่จะทำการแสดงภาพวีดีโอได้อย่างง่ายดาย ต่อไปจะเป็นการเพิ่มเติมการใช้งานอินเตอร์เฟส IMediaSeeking เพื่อให้สามารถควบคุมตำแหน่งการแสดงภาพของวีดีโด ซึ่งจะทำให้ผู้ใช้สามารถเลือกแสดงภาพของวีดีโอได้ตามต้องการ เช่น การเดินหน้า ,การถอยหลัง หรือระบุตำแหน่งที่ต้องการก็ได้

5.6.1 การค้นหา และเลือกตำแหน่งของมัลติมีเดียสตรีมที่ต้องการ

ผู้ใช้สามารถค้นหา และระบุตำแหน่งของมัลติมีเดียสตรีมที่ต้องการให้แสดงผลได้ตามต้องการ ทั้งยังสามารถกำหนดรูปแบบของข้อมูลในการแสดงผลได้อีกด้วย เช่น แสดงตามไบต์ของข้อมูล หรือแสดงตามเวลาของข้อมูล หรือแสดงตามเฟรม (Frame)ของภาพวีดีโอ โดยการเรียกใช้ฟังก์ชันที่อยู่ใน IMediaSeeking อินเตอร์เฟส

การเรียกใช้ IMediaSeeking อินเตอร์เฟส และกำหนดรูปแบบของข้อมูล

```
IMediaSeeking  *pims;
Hr = pigb ->QueryInterface(IID_IMediaSeeking, (void**)&pims);
Hr = pims->SetTimeFormat(&TIME_FORMAT_FRAME);
```

ถ้าคำสั่งเป็นผลสำเร็จรูปแบบของข้อมูลในการกำหนดตำแหน่งของข้อมูล จะเป็นเฟรมของภาพวีดีโอ โดยผู้สามารถเลือกแสดงเฟรมของภาพที่ต้องการได้จากฟังก์ชัน SetPosition() ของ IMediaSeeking อินเตอร์เฟส ดังต่อไปนี้

```
LONGLONG start = 5L;
LONGLONG stop = 15L;
pims->SetPosition(&starts ,AM_SEEKING_AbsolutePositioning ,&stop
,AM_SEEKJNG_AbsolutePositioning);
```

จากตัวอย่างเป็นการกำหนดให้แสดงภาพของวีดีโอเริ่มต้นที่เฟรมที่ 5 และให้หยุดเล่นเมื่อถึงเฟรมที่ 15 สำหรับการกำหนดรูปแบบของการแสดงผลนั้นสามารถกำหนดได้ดังต่อไปนี้

TIME_FORMAT_MEDIA_TIME	กำหนดรูปของข้อมูลให้เป็นเวลาในหน่วย nanosecond
TIME_FORMAT_BYTE	กำหนดรูปของข้อมูลให้เป็นไบต์ในสตรีมของมัลติมีเดีย
TIME_FORMAT_FRAME	กำหนดรูปของข้อมูลให้เป็นเฟรมของภาพวีดีโอ

5.6.2 การกำหนดการแสดงผลของภาพวีดีโอบนวินโดวส์ที่ต้องการ

จากที่ได้กล่าวมาผู้ใช้จะสามารถแสดงภาพของวีดีโอที่ต้องการได้บนหน้าจอคอมพิวเตอร์ โดยเมื่อผู้ใช้เรียกคำสั่ง Run() จาก IMediaControl อินเตอร์เฟส ตัวจัดการฟิลเตอร์กราฟจะทำการสร้างวินโดวส์ขึ้นมาใหม่เพื่อใช้แสดงภาพของวีดีโอโดยเฉพาะโดยผู้ใช้ไม่สามารถกำหนดรูปแบบของวินโดวส์ที่ต้องการได้ ต่อไปนี้จึงเป็นวิธีที่จะทำให้ผู้ใช้สามารถกำหนดการแสดงผลของภาพวีดีโอ ให้ไปแสดงผลบนวินโดวส์ที่มีรูปแบบตามที่ผู้ใช้ได้กำหนดขึ้นมาเองได้ สำหรับการเรียกใช้คำสั่งต่อไปนี้จะต้องเรียกใช้ภายหลังที่ได้เรียกใช้คำสั่ง RenderFile() และก่อนการเรียกใช้คำสั่ง Run() ของ IMediaControl อินเตอร์เฟส

1. กำหนดรูปแบบของวินโดวส์ที่ต้องการแสดงผล

```
pivw->put_WindowStyle(WS_CHILD | WS_CLIPCHILDREN | WS_CLIPSIBLINGS);
```

2. การรับเอาตำแหน่งของวินโดวส์ที่ต้องการแสดงผล

```
HWND ghApp;
RECT grc;
GetClientRect(ghApp, &grc);
```

ตัวแปร ghApp จะใช้เก็บแฮนเดิลวินโดวส์ (Handle window) ที่ต้องการใช้แสดงผลภาพวีดีโอ และตัวแปร grc จะใช้เก็บตำแหน่งของวินโดวส์ที่ต้องการแสดงผลภาพวีดีโอ โดยจะต้องใช้คำสั่ง GetClientRect(); เพื่อรับเอาตำแหน่งของวินโดวส์ ghApp มาใส่ไว้ใน grc

3. กำหนดตำแหน่งที่ต้องการให้แสดงผลโดยใช้ตัวแปร grc เป็นตัวกำหนด

```
pivw->SetWindowPosition(grc.left,grc.top,grc.right,grc.bottom);
```

คำสั่ง SetWindowPosition(); ของ IVideoWindow อินเตอร์เฟสจะทำให้ตำแหน่งการแสดงภาพวีดีโอ ถูกกำหนดตามค่าที่ใส่เข้าไปในฟังก์ชั่น

4. เริ่มต้นแสดงภาพวีดีโอ

```
hr = pimc->Run();
```

ถ้าเป็นคำสั่งเป็นผลสำเร็จ ภาพของวีดีโอที่ได้จะแสดงดังตำแหน่งที่ได้ระบุไว้ข้างต้น

สำหรับตัวอย่างที่ได้กล่าวไป จะเห็นได้ว่าการเขียนโปรแกรมด้วย DirectShow SDK ไม่ใช่เรื่องยากเลย เพียงแต่ต้องทำความเข้าใจกับฟังก์ชันต่างๆที่มีอยู่ในอินเตอร์เฟสแต่ละชนิด เพื่อให้สามารถใช้งานได้อย่างถูกต้อง และมีประสิทธิภาพ

บทที่ 6
โปรแกรมประยุกต์สำหรับเครือข่าย ATM

แอพพลิเคชั่นที่นำมาใช้งานบนระบบเครือข่าย ATM สามารถแบ่งออกได้ 2 ประเภทหลักๆคือ

1. Native ATM application คือ แอพพลิเคชั่นที่ถูกพัฒนาเพื่อนำมาใช้งานในระบบเครือข่าย ATM โดยเฉพาะ สามารถที่จะนำคุณสมบัติที่มีอยู่ในระบบเครือข่าย ATM มาใช้งานได้เช่น การกำหนดคุณภาพของบริการในการส่งข้อมูล,การจองแบนด์วิธสำหรับแต่ละการเชื่อมต่อตามต้องการ (Bandwidth on Demand) ซึ่งความสามารถต่างๆเหล่านี้เหมาะสำหรับแอพพลิเคชั่นที่การหน่วงเวลาของข้อมูลจะมีผลกระทบต่อคุณภาพของข้อมูลเช่น การส่งภาพเคลื่อนไหว,การประชุมทางไกล (Video conference) เป็นต้น

2. แอพพลิเคชั่นที่ทำงานบน LAN Emulation หรือ IP over ATM คือ แอพพลิเคชั่นที่ถูกพัฒนาขึ้นมาเพื่อใช้งานบนระบบเครือข่าย LAN หรือใช้โปรโตคอล TCP/IP ซึ่งถูกจำลองมาใช้งานบนระบบเครือข่าย ATM ดังนั้นก็คือ แอพพลิเคชั่นประเภทที่ใช้งานบนระบบเครือข่าย LAN นั่นเอง

ข้อเปรียบเทียบระหว่างการพัฒนาแอพพลิเคชั่นทั้ง 2 แบบ

1. ความเข้ากันได้ (Compatible) กับระบบเครือข่ายที่ใช้งานกันอยู่ทั่วไป

   Native ATM application นั้นจะไม่สามารถนำมาใช้งานบนระบบเครือข่าย LAN ทั่วไปได้เพราะเป็นแอพพลิเคชั่นที่พัฒนาสำหรับระบบเครือข่าย ATM ซึ่งมีลักษณะการทำงานที่แตกต่างจากระบบเครือข่ายทั่วไปทั้งโครงสร้างของโปรโตคอล (Protocol layer) ,วิธีการส่งข้อมูล, หรือแม้แต่การจัดการแอดเดรส ส่วนแอพพลิเคชั่นที่ทำงานบน LAN Emulation หรือ IP over ATM นั้นสามารถนำไปใช้บนเครือข่ายทั่วไปได้ตามปกติ

2. ประสิทธิภาพในการใช้งานเครือข่าย ATM

   Native ATM application สามารถที่จะใช้งานระบบเครือข่าย ATM ได้อย่างมีประสิทธิภาพมากกว่าคือสามารถติดต่อการระบบเครือข่าย ATM ได้โดยตรง และสามารถใช้งานคุณสมบัติซึ่งเป็นข้อเด่นของระบบเครือข่าย ATM ได้ แต่สำหรับแอพพลิเคชั่นที่ทำงานบน LAN Emulation หรือ IP over ATM นั้นจะต้องมีการแปลงให้อยู่ในรูปที่สามารถทำงานบนเครือข่าย ATM ได้จึงไม่สามารถใช้งานเครือข่ายได้อย่างเต็มประสิทธิภาพ และไม่สามารถใช้คุณสมบัติของเครือข่าย ATM ได้เช่น การกำหนดคุณภาพของบริการในการส่งข้อมูล,การจองแบนด์วิธสำหรับแต่ละการเชื่อมต่อตามต้องการ

3. เครื่องมือที่ใช้ในการพัฒนาแอพพลิเคชั่น

   สำหรับ Native ATM application แล้วเนื่องจากระบบเครือข่าย ATM เป็นระบบที่ค่อนข้างใหม่ ทำให้เครื่องมือที่จะช่วยในการพัฒนาแอพพลิเคชั่นมีน้อยซึ่งมีผลทำให้การพัฒนาแอพพลิเคชั่นนั้นทำได้ยาก แต่สำหรับแอพพลิเคชั่นที่ทำงานบน LAN Emulation หรือ IP over ATM

นั้นการพัฒนาจะทำได้ง่ายกว่าเพราะมีเครื่องมือที่ช่วยในการพัฒนามีอยู่ทั่วไปและมีการพัฒนากันมาอย่างต่อเนื่อง

ในโครงงานวิจัยนี้จะทำการสร้าง**แอพพลิเคชั่นที่ทำงานบนระบบเครือข่าย ATM โดยเฉพาะ (Native ATM Application)**โดยจะสามารถให้บริการส่งไฟล์ได้ทั้งแบบส่งไฟล์ไปทั้งหมดในทีเดียว หรืออาจจะส่งไฟล์วีดีโอสตรีมให้ไคลเอนต์ก็ได้ โดยตัวแอพพลิเคชั่นที่สร้างขึ้นมานี้สามารถแบ่งออกได้เป็น 2 ส่วนหลัก คือ

6.1 ส่วนของการสร้างการติดต่อ

ในส่วนนี้จะจัดการเกี่ยวกับขั้นตอนในการสร้างการติดต่อโดยจะทำงานแบบไคลเอนต์เซิร์ฟเวอร์โมเดลคือจะมีตัวเซิร์ฟเวอร์คอยให้บริการกับตัวไคลเอนต์ที่จะเข้ามาขอใช้บริการ โดยชนิดการติดต่อจะเป็นแบบคอนเน็กชันโอเรียนเต็ท เพราะในเครือข่าย ATM นั้นก่อนที่จะทำการส่งข้อมูลนั้นจะต้องมีการสร้างเส้นทางการติดต่อขึ้นมาก่อน นอกจากนี้ทางฝั่งไคลเอนต์จะสามารถกำหนดคุณภาพของบริการที่ต้องการให้เซิร์ฟเวอร์จัดหาให้ด้วย

สำหรับการเขียนโปรแกรมในส่วนนี้จะใช้ Winsock2.0 เป็นเครื่องมือช่วยในการเขียนโปรแกรม เพราะ Winsock2.0 เป็นแอพพลิเคชั่นโปรแกรมอินเตอร์เฟสที่สนับสนุนการเขียนแอพพลิเคชั่นที่ทำงานบนระบบเครือข่าย ATM โดยเฉพาะ

6.2 ส่วนของการนำเอาสตรีมของวีดีโอไปแสดงเป็นภาพวีดีโอ

ส่วนนี้จะอยู่ทางฝั่งของไคลเอนต์เพียงอย่างเดียว ทางฝั่งเซิร์ฟเวอร์จะไม่มีเพราะเซิร์ฟเวอร์จะทำหน้าที่ให้บริการเกี่ยวกับไฟล์เพียงอย่างเดียว โดยในส่วนนี้จะรับเอาสตรีมของวีดีโอที่ได้รับจากเซิร์ฟเวอร์ไปแสดงผลได้แบบเรียลไทม์ และผู้ใช้สามารถหยุดภาพ หรือ เล่นภาพวีดีโอได้ตามต้องการ

สำหรับการเขียนโปรแกรมในส่วนนี้จะใช้ ชุดพัฒนาโปรแกรม DirectShow SDK เป็นเครื่องมือช่วยในการเขียนโปรแกรม เพราะใน DirectShow จะมีเครื่องมือช่วยในสร้างส่วนรับ และประมวลผลสตรีมของวีดีโอ ทำให้การพัฒนาโปรแกรมในส่วนนี้ทำได้สะดวกรวดเร็วและมีประสิทธิภาพมากขึ้น

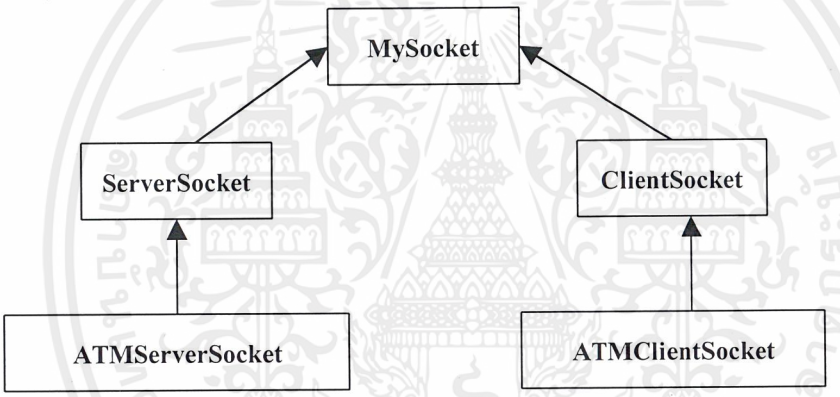
บทที่ 7
แอพพลิเคชั่นในส่วนของการสร้างการติดต่อ

ในการเขียนโปรแกรมบน WinSock ที่จะทำงานบนระบบเครือข่ายATM นั้นจะต้องมีการรวม (include) เฮดเดอร์ไฟล์ดังนี้คือ

1. WinSock2.h
2. ws2atm.h

และต้องมีการเพิ่มลิงก์ไลบรารีที่ชื่อ ws2_32.lib เพื่อให้สามารถเรียกฟังก์ชันของ WinSock2.0 ได้

7.1 คลาส(Class)ที่สำคัญในแอพพลิเคชั่นในส่วนของการสร้างการติดต่อ



รูปที่ 7.1 แสดงการความสัมพันธ์ของคลาสที่สำคัญในแอพพลิเคชั่นในส่วนของการสร้างการติดต่อ

แอพพลิเคชั่นในส่วนของการสร้างการติดต่อจะมีคลาสที่สำคัญดังนี้

1. **คลาส MySocket** จะเป็นคลาสฐานที่รวมคุณสมบัติและฟังก์ชันพื้นฐานในการจัดการกับซอกเก็ตไว้

   คุณสมบัติ(Property)ที่สำคัญในคลาส MySocket

   1. AddressFamily เป็นคุณสมบัติที่ใช้กำหนดชนิดของประเภทของแอดเดรส (Address Family) ที่ใช้ ซึ่งถ้าต้องการสร้างแอพพลิเคชั่นที่ทำงานบนระบบเครือข่าย ATM คุณสมบัตินี้ก็จะมีค่าเป็น AF_ATM
   2. SocketType เป็นคุณสมบัติที่กำหนดชนิดของซอกเก็ตซึ่งถ้าต้องการสร้างแอพพลิเคชั่นที่ทำงานบนระบบเครือข่าย ATM คุณสมบัตินี้ก็จะมีค่าเป็น SOCK_RAW ซึ่งทำให้สามารถเข้าถึงโปรโตคอลในระดับต่ำได้

3. ProtocolType เป็นคุณสมบัติที่ใช้กำหนดโปรโตคอลที่ใช้ในการติดต่อสื่อสาร ซึ่งถ้าต้องการสร้างแอพพลิเคชั่นที่ทำงานบนระบบเครือข่าย ATM คุณสมบัตินี้ก็จะมีค่าเป็น ATMPROTO_AAL5 หรือ ATMPROTO_AALUSER
4. s เป็นคุณสมบัติที่ใช้เป็นซอกเก็ตที่ใช้ในการรับส่งข้อมูล
5. ReceiveBuffer เป็นคุณสมบัติที่ใช้เป็นบัฟเฟอร์สำหรับเก็บข้อมูลที่ได้รับมา
6. SendBuffer เป็นคุณสมบัติที่ใช้เป็นบัฟเฟอร์สำหรับเก็บข้อมูลที่จะส่ง

เมธอด(Method) ที่สำคัญในคลาส MySocket

1. CreateSocket() ทำหน้าที่ในการสร้างซอกเก็ต
2. CloseSocket() ทำหน้าที่ในการปิดซอกเก็ต
3. SendData() เป็นเมธอดที่ใช้ในการส่งข้อมูลที่เก็บอยู่ในคุณสมบัติ SendBuffer
4. ReceiveData()เป็นเมธอดที่ใช้ในการรับข้อมูลมาเก็บอยู่ในคุณสมบัติ ReceiveBuffer

2. คลาส ServerSocket จะเป็นคลาสที่สืบทอดมาจากคลาส MySocket อีกทีจะมีคุณสมบัติสำคัญที่ออปเจกต์ที่จะเป็นเซิร์ฟเวอร์ควรต้องมี การที่ต้องมีคลาส ServerSocket ก็เพื่อที่จะได้สามารถสร้างเซิร์ฟเวอร์ชนิดอื่นที่ไม่ใช่ ATM เซิร์ฟเวอร์ได้โดยง่าย

คุณสมบัติที่สำคัญในคลาส ServerSocket

1. MaximumConnection เป็นคุณสมบัติที่ใช้กำหนดจำนวนของไคลเอนต์สูงสุดที่เซิร์ฟเวอร์ให้บริการได้
2. ListenQueue เป็นคุณสมบัติที่ใช้กำหนดจำนวนของไคลเอนต์ที่สามารถรอขอการติดต่อเซิร์ฟเวอร์ได้

3. คลาส ClientSocket จะเป็นคลาสที่สืบทอดมาจากคลาส MySocket อีกทีจะมีคุณสมบัติสำคัญที่ออปเจกต์ที่จะเป็นไคลเอนต์ควรต้องมี การที่ต้องมีคลาส ClientSocket ก็เพื่อที่จะได้สามารถสร้างไคลเอนต์ชนิดอื่นที่ไม่ใช่ ATM ไคลเอนต์ได้โดยง่าย

คุณสมบัติที่สำคัญในคลาส ClientSocket

1. SocketQOS เป็นคุณสมบัติที่ใช้ในการกำหนดถึงคุณภาพของบริการที่ไคลเอนต์ต้องการให้เซิร์ฟเวอร์จัดสรรให้โดยคุณสมบัตินี้จะมีชนิดเป็น LPQOS คือเป็นตัวชี้(Pointer) ไปยังชนิดข้อมูลชนิด QOS

ชนิดข้อมูล QOS

```
typedef struct _QualityOfService{
        FLOWSPEC     SendingFlowspec;     /* the flow spec for data sending */
        FLOWSPEC     ReceivingFlowspec;   /* the flow spec for data receiving */
        WSABUF       ProviderSpecific;    /* additional provider specific stuff */
} QOS, FAR * LPQOS;
```

ซึ่งในการเขียนโปรแกรมบน WinSock ที่ทำงานบนระบบเครือข่าย ATM นั้นข้อมูลเกี่ยวกับรายละเอียดของคุณภาพของบริการนั้นจะถูกเก็บอยู่ในส่วนของฟิลด์(field) ProviderSpecific โดยส่วนของฟิลด์ SendingFlowspec และ ReceiveFlowspec จะไม่ถูกใช้ โดยรายละเอียดของคุณภาพของบริการนั้นจะมีชนิดของข้อมูลที่สำคัญที่ต้องใช้ดังนี้

ชนิดข้อมูล Q2931_IE

```
typedef struct {
        Q2931_IE_TYPE IEType;
        ULONG         IELength;
        UCHAR         IE[1];
} Q2931_IE;
```

ชนิดข้อมูลนี้เป็นตัวระบุถึงชนิดของรายละเอียดของคุณภาพของบริการ (Information Element) ที่อยู่ในฟิลด์ ProviderSpecific เพราะในฟิลด์นี้จะเก็บชนิดของรายละเอียดของคุณภาพของบริการไว้หลายตัวซึ่งมีชนิดของข้อมูลหลายชนิดดังนี้

ชนิดข้อมูล AAL_PARAMETERS_IE

```
typedef struct {
    AAL_TYPE AALType;
    union {
            AAL5_PARAMETERS    AAL5Parameters;
            AALUSER_PARAMETERS  AALUserParameters;
          } AALSpecificParameters;
} AAL_PARAMETERS_IE;
```

ชนิดข้อมูล AAL_PARAMETERS_IE นี้จะใช้บอกถึงรายละเอียดเกี่ยวกับโปรโตคอลที่ต้องการใช้โดยใน WinSock2.0 รุ่นปัจจุบันนี้จะสนับสนุนเพียง 2 ชนิดคือAAL5 และ AALที่ผู้ใช้กำหนดเอง

ชนิดข้อมูล ATM_TRAFFIC_DESCRIPTOR_IE

```
typedef struct {
    ATM_TD Forward;
    ATM_TD Backward;
```

```
        BOOL   BestEffort;
    } ATM_TRAFFIC_DESCRIPTOR_IE;
```

ชนิดข้อมูลนี้จะใช้ระบุถึงค่า Peak Cell Rate(PCR) ,ค่า Sustainable Cell Rate (SCR) ,ค่า Maximum Burst Size (MBS) ซึ่งเป็นค่าพารามิเตอร์(Parameter) ที่สำคัญในการเซตค่าคุณภาพของบริการในระบบเครือข่าย ATM

ชนิดข้อมูล ATM_BROADBAND_BEARER_CAPABILITY_IE

```
typedef struct {
    UCHAR BearerClass;
    UCHAR TrafficType;
    UCHAR TimingRequirements;
    UCHAR ClippingSusceptability;
    UCHAR UserPlaneConnectionConfig;
} ATM_BROADBAND_BEARER_CAPABILITY_IE;
```

ชนิดข้อมูลนี้จะใช้ระบุบริการของการติดต่อที่จะสร้างเช่นระบุว่าต้องการชนิดของคุณภาพของบริการแบบไหน Constant Bit Rate (CBR) หรือ Variable Bit Rate (VBR) เป็นต้น

ชนิดข้อมูล ATM_QOS_CLASS_IE

```
typedef struct {
    UCHAR QOSClassForward;
    UCHAR QOSClassBackward;
} ATM_QOS_CLASS_IE;
```

ชนิดข้อมูลนี้จะใช้ระบุคลาส(class) ของบริการที่ต้องการ

4. คลาส ATMServerSocket จะเป็นคลาสที่สืบทอดมาจากคลาส ServerSocket อีกทีหนึ่ง จะเป็นคลาสที่ใช้สร้างออปเจกต์ที่จะเป็น ATM เซิร์ฟเวอร์ในแอพพลิเคชั่นในส่วนการสร้างการติดต่อนี้

คุณสมบัติที่สำคัญในคลาส ATMServerSocket

1. saATM เป็นคุณสมบัติที่มีชนิดเป็น sockaddr_atm ซึ่งใช้สำหรับเก็บข้อมูลเกี่ยวกับเซิร์ฟเวอร์ เช่น ประเภทของแอดเดรส,ATM แอดเดรสของเซิร์ฟเวอร์

ชนิดข้อมูล sockaddr_atm

```
typedef struct sockaddr_atm {
    u_short satm_family;          /* address family should be AF_ATM  */
    ATM_ADDRESS satm_number;      /* ATM address                      */
    ATM_BLLI satm_blli;           /* B-LLI                            */
    ATM_BHLI satm_bhli;           /* B-HLI                            */
}sockaddr_atm,SOCKADDR_ATM,*PSOCKADDR_ATM,*LPSOCKADDR_ATM;
```

2. ATMSEL เป็นคุณสมบัติที่ใช้กำหนดค่าไบต์สุดท้ายในATM แอดเดรส
3. ATMClient[16] เป็นคุณสมบัติที่ใช้สร้างออปเจกต์ที่จะทำหน้าที่ติดต่อกับไคลเอนต์โดยคุณสมบัตินี้จะมีชนิดของข้อมูลเป็นคลาส MySocket

เมธอดที่สำคัญในคลาส ATMServerSocket

1. ATMBind() เป็นเมธอดที่ใช้ในการตั้งชื่อให้กับซอกเก็ตเพื่อให้ไคลเอนต์สามารถค้นหาซอกเก็ตของเซิร์ฟเวอร์ได้
2. ATMListen() เป็นเมธอดที่ใช้ในการสั่งให้ซอกเก็ตรอรับการติดต่อที่ไคลเอนต์ขอสร้างการติดต่อมา และกำหนดจำนวนไคลเอนต์ที่สามารถรอการตอบรับในคิว(queue)ได้ตามจำนวนที่กำหนดตาม ListenQueue
3. ATMAccept() เป็นเมธอดที่ใช้ในการตอบรับการติดต่อจากไคลเอนต์เมื่อไคลเอนต์ทำการขอสร้างการติดต่อเข้ามา เมธอดจะคืนค่าเป็นซอกเก็ตที่เซิร์ฟเวอร์จะใช้ติดต่อกับไคลเอนต์ที่ทำการขอสร้างการติดต่อนั้น
4. ATMFindAddress() เป็นเมธอดที่ใช้ในการหาATM แอดเดรสของเซิร์ฟเวอร์เองเพื่อใช้ในการตั้งชื่อให้ซอกเก็ต
5. GetATMConnectionID() เป็นเมธอดที่ใช้ในการหาค่า VPI และ VCI ของวงจรเสมือนที่สร้างขึ้น
6. ATMCreateClientThread() เป็นเมธอดที่ใช้ในการสร้างเทรดที่ใช้ในการจัดการการติดต่อกับไคลเอนต์ที่ขอสร้างการติดต่อมา

5. คลาส ATMClientSocket จะเป็นคลาสที่สืบทอดมาจากคลาส ClientSocket อีกทีหนึ่งจะเป็นคลาสที่ใช้สร้างออปเจกต์ที่จะเป็น ATM ไคลเอนต์ในแอพพลิเคชั่นส่วนการสร้างการติดต่อนี้

คุณสมบัติที่สำคัญของคลาส ATMClientSocket

1. saATM เป็นคุณสมบัติที่ชนิดเป็น sockaddr_atm ซึ่งใช้สำหรับเก็บข้อมูลเกี่ยวกับเซิร์ฟเวอร์ เช่น ประเภทของแอดเดรส,ATM แอดเดรสของเซิร์ฟเวอร์ ซึ่งต้องใช้ในการขอสร้างการติดต่อกับเซิร์ฟเวอร์
2. ATMSEL เป็นคุณสมบัติที่ใช้กำหนดค่าไบต์สุดท้ายในATM แอดเดรสซึ่งต้องกำหนดให้ถูกต้องตรงกับเซิร์ฟเวอร์เพื่อจะได้ขอสร้างการติดต่อได้ถูกต้อง

เมธอดที่สำคัญของคลาส ATMClientSocket

1. GetATMAddress() เป็นเมธอดที่จะทำหน้าที่ในการแมป (map) ชื่อของเซิร์ฟเวอร์ที่ไคลเอนต์ต้องการขอสร้างการติดต่อกับไฟล์ ATMHOST ซึ่งเก็บตารางการแมปชื่อโฮสต์และATM แอดเดรสของโฮสต์ไว้ ให้เป็นATM แอดเดรส เพื่อที่จะนำไปเก็บไว้ในคุณสมบัติ saATM เพื่อใช้ในการขอสร้างการติดต่อ

2. ATMConnect() เป็นเมธอดที่ทำหน้าที่ขอสร้างการติดต่อกับเซิร์ฟเวอร์โดยก่อนที่จะเรียกใช้ฟังก์ชันนี้จะต้องมีการระบุคุณสมบัติ SockQOS ให้เรียบร้อยก่อนเพื่อระบุคุณภาพของบริการที่ต้องการ
3. GetATMConnectionID() เป็นเมธอดที่ใช้ในการหาค่า VPI และ VCI ของวงจรเสมือนที่สร้างขึ้น

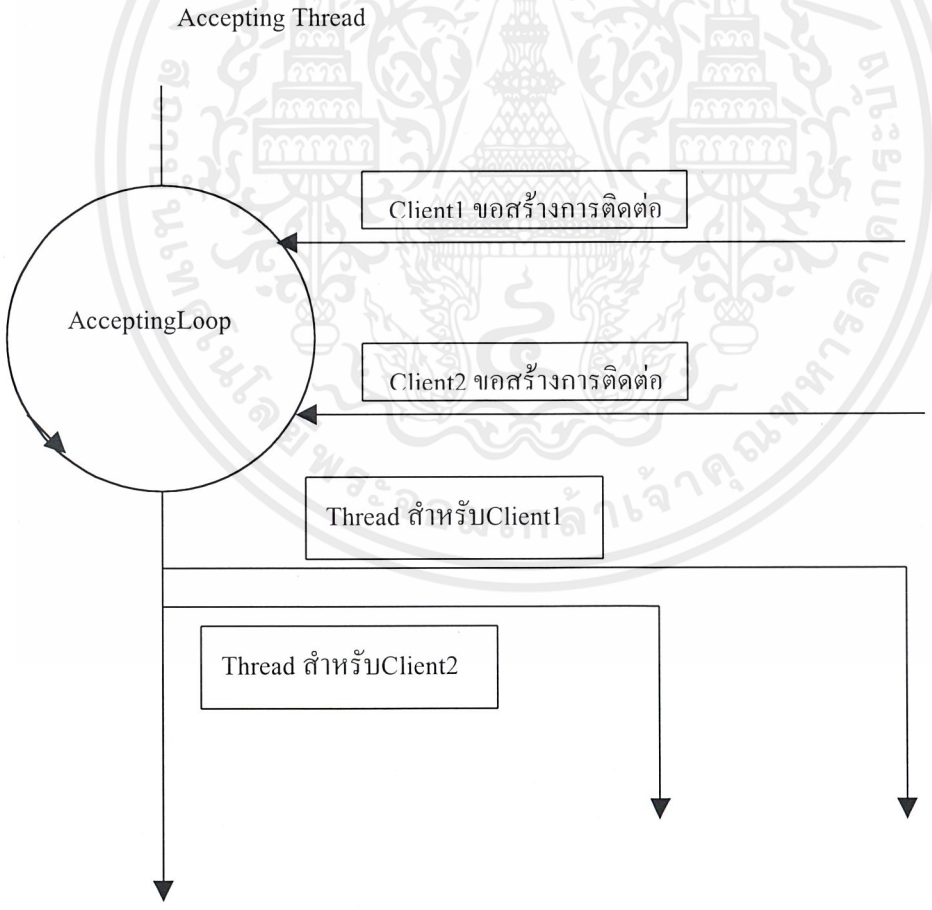
7.2 ส่วนประกอบแอพพลิเคชั่นในส่วนของการสร้างการติดต่อ

จะแบ่งเป็น 2 ส่วนหลักคือ

1. เซิร์ฟเวอร์
2. ไคลเอนต์

7.2.1 เซิร์ฟเวอร์

เป็นส่วนที่ทำหน้าในการให้บริการส่งไฟล์ข้อมูลให้กับไคลเอนต์ที่ขอบริการมาโดยตัวเซิร์ฟเวอร์จะมีการทำงานเป็นมัลติเทรดเซิร์ฟเวอร์ดังรูป



รูปที่ 7.2 แสดงการทำงานของมัลติเทรดเซิร์ฟเวอร์

จากรูปที่ 7.2 เมื่อเซิร์ฟเวอร์เริ่มทำงานจะทำการสร้างเทรดที่ทำหน้าที่ในการวนลูป (loop) รอรับการติดต่อจากตัวไคลเอนต์ เมื่อมีไคลเอนต์ทำการขอติดต่อเข้ามา เซิร์ฟเวอร์จะทำการสร้างเทรดใหม่ขึ้นมาเพื่อทำหน้าที่ในการจัดการการบริการตามที่ตัวไคลเอนต์นั้นขอมา ทำให้สามารถใช้การสร้างซอกเก็ตแบบบล็อกให้ทำงานกับแต่ละไคลเอนต์ที่ขอติดต่อเข้ามาได้โดยปล่อยให้ระบบปฏิบัติการช่วยจัดการในการแบ่งเวลาในการให้บริการกับแต่ละเทรดเอง ทำให้การเขียนโปรแกรมในการจัดการกับไคลเอนต์แต่ละตัวทำได้ง่ายขึ้น โดยตัวเซิร์ฟเวอร์สามารถให้บริการกับไคลเอนต์ได้ 2 ลักษณะคือ

1. ให้บริการดาวน์โหลดไฟล์ไปทั้งไฟล์

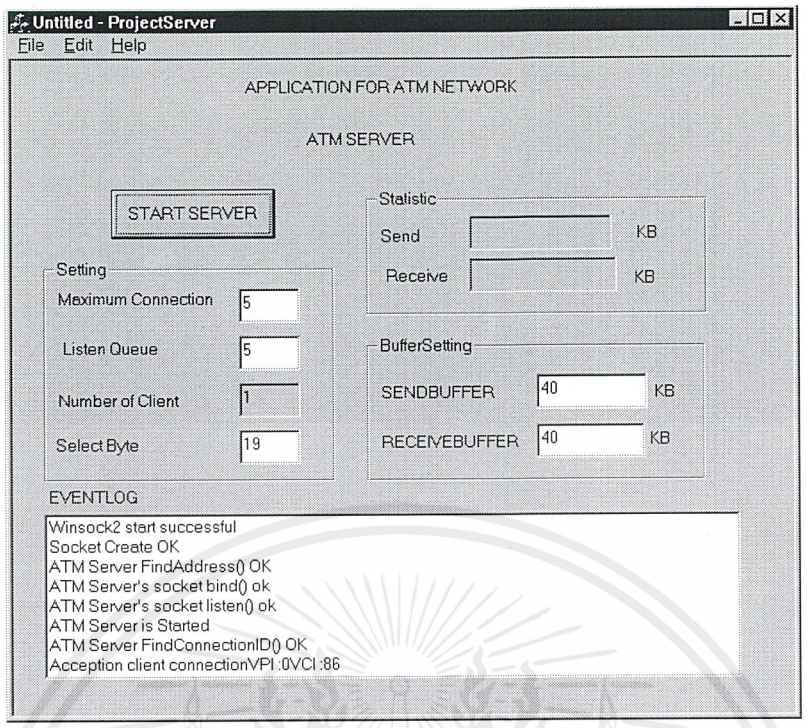
รูปที่ 7.3 แสดงการให้บริการดาวน์โหลดไฟล์ไปทั้งไฟล์

การให้บริการลักษณะนี้ไคลเอนต์จะต้องรอรับไฟล์ทั้งหมดก่อนแล้วจึงนำไปใช้งานได้

2. ให้บริการส่งไฟล์วีดีโอสตรีม

รูปที่ 7.4 แสดงการให้บริการแบบส่งไฟล์วีดีโอสตรีม

การให้บริการในลักษณะนี้ไคลเอนต์สามารถใช้งานข้อมูลได้ทันทีโดยไม่ต้องรอให้รับไฟล์มาทั้งหมดก่อน

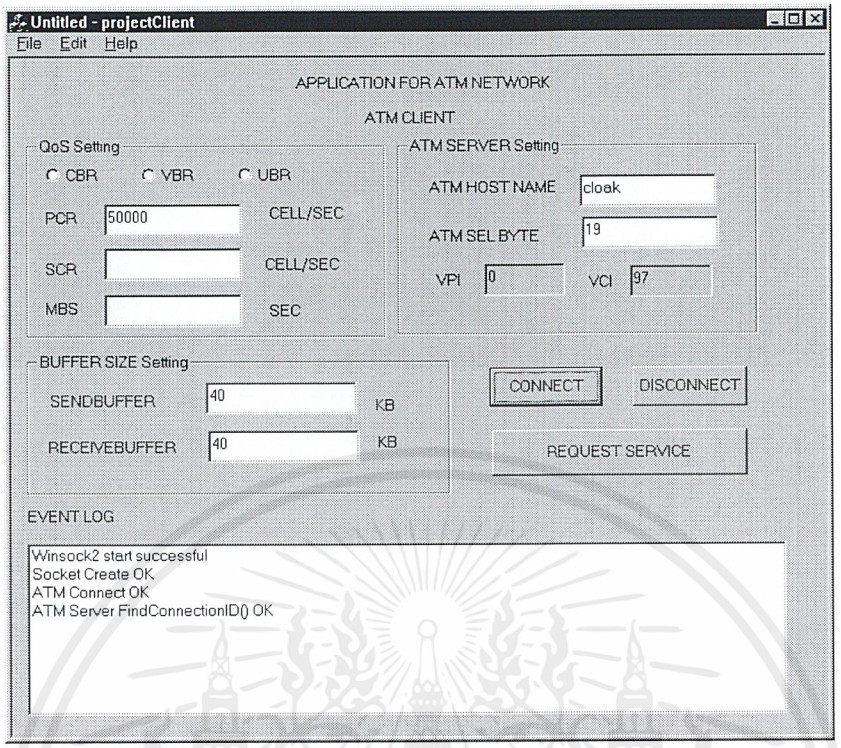


รูปที่ 7.5 แสดงหน้าจอของโปรแกรมเซิร์ฟเวอร์

7.2.2 ไคลเอนต์

ส่วนไคลเอนต์นี้จะสามารถขอสร้างการติดต่อเพื่อขอใช้บริการจากเซิร์ฟเวอร์ได้ โดยสามารถระบุคุณภาพของบริการที่ต้องการได้ โดยจะต้องไม่เกินความสามารถของตัวเซิร์ฟเวอร์ที่จะให้บริการได้ โดยมิฉะนั้นแล้วการขอสร้างการติดต่อนั้นจะถูกปฏิเสธ เมื่อไคลเอนต์สามารถสร้างการติดต่อกับเซิร์ฟเวอร์ได้สำเร็จ ตัวไคลเอนต์ก็จะทำการสร้างเทรดขึ้นมาใหม่หนึ่งเทรดเพื่อใช้ในการรับส่งข้อมูลกับเซิร์ฟเวอร์

การทำงานของไคลเอนต์นั้นจะสามารถขอบริการได้ 2 แบบตามที่เซิร์ฟเวอร์ให้บริการได้และตัวไคลเอนต์จะมีไฟล์อยู่ไฟล์หนึ่งชื่อ ATMHOST จะเก็บตารางที่เก็บชื่อโฮสต์และATM แอดเดรสของโฮสต์นั้นไว้ก่อน เพื่อให้ไคลเอนต์สามารถทราบATM แอดเดรสของเซิร์ฟเวอร์ที่ต้องการจะติดต่อได้โดยจะดูจากไฟล์ ATMHOST นี้



รูปที่ 7.6 แสดงหน้าจอของโปรแกรมไคลเอนต์

จากรูปที่ 7.6 จะเห็นว่าผู้ใช้ทางฝั่งไคลเอนต์สามารถที่จะเลือกคุณภาพของการบริการได้ 3 แบบ คือ

1. Constant Bit Rate (CBR) ใช้สำหรับแอพพลิเคชั่นที่ต้องการแบนด์วิธในการส่งข้อมูลคงที่ คือในการส่งข้อมูลจะส่งด้วยความเร็วใดความเร็วหนึ่งคงที่

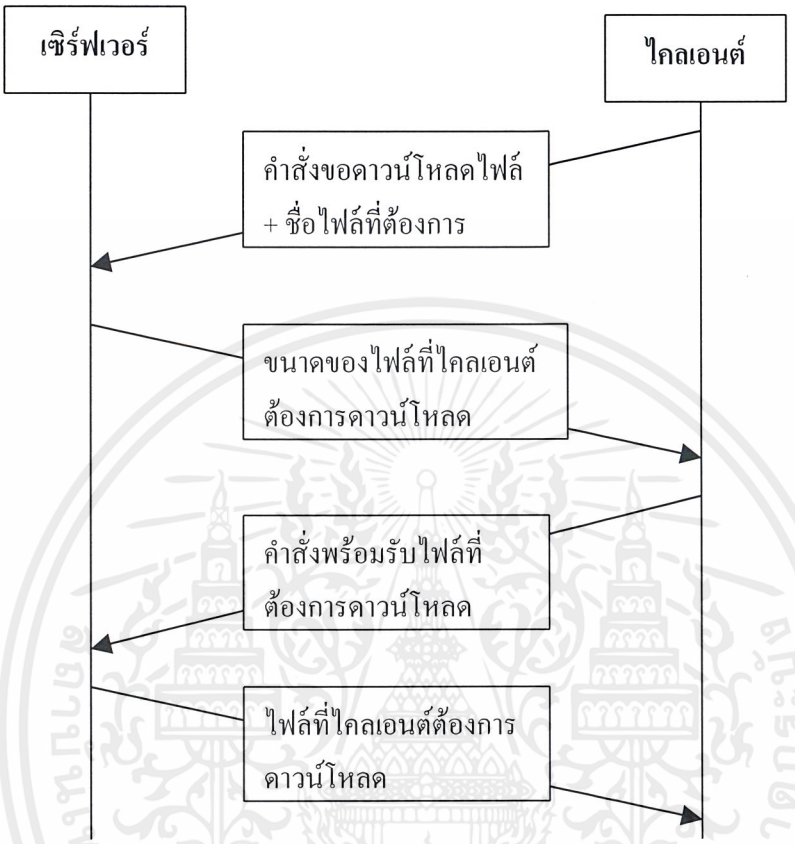
2. Variable Bit Rate (VBR) ใช้สำหรับแอพพลิเคชั่นที่ใช้แบนด์วิธไม่คงที่คือจะส่งข้อมูลด้วยความเร็วหนึ่งแล้วอาจจะเปลี่ยนไปส่งข้อมูลอีกความเร็วหนึ่งก็ได้ โดยค่าหน่วงเวลาจะมีผลต่อคุณภาพของข้อมูลที่ส่ง

3. Unspecified Bit Rate (UBR) ใช้สำหรับแอพพลิเคชั่นที่ไม่ต้องการคุณภาพของบริการในการส่งข้อมูลเช่นการดาวน์โหลดไฟล์ข้อมูล

7.2.3 การส่งเมสเสจกันระหว่างไคลเอนต์และเซิร์ฟเวอร์

ในการติดต่อขอบริการของไคลเอนต์ไปยังเซิร์ฟเวอร์นั้น จะต้องมีการส่งเมสเสจกันระหว่างไคลเอนต์และเซิร์ฟเวอร์ดังนี้

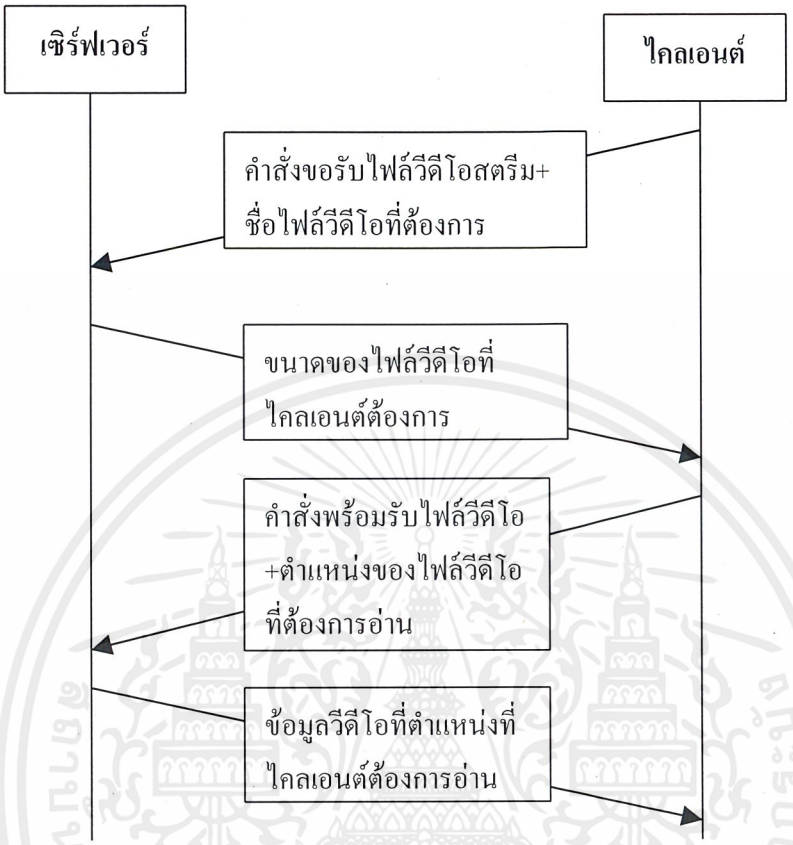
1. บริการดาวน์โหลดไฟล์ทั้งไฟล์



รูปที่ 7.7 แสดงการส่งเมสเสจของการบริการแบบดาวน์โหลดไฟล์ทั้งไฟล์

จากรูปที่ 7.7 จะเห็นว่าในการส่งข้อมูลระหว่างไคลเอนต์และเซิร์ฟเวอร์สำหรับบริการแบบดาวน์โหลดไฟล์ไปทั้งไฟล์นั้น จะเริ่มจากไคลเอนต์จะทำการส่งเมสเสจไปบอกเซิร์ฟเวอร์ว่าต้องการดาวน์โหลดไฟล์โดยจะส่งชื่อไฟล์ที่ต้องการดาวน์โหลดไปให้เซิร์ฟเวอร์ด้วย เมื่อเซิร์ฟเวอร์ได้รับเมสเสจของไคลเอนต์แล้วก็จะทำการส่งขนาดของไฟล์ที่ไคลเอนต์ต้องการดาวน์โหลดไปให้ไคลเอนต์ จากนั้นเมื่อไคลเอนต์ได้รับขนาดของไฟล์ที่ต้องการแล้ว ก็จะทำการส่งเมสเสจไปบอกเซิร์ฟเวอร์ว่าพร้อมที่จะรับไฟล์ข้อมูลแล้ว เมื่อเซิร์ฟเวอร์ได้รับเมสเสจจากไคลเอนต์ซึ่งแสดงว่าไคลเอนต์พร้อมที่จะรับข้อมูลแล้ว เซิร์ฟเวอร์ก็จะทำการส่งไฟล์ข้อมูลไปให้ไคลเอนต์จนกว่าจะครบขนาดไฟล์ ฝ่ายไคลเอนต์ก็จะคอยรับไฟล์ข้อมูลที่เซิร์ฟเวอร์ส่งไปให้จนกว่าจะครบ

2. บริการส่งไฟล์วีดีโอสตรีม



รูปที่ 7.8 แสดงการส่งเมสเสจของการบริการส่งไฟล์วีดีโอสตรีม

จากรูปที่ 7.8 จะเห็นว่าการส่งเมสเสจระหว่างไคลเอนต์และเซิร์ฟเวอร์สำหรับบริการส่งไฟล์วีดีโอสตรีมนั้นจะเริ่มจากไคลเอนต์จะส่งเมสเสจไปยังเซิร์ฟเวอร์เพื่อบอกว่าต้องการรับไฟล์วีดีโอสตรีมโดยจะทำการส่งชื่อของไฟล์วีดีโอที่ต้องการไปให้เซิร์ฟเวอร์ด้วย เมื่อเซิร์ฟเวอร์ได้รับคำร้องขอไฟล์วีดีโอแล้ว ก็จะทำการส่งขนาดของไฟล์วีดีโอที่ไคลเอนต์ต้องการไปให้ไคลเอนต์ จากนั้นเมื่อไคลเอนต์ได้รับขนาดของไฟล์วีดีโอแล้ว ก็จะส่งเมสเสจไปบอกเซิร์ฟเวอร์ว่าพร้อมที่จะรับไฟล์วีดีโอแล้วโดยจะระบุตำแหน่งของไฟล์วีดีโอที่ต้องการไปให้เซิร์ฟเวอร์ด้วย เมื่อเซิร์ฟเวอร์ได้รับเมสเสจจากไคลเอนต์ซึ่งแสดงว่าไคลเอนต์พร้อมรับไฟล์วีดีโอแล้วก็จะส่งไฟล์วีดีโอ ณ ตำแหน่งที่ไคลเอนต์ต้องการไปให้ไคลเอนต์ จากการที่มีการส่งตำแหน่งของไฟล์วีดีโอที่ต้องการใช้นี้จะทำให้ผู้ใช้ทางฝั่งไคลเอนต์สามารถดูไฟล์วีดีโอ ณ ตำแหน่งใดก็ได้ตามต้องการได้แบบเรียลไทม์

บทที่ 8
แอพพลิเคชั่นในส่วนรับสตรีม และแสดงภาพวีดีโอ

ในบทนี้จะกล่าวถึงส่วนของโปรแกรมในโครงงานนี้ ซึ่งจะทำหน้าที่ในการรับสตรีมของวีดีโอ MPEG-1 จากเครื่องเซิร์ฟเวอร์และทำการแสดงผลเป็นภาพวีดีโอบนเหน้าจอของเครื่องไคลเอนต์ โดยผู้ใช้สามารถควบคุมการแสดงภาพวีดีโอได้ตามต้องการ เช่น การเล่นภาพวีดีโอ หรือ หยุดภาพวีดีโอ หรือ เลือกตำแหน่งของภาพวีดีโอที่ต้องการ

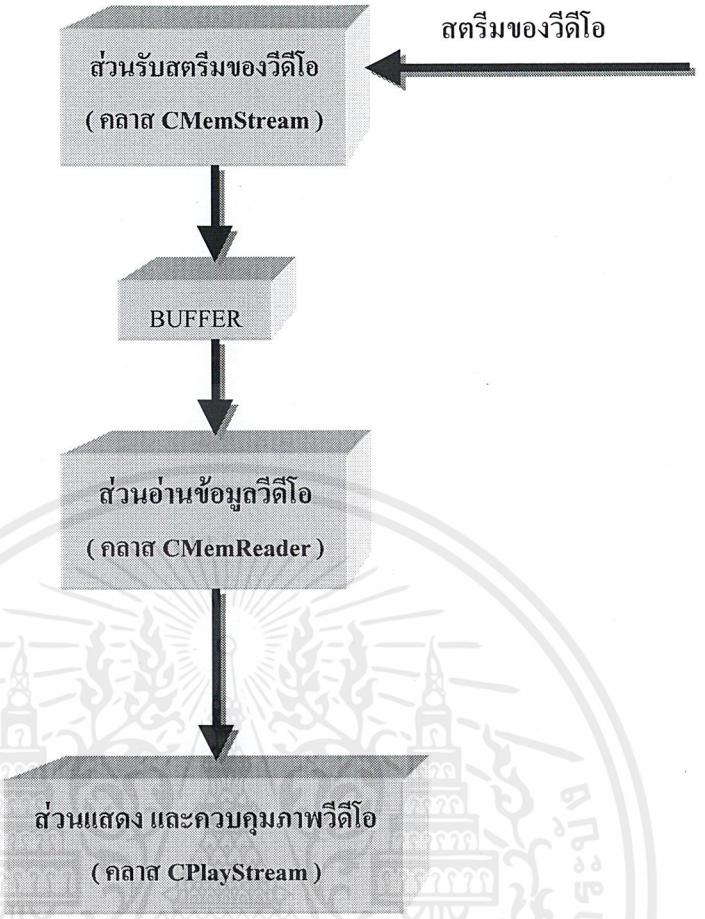
8.1 หลักการทำงานของโปรแกรม

โปรแกรมรับสตรีม และแสดงภาพวีดีโอ MPEG-1 ที่ได้สร้างขึ้นในโครงงานครั้งนี้จะแบ่งส่วนของการทำงานในแต่ละหน้าที่ออกเป็น 3 ส่วนดังต่อไปนี้

1. **ส่วนรับสตรีมของวีดีโอ**
   ทำหน้าที่ในการส่งตำแหน่งของไฟล์วีดีโอที่ต้องการอ่านไปให้เซิร์ฟเวอร์ผ่านทางซอกเก็ตที่ได้สร้างขึ้นติดต่อกับเซิร์ฟเวอร์ จากนั้นจึงรอรับสตรีมของวีดีโอที่ส่งมาจากเซิร์ฟเวอร์และนำไปเก็บไว้ในบัฟเฟอร์เพื่อรอการอ่านไปแสดงเป็นภาพวีดีโอ สำหรับส่วนรับสตรีมของวีดีโอนี้จะทำงานโดยคลาส CMemStream

2. **ส่วนอ่านข้อมูลวีดีโอ**
   ทำหน้าที่ในการอ่านข้อมูลที่ถูกเก็บไว้ในบัฟเฟอร์ และจัดส่งต่อไปให้ส่วนที่ทำหน้าที่ในการแสดงภาพวีดีโอ โดยส่วนนี้จะทำหน้าที่เป็นซอร์สฟิลเตอร์สำหรับฟิลเตอร์กราฟ และสำหรับส่วนอ่านข้อมูลวีดีโอนี้จะทำงานโดยคลาส CMemReader

3. **ส่วนแสดง และควบคุมภาพวีดีโอ**
   ทำหน้าที่ในการแสดงภาพวีดีโอที่ได้ออกทางหน้าจอคอมพิวเตอร์ และควบคุมการแสดงผลด้วย โดยผู้ใช้สามารถควบคุมการแสดงผลของภาพวีดีโอ เช่น การเล่นภาพ หรือการหยุดภาพได้โดยผ่านปุ่มควบคุมต่างๆ โดยส่วนนี้จะทำหน้าที่เป็นทรานฟอร์มฟิลเตอร์ และเรนเดอร์ฟิลเตอร์สำหรับฟิลเตอร์กราฟ และส่วนแสดง และควบคุมภาพวีดีโอนี้จะทำงานโดยคลาส CPlayStream



รูปที่ 8.1 แสดงการทำงาน และเชื่อมต่อกันของแต่ส่วนของโปรแกรม

8.1.1 การเก็บสตรีมของวีดีโอลงบัฟเฟอร์

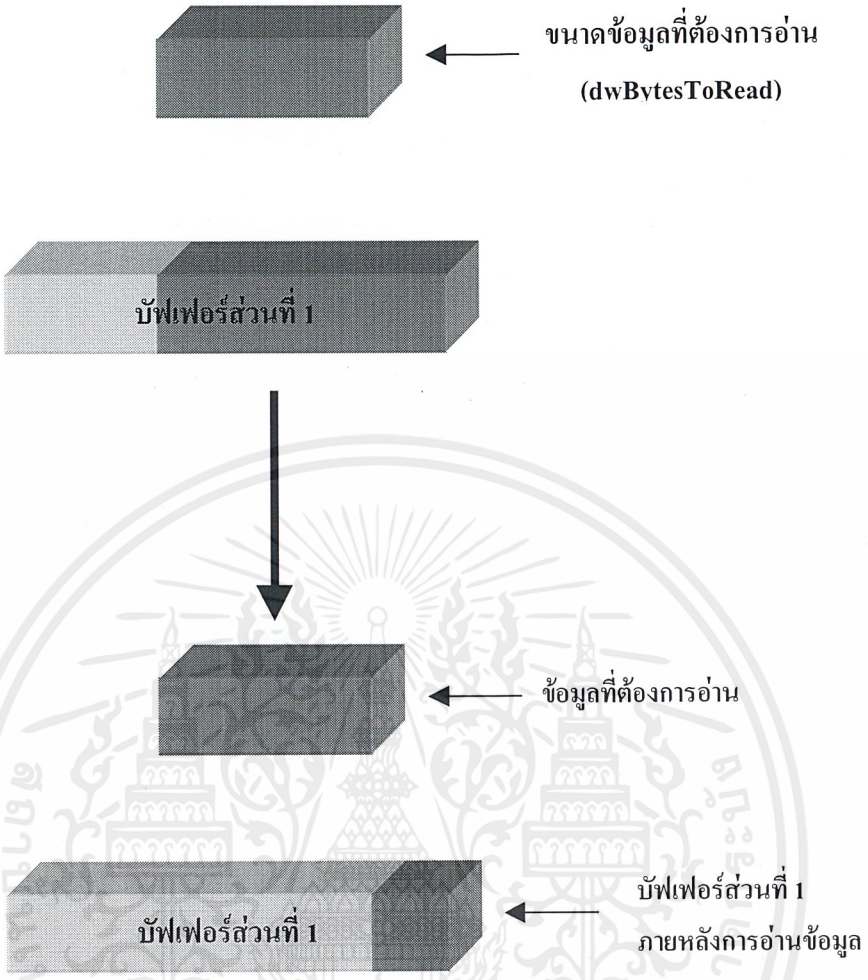
บัฟเฟอร์ที่ใช้ในการเก็บสตรีมของวีดีโอมี 2 ส่วน ดังต่อไปนี้

- บัฟเฟอร์ส่วนที่ 1 (ตัวแปรชื่อ m_pbData)

เป็นบัฟเฟอร์หลักที่ใช้เก็บสตรีมของวีดีโอที่จะถูกอ่านจากส่วนอ่านข้อมูลวีดีโอเพื่อนำไปประมวลผล และส่งต่อไปเพื่อแสดงผลเป็นภาพวีดีโอ

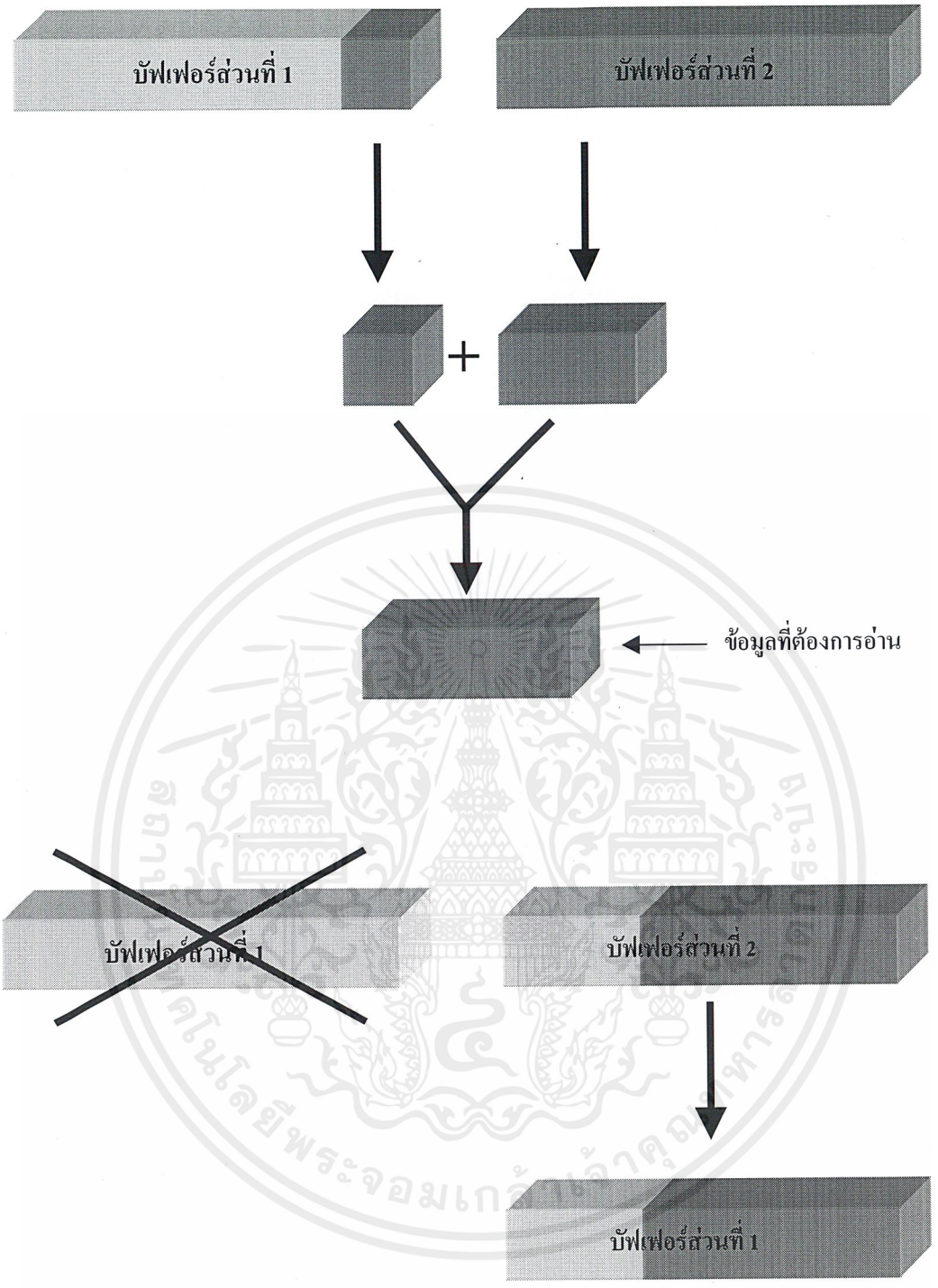
- บัฟเฟอร์ส่วนที่ 2 (ตัวแปรชื่อ m_pbDataBuffer)

เป็นบัฟเฟอร์สำรอง ที่ใช้เก็บสตรีมของวีดีโอ โดยจะถูกใช้งานเมื่อข้อมูลสตรีมของวีดีโอในบัฟเฟอร์ส่วนที่ 1 (m_pbData) ถูกอ่านไปจนหมดแล้ว จึงจะนำเอาบัฟเฟอร์ส่วนที่ 2 มาใช้งานแทน โดยการย้ายพอยท์เตอร์ที่ชี้ไปยัง m_pbData มาชี้ที่ m_pbDataBuffer แทน



รูปที่ 8.2 แสดงกระบวนการอ่านข้อมูลจากบัฟเฟอร์ กรณีที่ข้อมูลในบัฟเฟอร์ส่วนที่ 1 มีพอสำหรับการอ่าน

จากรูปที่ 8.2 แสดงให้เห็นการอ่านข้อมูลจากบัฟเฟอร์ กรณีที่ข้อมูลในบัฟเฟอร์ส่วนที่ 1 มีพอสำหรับการอ่านข้อมูลเพื่อนำไปแสดงเป็นภาพวีดีโอ โดยขนาดของข้อมูลที่ต้องการอ่านเพื่อนำไปแสดงเป็นภาพวีดีโอ จะถูกใส่ไว้ในตัวแปร dwBytesToRead โดยจากรูปสีจางในบัฟเฟอร์ส่วนที่ 1 คือ ส่วนที่ข้อมูลถูกอ่านไปแล้ว และส่วนสีเข้มคือ ส่วนที่ยังมีข้อมูลอยู่ ซึ่งจะเห็นได้ว่าภายหลังการอ่านข้อมูลไปแล้ว ส่วนที่เป็นสีจางในบัฟเฟอร์ส่วนที่ 1 จะมีขนาดมากขึ้นเท่ากับขนาดของข้อมูลที่ได้อ่านไป



รูปที่ 8.3 แสดงกระบวนการอ่านข้อมูลจากบัฟเฟอร์กรณีที่ข้อมูลในบัฟเฟอร์ส่วนที่ 1ไม่พอสำหรับการอ่าน

จากรูปที่ 8.3 แสดงให้เห็นการอ่านข้อมูลจากบัฟเฟอร์ กรณีที่ข้อมูลในบัฟเฟอร์ส่วนที่ 1 มีไม่พอสำหรับการอ่านข้อมูลเพื่อนำไปแสดงเป็นภาพวีดีโอ โดยจากรูปจะเห็นได้ว่าข้อมูลในบัฟเฟอร์ส่วนที่ 1 มีไม่พอสำหรับการอ่านเพื่อแสดงเป็นภาพวีดีโอ ดังนั้นจึงต้องนำเอาบัฟเฟอร์ส่วนที่ 2 มาใช้งาน โดยการอ่านข้อมูลจะต้องแบ่งเป็น 2 ส่วน แล้วจึงนำมารวมกัน เป็นข้อมูลที่จะใช้ในการแสดงภาพวีดีโอ ซึ่งภาย

หลังการอ่านข้อมูลแล้ว จะเห็นได้ว่าข้อมูลในบัฟเฟอร์ส่วนที่ 1 ถูกอ่านไปจนหมดแล้ว ส่วนข้อมูลในบัฟเฟอร์ส่วนที่ 2 จะถูกอ่านไปบางส่วนดังรูป ดังนั้นบัฟเฟอร์ส่วนที่ 1 จะถูกทิ้งไป และนำเอาบัฟเฟอร์ส่วนที่ 2 มาใช้เป็นบัฟเฟอร์ส่วนที่ 1 เพื่อใช้ในการอ่านข้อมูลเพื่อนำไปแสดงภาพวีดีโอครั้งต่อไป

8.2 คลาส CMemStream

คลาส CMemStream จะทำหน้าที่ในส่วนของการส่งตำแหน่งของไฟล์วีดีโอที่ต้องการแสดงผลไปให้เซิร์ฟเวอร์ และรอรับสตรีมของวีดีโอจากเครื่องเซิร์ฟเวอร์มาเก็บลงบัฟเฟอร์ เพื่อรอการอ่านไปแสดงเป็นภาพวีดีโอต่อไป โดยจะมีเมธอดที่ทำหน้าหลักคือ

- เมธอด Read( )

เมธอดนี้เป็นเมธอดที่ใช้ในการติดต่อกับเซิร์ฟเวอร์ โดยจะรับตำแหน่งของไฟล์ที่ต้องการอ่านมาจากส่วนอ่านข้อมูลวีดีโอ และส่งไปบอกเซิร์ฟเวอร์ให้ส่งข้อมูลวีดีโอตามตำแหน่งที่ส่งไป จากนั้นรอรับข้อมูลวีดีโอ และนำข้อมูลวีดีโอที่ได้รับไปเก็บลงในบัฟเฟอร์ โดยกระบวนการที่ใช้ในการอ่านข้อมูลจากบัฟเฟอร์ภายในเมธอดนี้จะกล่าวในหัวข้อที่ 8.2.1 เมื่ออ่านข้อมูลวีดีโอได้แล้ว จะส่งต่อไปให้ส่วนอ่านข้อมูลวีดีโอเพื่อนำไปแสดงเป็นภาพวีดีโอต่อไป

8.2.1 กระบวนการอ่านข้อมูลจากบัฟเฟอร์

การอ่านสตรีมของวีดีโอ ขั้นแรกจำเป็นที่จะต้องรู้ขนาดของไฟล์วีดีโอก่อน โดยขนาดของไฟล์ของวีดีโอจะถูกเก็บไว้ในตัวแปร m_llLength ซึ่งถูกส่งมาจากเซิร์ฟเวอร์ และในการอ่านข้อมูลแต่ละครั้งก็จะมีตัวแแปร m_llPosition ที่ใช้เก็บตำแหน่งของข้อมูลที่ต้องการอ่านในปัจจุบัน โดยตัวแปร m_llPosition จะต้องถูกเพิ่มค่าเท่ากับจำนวนไบต์ที่อ่านในแต่ละครั้งทุกครั้ง เพื่อให้ทราบว่าตำแหน่งที่จะต้องอ่านครั้งต่อไปอยู่ที่ตำแหน่งที่เท่าไร จึงจะอ่านข้อมูลมาจากบัฟเฟอร์ได้อย่างถูกต้อง สำหรับตัวแปร m_llPosition นั้นสามารถเปลี่ยนแปลงได้เพื่อใช้กำหนดตำแหน่งของไฟล์วีดีโอที่ต้องการอ่าน โดยจะเห็นได้ว่าขนาดของไฟล์วีดีโอจะมีขนาดใหญ่มาก ดังนั้นการที่จะส่งไฟล์ของวีดีโอมาใส่ในบัฟเฟอร์ในครั้งเดียวย่อมเป็นไปไม่ได้ จึงจำเป็นที่จะต้องใช้การอ่านไฟล์วีดีโอมาทีละส่วน และเก็บไว้ในบัฟเฟอร์ จากนั้นรอการอ่านจากส่วนอ่านข้อมูลวีดีโอ เมื่ออ่านบัฟเฟอร์ไปจนหมด จึงค่อยอ่านข้อมูลจากไฟล์วีดีโอมาเก็บไว้ใหม่ สำหรับการอ่านข้อมูลมาใส่ในบัฟเฟอร์ในขั้นตอนนี้ คลาส CMemStream จะมีบัฟเฟอร์ไว้ 2 ส่วน เพื่อสลับกันใช้งาน ทำให้การอ่านข้อมูลเป็นไปอย่างรวดเร็วขึ้น

สำหรับบัฟเฟอร์มี 2 ส่วน คือ m_pbData และm_pbDataBuffer ซึ่งมีขนาดเท่ากันตามแต่ผู้ใช้จะกำหนด โดยในการอ่านข้อมูลแต่ละครั้ง ส่วนอ่านข้อมูลวีดีโอจะส่งจำนวนของไบต์ที่ต้องการอ่านมาให้ จากนั้นเมธอด Read() ของคลาส CMemStream จะทำการอ่านข้อมูลจากบัฟเฟอร์ และส่งไปให้ส่วนอ่าน

ข้อมูลวีดีโอ โดยจะใช้ตัวแปร m_llPosition ในการชี้ตำแหน่งของไฟล์วีดีโอที่ต้องการอ่าน แต่ตัวแปร m_llPosition นั้นจะเป็นตัวแปรที่ชี้ตำแหน่งจริงของไฟล์วีดีโอที่ต้องการอ่าน ซึ่งในการอ่านไม่ได้อ่านจากไฟล์วีดีโอจริงๆไม่สามารถจะชี้ตำแหน่งไปยังตำแหน่งใดก็ได้ในไฟล์วีดีโอจะชี้ตำแหน่งได้เฉพาะตำแหน่งที่ไม่เกินขนาดของบัฟเฟอร์เท่านั้น ดังนั้นจึงจะต้องคำนวณหาตำแหน่งที่ต้องการอ่านโดยใช้การคำนวณต่อไปนี้

offset = m_llPosition MOD m_bufferLength

offset คือ ตำแหน่งที่ต้องการอ่านในบัฟเฟอร์

m_llPosition คือ ตำแหน่งที่ต้องการอ่านจริงในไฟล์วีดีโอ

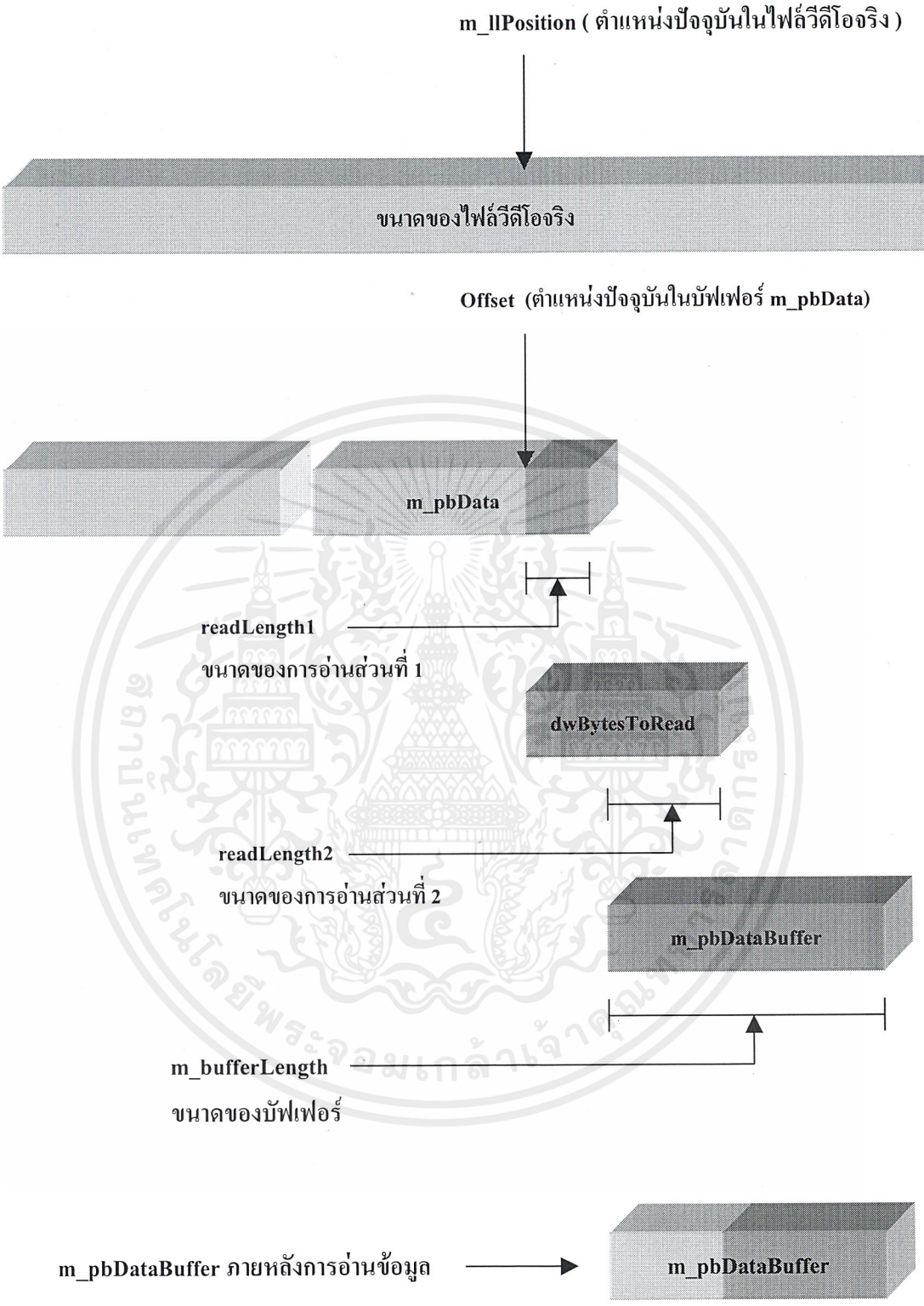
m_bufferLength คือ ขนาดของบัฟเฟอร์

เมื่อได้ค่า offset แล้วก็จะทราบตำแหน่งเริ่มต้นในการอ่านข้อมูลจากบัฟเฟอร์ m_pbData โดยจะเริ่มต้นที่ตำแหน่ง offset และสิ้นสุดที่ตำแหน่ง offset + dwBytesToRead (dwBytesToRead คือ จำนวนของไบต์ที่ต้องการอ่านจากบัฟเฟอร์เพื่อนำไปแสดงเป็นภาพวีดีโอ) ซึ่งถ้าตำแหน่งสิ้นสุดมีค่าน้อยกว่าขนาดของบัฟเฟอร์ ก็สามารถอ่านข้อมูลไปได้เลย แต่ถ้าตำแหน่งสิ้นสุดมีค่ามากกว่า หรือเท่ากับขนาดของบัฟเฟอร์ นั้นจะทำให้ข้อมูลที่มีอยู่ในบัฟเฟอร์ m_pbData มีไม่พอสำหรับการอ่านข้อมูลในครั้งนั้น จึงต้องนำเอาบัฟเฟอร์ m_pbDataBuffer ซึ่งได้สำรองข้อมูลส่วนต่อไปไว้ให้เรียบร้อยแล้ว โดยการอ่านจะต้องแบ่งการอ่านข้อมูลออกเป็น 2 ส่วน ดังต่อไปนี้

- ส่วนที่ 1 อ่านเริ่มต้นจากตำแหน่ง offset ไปจนสุดบัฟเฟอร์ m_pbData โดยขนาดการอ่านส่วนที่1 จะมีค่าดังนี้ readLength1 = m_bufferLength – offset

- ส่วนที่ 2 อ่านตั้งแต่จุดเริ่มต้นของบัฟเฟอร์ m_pbDataBuffer ไปจนถึงตำแหน่ง readLength2 ซึ่งคำนวณได้ดังนี้ readLength2 = dwBytesToRead - readLength1

เมื่ออ่านข้อมูลทั้ง 2 ส่วนเรียบร้อย จึงส่งข้อมูลต่อไปให้กับส่วนอ่านข้อมูลวีดีโอ ผ่านพอยท์เตอร์ pbBuffer ซึ่งถูกส่งผ่านเข้ามาโดยเมธอด Read() ของคลาส CMemStream โดยการก๊อปปี้ข้อมูลที่อ่านได้มาไปยังเมมโมรีที่ตำแหน่งที่ชี้โดยพอยท์เตอร์ pbBuffer



รูปที่ 8.4 แสดงการอ่านข้อมูลจากบัฟเฟอร์ในกรณีที่ dwBytesToRead มีขนาดมากกว่าขนาดของ readLength1

8.3 คลาส CMemReader

การทำงานในส่วนของการอ่านข้อมูลจากบัฟเฟอร์ เพื่อเตรียมพร้อมไว้ให้คลาส CPlayStream นำไปแสดงผล จะเป็นหน้าที่ของคลาส CMemReader ซึ่งจะเปรียบเสมือนซอร์สฟิลเตอร์ โดยจะมีเอาท์พุทพินซึ่งหาได้จากการเรียกใช้เมธอด GetPin() ในคลาส CMemReader เพื่อใช้เชื่อมต่อกับฟิลเตอร์ที่สร้างขึ้นโดยส่วนแสดง และควบคุมภาพวีดีโอ(คลาส CPlayStream)

สำหรับการใช้งานคลาส CMemReader นั้นจำเป็นจะต้องระบุชนิดของไฟล์วีดีโอก่อนที่จะทำการสร้างอินสแตนซ์ของคลาส CMemReader ดังต่อไปนี้

ขั้นที่ 1 สร้างตัวแปรที่ใช้เก็บชนิดของมีเดียไฟล์

```
CMediaType mt;      // ตัวแปร mt จะใช้เก็บชนิดของมีเดียไฟล์เพื่อใช้ในการสร้างอินสแตนซ์ของคลาส
                    CMemReader
```

ขั้นที่ 2 กำหนดชนิดของมีเดียไฟล์

```
mt.majortype  =  MEDIATYPE_Stream;
mt.subtype    =  MEDIASUBTYPE_MPEG1VideoCD   //ใช้กับมีเดียไฟล์ที่มีนามสกุล ".dat "
```

การกำหนดค่า subtype นั้นสามารถกำหนดได้หลายแบบตามแต่ละชนิดของมีเดียไฟล์ ซึ่งจะมีชนิดต่างๆที่สามารถกำหนดได้ดังต่อไปนี้

- MPEGSUBTYPE_MPEG1System ใช้กับมีเดียไฟล์ที่มีนามสกุล ".mpg "
- MPEGSUBTYPE_MPEG1Audio ใช้กับมีเดียไฟล์ที่มีนามสกุล ".mpa"
- MPEGSUBTYPE_MPEG1Video ใช้กับมีเดียไฟล์ที่มีนามสกุล ".mpv"
- MPEGSUBTYPE_MPEG1VideoCD ใช้กับมีเดียไฟล์ที่มีนามสกุล ".dat"
- MPEGSUBTYPE_Avi ใช้กับมีเดียไฟล์ที่มีนามสกุล ".avi"
- MPEGSUBTYPE_QTMovie ใช้กับมีเดียไฟล์ที่มีนามสกุล ".mov"
- MPEGSUBTYPE_WAV ใช้กับมีเดียไฟล์ที่มีนามสกุล ".wav"

ขั้นที่ 3 สร้างอินสแตนซ์ของคลาส CMemReader

```
HRESULT   hr;
CMemReader *p_mr  =  new  CMemReader(p_ms, &mt, &hr);
```

p_ms คือ พอยท์เตอร์ที่ชี้ไปยังอินสแตนซ์ของคลาส CMemStream

mt คือ ชนิดของมีเดียไฟล์ที่ได้กำหนดไว้ข้างต้น

hr คือ ผลลัพธ์ของการสร้างอินสแตนซ์ของคลาส CMemReader

p_mr คือ พอยท์เตอร์ที่ชี้ไปยังอินสแตนซ์ของคลาส CMemReader

8.4 คลาส CPlayStream

การทำงานในส่วนของการแสดงภาพและควบคุมภาพวีดีโอจะเป็นหน้าที่ของคลาส CPlayStream โดยคลาสนี้จะเป็นตัวสร้างฟิลเตอร์กราฟทั้งหมดที่จำเป็นในการแสดงผลภาพวีดีโอ โดยภายในคลาสจะมีเมธอดที่ทำหน้าที่หลักคือ

เมธอด CreateFilter( )

เมธอดนี้จะเป็นเมธอดหลักที่ใช้ในการสร้างฟิลเตอร์กราฟทั้งหมด รวมทั้งการเรียกใช้อินเตอร์เฟสต่างๆเพื่อนำมาใช้ในการควบคุมการแสดงภาพวีดีโอ โดยจะมีการทำงานดังต่อไปนี้

- สร้างอินสแตนซ์ของฟิลเตอร์กราฟ โดยจะต้องระบุอินเตอร์เฟสเป็น IFilterGraph

```
HRESULT hr;
IFilterGraph *p_fg;        //พอยท์เตอร์ที่ชี้ไปยังอินเตอร์เฟส IFilterGraph
hr = CoCreateInstance(CLSID_FilterGraph,NULL,CLSCTX_INPROC,IID_IFilterGraph,
(void**)&p_fg);
```

ถ้าเป็นผลสำเร็จ hr จะมีค่าเท่ากับ S_OK ซึ่งในขั้นตอนนี้จะมีฟิลเตอร์กราฟเกิดขึ้นแล้ว โดยจะมีพอยท์เตอร์ p_fg เป็นพอยท์เตอร์ที่ชี้ไปยังอินเตอร์เฟส IFilterGraph เพื่อเรียกใช้เมธอดต่างๆ โดยตอนนี้ฟิลเตอร์กราฟยังว่างอยู่ เพราะยังไม่ได้สร้างฟิลเตอร์ให้กับฟิลเตอร์กราฟ

- เรียกใช้ IGraphBuilder อินเตอร์เฟส เพื่อทำหน้าที่เป็นตัวจัดการฟิลเตอร์กราฟ (Filter Graph Manager)

```
IGraphBuilder *p_gb;
hr = p_fg ->QueryInterface(IID_ IGraphBuilder, (void**)&p_gb);
```

ถ้าเป็นผลสำเร็จ hr จะมีค่าเท่ากับ S_OK และจะได้พอยท์เตอร์ p_gb ที่ชี้ไปยังอินเตอร์เฟส IGraphBuilder ซึ่งจะถูกใช้เป็นตัวจัดการฟิลเตอร์ โดยจะทำหน้าที่ในการสร้างฟิลเตอร์ที่จำเป็นทั้งหมดในการแสดงภาพไฟล์วีดีโอ

- เพิ่มฟิลเตอร์ของคลาส CMemReader เข้าไปในฟิลเตอร์กราฟเพื่อทำหน้าที่เป็นซอร์สฟิลเตอร์

```
p_gb->AddFilter(p_mr,NULL);      // p_mr คือ พอยท์เตอร์ที่ชี้ไปยังอินสแตนซ์ของคลาส CMemReader
```

- สร้างตัวแปรที่ใช้เก็บพอยท์เตอร์ที่ชี้ไปยังอินเตอร์เฟสต่างๆที่ต้องใช้ในการแสดง และควบคุมภาพวีดีโอ

```
IMediaControl *p_mc;          // ใช้ในการควบคุมการแสดงภาพวีดีโอ เช่น เล่นภาพ ,หยุดภาพ
IMediaEventEx *p_me;          // ใช้ในการควบคุมเหตุการ์ณต่างๆที่เกิดขึ้น
IVideoWindow *p_vw;           // ใช้จัดการกับการแสดงภาพวีดีโอ
```

```
IMediaSeeking *p_mseek;          // ใช้ในการค้นหา และระบุตำแหน่งในสตรีมของวีดีโอ
```

- เรียกใช้อินเตอร์เฟสต่างๆที่ต้องใช้ในการแสดง และควบคุมภาพวีดีโอ

```
p_fg->QueryInterface(IID_IMediaControl,(void**)&p_mc);
p_fg->QueryInterface(IID_IMediaEventEx,(void**)&p_me);
p_fg->QueryInterface(IID_IVideoWindow, (void **)&p_vw);
p_fg->QueryInterface(IID_IMediaSeeking, (void **)&p_mseek);
```

- การสร้างฟิลเตอร์กราฟที่จำเป็นทั้งหมดในการแสดงภาพวีดีโอ

```
HRESULT  hr;
hr = p_gb->Render(p_mr->GetPin(0));
```

ถ้าเป็นผลสำเร็จฟิลเตอร์ที่จำเป็นทั้งหมดจะถูกสร้างให้อัตโมมัติ สำหรับ คำสั่ง p_mr->GetPin(0) นั้นเป็นการเรียกใช้เอาท์พุทพินของ คลาส CMemReader เพื่อนำไปเชื่อมต่อกับฟิลเตอร์ที่ถูกสร้างขึ้นโดยคำสั่ง Render() ของ IGraphBuilder อินเตอร์เฟส

- การกำหนดตำแหน่งและขนาดของวินโดว์สที่ใช้ในการแสดงผล

```
RECT r;          // r เป็นตัวแปรใช้เก็บขนาดของวินโดว์สที่ใช้ในการแสดงผล
p_vw->put_Owner((OAHWND)m_hWnd);   //m_hWnd เป็นตัวแปรทีใช้เก็บแฮนเดิลวินโดว์สที่ใช้แสดง
                                   ภาพวีดีโอ
p_vw->put_WindowStyle(WS_CHILD|WS_CLIPCHILDREN|WS_CLIPSIBLINGS);
GetClientRect(m_hWnd,&r);
p_vw->SetWindowPosition(r.left,r.top,r.right,r.bottom);
```

- การแสดงภาพวีดีโอแบบเต็มจอภาพ

```
p_vw->put_FullScreen(OATRUE);
```

- การกำหนดรูปแบบชนิดข้อมูลในการแสดงภาพวีดีโอ

```
p_mseek->SetTimeFormat(&TIME_FORMAT_FRAME);
```

- การรับขนาดของไฟล์วีดีโอมาเก็บไว้ที่ตัวแปร m_frameRange

```
p_mseek->GetDuration(&m_frameRange);
```

- การกำหนดให้วินโดว์สที่ใช้ในการแสดงผลรับเมสเสจจาก DirectShow

```
p_vw->put_MessageDrain((OAHWND)m_hWnd);
```

การรับเมสเสจจะใช้เพื่อตรวจสอบการกดคีย์ต่างของผู้ใช้

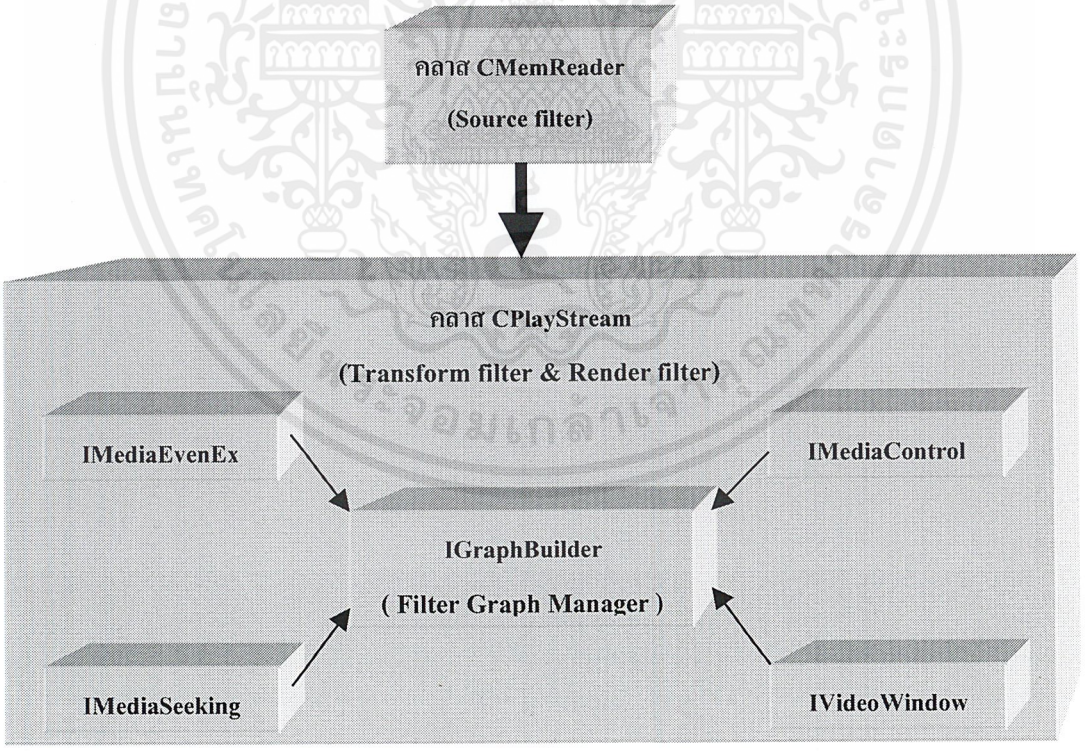
- การกำหนดตำแหน่งเริ่มต้น และสิ้นสุดในการแสดงภาพวีดีโอ

```
LONGLONG start = 0L;          //เป็นตัวแปรใช้เก็บตำแหน่งเริ่มต้นของการแสดงภาพวีดีโอ
LONGLONG stop = 10L;          //เป็นตัวแปรใช้เก็บตำแหน่งสิ้นสุดของการแสดงภาพวีดีโอ
p_mseek->SetPositions(&start,AM_SEEKING_AbsolutePositioning,&m_frameRange,
                      AM_SEEKING_AbsolutePositioning);
```

สำหรับเมธอด SetPosition() นี้จะถูกนำไปใช้ในการเดินหน้า หรือถอยหลังภาพวีดีโอ โดยจะสามารถกำหนดตำแหน่งที่ต้องการเดินหน้า หรือ ถอยได้ตามต้อง ด้วยการเปลี่ยนแปลงตำแหน่งเริ่มต้นที่ให้กับเมธอด SetPosition()

- การเล่น และหยุดภาพวีดีโอ

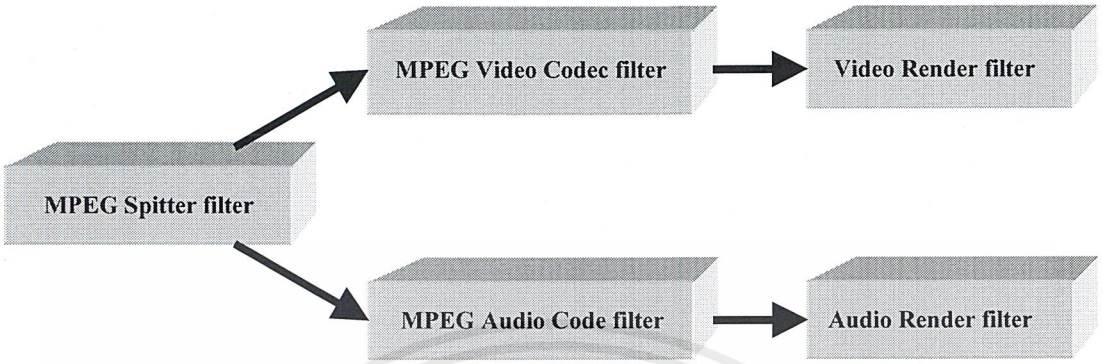
```
p_mc->Run();      //ใช้ในการเล่นภาพวีดีโอ
p_mc->Stop();     //ใช้ในการหยุดภาพวีดีโอ
```



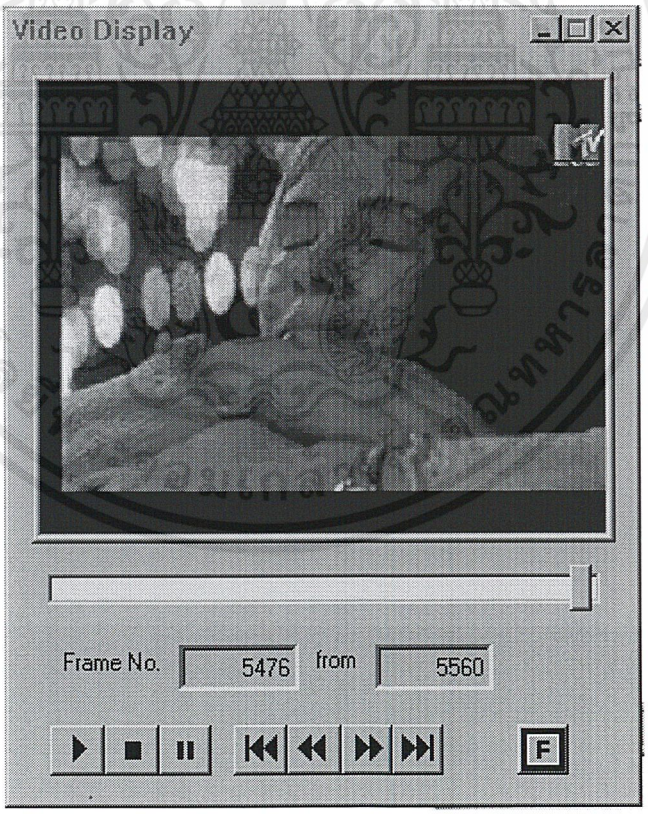
รูปที่ 8.5 แสดงการเชื่อมต่อกันของคลาส และอินเตอร์เฟสต่างๆ

จากรูปที่ 8.5 ส่วนของ IGraphBulider อินเตอร์เฟสจะเป็นตัวจัดการฟิลเตอร์ซึ่งจะทำหน้าที่ในการสร้างฟิลเตอร์ที่จำเป็นให้อย่างอัตโนมัติ เมื่อมีการเรียกคำสั่ง Render() สำหรับฟิลเตอร์ต่างๆที่ได้สร้าง

ขึ้นโดยตัวจัดการฟิลเตอร์จะแสดงดังรูปที่ 8.6 และส่วนอินเตอร์เฟสอื่นๆจะทำหน้าที่คอยควบคุมการทำงานของฟิลเตอร์ต่างๆที่ได้สร้างขึ้น ตามหน้าที่ของแต่ละอินเตอร์เฟส



รูปที่ 8.6 แสดงฟิลเตอร์ที่ถูกสร้างโดยตัวจัดการฟิลเตอร์ (Filter Graph Manager)



รูปที่ 8.7 แสดงหน้าจอของโปรแกรมรับสตรีม และแสดงภาพวีดีโอ MPEG-1

บทที่ 9
การทดสอบแอพพลิเคชั่นในโครงงานวิจัย

ในบทนี้จะกล่าวถึงรายละเอียดของฮาร์ดแวร์ต่างๆที่จำเป็นต้องนำมาใช้ในการทดสอบการทำงานของโปรแกรมที่ได้สร้างขึ้นในโครงงานครั้งนี้ รวมทั้งขั้นตอนและวิธีในการทดสอบโปรแกรม

9.1 ฮาร์ดแวร์ที่ใช้ในการทดสอบโปรแกรม

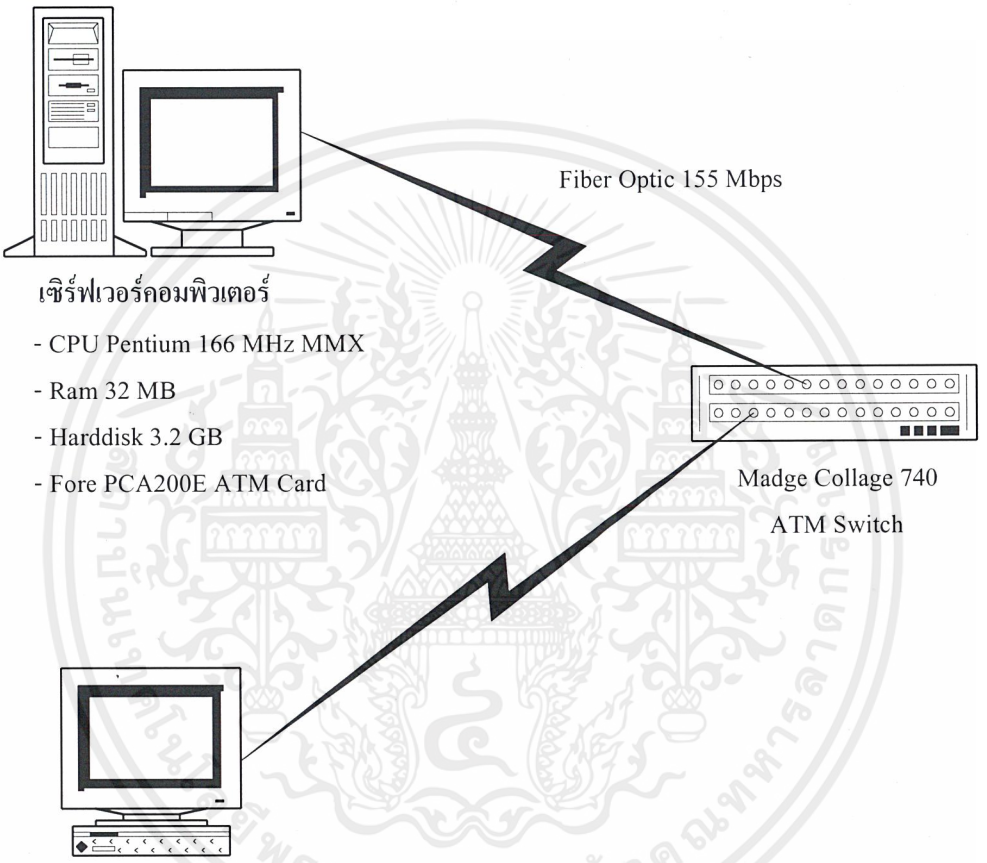
ฮาร์ดแวร์ที่จำเป็นต้องใช้ในการทดสอบการทำงานของโปรแกรมที่ได้สร้างขึ้นในโครงงาน มีดังต่อไปนี้

1. เครื่องเซิร์ฟเวอร์คอมพิวเตอร์
   CPU : Pentium 166 MHz MMX
   RAM : 32 MB
   HARDDISK : 3.2 GB
2. เครื่องไคลเอนต์คอมพิวเตอร์
   CPU : Pentium Celeron 333 MHz
   RAM : 64 MB
   HARDDISK : 3.2 GB
3. Fore Runner LE 155 Mbps ATM card
   ติดตั้งไว้กับเครื่องไคลเอนต์คอมพิวเตอร์
4. Fore PCA200E 155 Mbps ATM card
   ติดตั้งไว้กับเครื่องเซิร์ฟเวอร์คอมพิวเตอร์
5. สายไฟเบอร์ออปติกแบบ SC connector จำนวน 4 เส้น
6. Madge Collage 740 ATM Switch

หมายเหตุ : สาเหตุที่ต้องให้เครื่องไคลเอนต์มีความเร็วสูง เพราะมีผลต่อการแสดงภาพวีดีโอ MPEG ถ้าการทดสอบใช้เครื่องไคลเอนต์ที่มีความเร็วต่ำจะทำให้ภาพวีดีโอกระตุก ไม่ราบรื่น เนื่องจากไม่สามารถประมวลผลภาพวีดีโอได้ทันเวลา โดยภาพที่กระตุกไม่ได้เกิดจากความล่าช้าของเครือข่าย ATM ซึ่งจะทำให้การทดสอบผิดพลาด และสำหรับเครื่องเซิร์ฟเวอร์ที่ใช้ความเร็วไม่สูงนัก เพราะตัวเครื่องเซิร์ฟเวอร์ทำหน้าที่ในการส่งสตรีมของวีดีโอเท่านั้น จึงไม่ต้องใช้ความเร็วสูงนัก และจากทดสอบก็พบว่าสามารถส่งสตรีมของวีดีโอได้ทันเวลา โดยภาพวีดีโอที่ได้ไม่เกิดการกระตุก

9.2 การจัดเตรียมฮาร์ดแวร์ในการทดสอบ

ในขั้นตอนนี้จะทำการติดตั้งฮาร์ดแวร์ต่างๆ เพื่อให้พร้อมสำหรับการทดสอบโปรแกรม โดยได้ทำการติดตั้ง Fore Runner LE ATM Card ลงบนเครื่องไคลเอนต์คอมพิวเตอร์ และติดตั้ง Fore PCA200E ATM Card ลงบนเครื่องเซิร์ฟเวอร์คอมพิวเตอร์ จากนั้นจึงต่อสายไฟเบอร์ออปติกจากเครื่องเซิร์ฟเวอร์คอมพิวเตอร์ และเครื่องไคลเอนต์คอมพิวเตอร์ ไปยัง Madge Callage 740 ATM Switch ดังรูปที่ 9.2



รูปที่ 9.1 แสดงการติดตั้งฮาร์ดแวร์ เพื่อการทดสอบโปรแกรม

9.3 การทดสอบโปรแกรม

การทดสอบโปรแกรมจะแบ่งการทดสอบออกเป็น 2 แบบ คือ การส่งข้อมูลแบบดาวน์โหลดไฟล์ไปทั้งไฟล์ และการรับสตรีมของวีดีโอมาแสดงเป็นภาพวีดีโอ โดยแต่ละแบบจะทดสอบทั้งบริการแบบ CBR และ UBR สำหรับบริการแบบ VBR ไม่สามารถทดสอบได้ เพราะจากการทดสอบพบว่า Fore PCA200E ATM Card ไม่สนับสนุนบริการแบบ VBR ทำให้ไม่สามารถสร้างการติดต่อระหว่างเซิร์ฟเวอร์คอมพิวเตอร์ และไคลเอนต์คอมพิวเตอร์ได้

การทดสอบส่งข้อมูลแบบดาวน์โหลดไฟล์ไปทั้งไฟล์

ทำการทดสอบโดยการทดลองส่งไฟล์จากเครื่องเซิร์ฟเวอร์ไปยังไคลเอนต์ โดยใช้บริการแบบ CBR แล้วลองปรับเปลี่ยนค่า PCR (Peak Cell Rate) ให้มากขึ้น และน้อยลง เพื่อทดสอบการจองแบนด์วิธในการส่งข้อมูลบนเครือข่าย ATM

ผลการทดสอบพบว่า เมื่อปรับเปลี่ยนค่า PCR จะทำให้ความเร็วในการส่งข้อมูลเปลี่ยนแปลงไป โดยถ้าเพิ่มค่า PCR ให้มากขึ้น ความเร็วในการส่งข้อมูลก็จะเพิ่มขึ้น และถ้าลดค่า PCR ให้น้อยลง ความเร็วในการส่งข้อมูลก็จะลดลง ซึ่งจากจุดนี้จะเป็นการแสดงให้เห็นถึงคุณสมบัติของระบบเครือข่าย ATM ที่สามารถจองแบนด์วิธได้ตามต้องการ แต่จะต้องไม่เกินขนาดของแบนด์วิธมากสุดที่สามารถส่งได้และต้องไม่เกินความสามารถของฝ่ายรับและฝ่ายส่งที่จะจัดการข้อมูลได้

การทดสอบการรับสตรีมของวีดีโอมาแสดงเป็นภาพวีดีโอ

ทำการทดสอบโดยการทดลองส่งสตรีมของวีดีโอ MPEG-1 จากเครื่องเซิร์ฟเวอร์ไปยังไคลเอนต์ โดยใช้บริการแบบ UBR และ CBR แล้วลองปรับเปลี่ยนค่า PCR ให้มากขึ้น และน้อยลง เพื่อทดสอบการจองแบนด์วิธในการส่งข้อมูลบนเครือข่าย ATM

ผลการทดสอบพบว่า เมื่อเลือกบริการแบบ UBR ภาพวีดีโอที่ได้จะมีการกระตุกบ้างเป็นบางครั้งขึ้นอยู่กับสถานะการใช้งานเครือข่าย ATM ในขณะนั้น ซึ่งจากจุดนี้จะเป็นการแสดงให้เห็นถึงคุณสมบัติของบริการแบบ UBR ที่ไม่สามารถกำหนดแบนด์วิธในการส่งข้อมูลได้ และเมื่อทดสอบต่อไปโดยเลือกบริการเป็นแบบ CBR เมื่อปรับเปลี่ยนค่า PCR จะทำให้คุณภาพของภาพวีดีโอที่ได้แตกต่างกันไป โดยถ้าเพิ่มค่า PCR ให้มากขึ้นจะทำให้ภาพวีดีโอมีคุณภาพดีขึ้น ไม่กระตุก แต่ถ้าลดค่า PCR ให้น้อยลงจะทำให้ภาพวีดีโอ กระตุก ไม่ราบรื่น ซึ่งจากจุดนี้จะเป็นการแสดงให้เห็นถึงคุณสมบัติของระบบเครือข่าย ATM ที่สามารถจองแบนด์วิธได้ตามต้องการ และสามารถกำหนดคุณภาพของบริการในการส่งข้อมูลได้ สำหรับค่า PCR ที่ใช้ในการทดสอบแล้วทำให้ภาพวีดีโอที่ได้มีคุณภาพดี ภาพไม่กระตุก โดยประมาณคือ 50,000 เซลล์ขึ้นไป

บทที่ 10
บทสรุปและวิจารณ์

ระบบเครือข่าย ATM เป็นระบบเครือข่ายความเร็วสูง ที่สามารถรองรับการใช้งานมัลติมีเดีย แต่ก็ยังเป็นเทคโนโลยีที่ใหม่และมีราคาแพงอยู่ ทำให้ยังขาดแอพพลิเคชั่นที่จะมาสนับสนุนการใช้งานให้เกิดประสิทธิภาพสูงสุด โดยการทำงานส่วนใหญ่ยังต้องใช้การทำ LAN Emulation และ IP over ATM เพื่อให้แอพพลิเคชั่นที่มีอยู่ทั่วไปสามารถทำงานบนระบบเครือข่าย ATM ได้ แต่ก็จะทำให้ประสิทธิภาพของการใช้งานระบบเครือข่าย ATM ลดลง ถ้าแอพพลิเคชั่น ที่นำมาใช้ในระบบเครือข่าย ATM สามารถทำงานได้ทันทีโดยไม่ต้องผ่านการทำ LAN Emulation หรือ IP over ATM ก็จะสามารถใช้งานระบบเครือข่าย ATM ได้อย่างเต็มประสิทธิภาพ

ดังนั้นการนำระบบเครือข่าย ATM มาใช้งานควรจะต้องพิจารณาที่จุดประสงค์ของการใช้งานว่านำมาใช้งานประเภทใด หากนำมาใช้เพื่อทดแทนระบบเครือข่าย LAN เดิมก็จะไม่เหมาะสม หากเปรียบเทียบประสิทธิภาพที่ได้รับกับค่าใช้จ่ายที่เสียไปในการลงทุนก็จะไม่คุ้มค่า แต่หากเป็นการนำมาเพื่อใช้งานสำหรับงานประเภท Video conference, Video on demand ก็จะเหมาะสมอย่างยิ่งเพราะสามารถใช้งานระบบได้อย่างคุ้มค่า และมีประสิทธิภาพ

สำหรับโครงงานนี้ได้ทำการศึกษาระบบเครือข่าย ATM และทำการสร้างแอพพลิเคชั่นสำหรับเครือข่าย ATM ซึ่งขณะดำเนินงานจะประสบปัญหาค่อนข้างมาก ดังต่อไปนี้

- อุปกรณ์ที่ใช้งานหาได้ยาก เช่น การ์ด ATM , สวิตช์ ATM ทำให้การทดสอบโปรแกรมทำได้ยากลำบาก
- ข้อมูลที่ต้องใช้ในโครงงานวิจัยหาได้ยาก เพราะ ATM ยังเป็นระบบที่ใหม่ และไม่แพร่หลายนัก อีกทั้งยังไม่มีโครงงานต้นแบบใดทำเรื่องนี้มาก่อน
- เซอร์วิสโพวายเดอร์อินเตอร์เฟส (SPI) ของแต่ละบริษัทยังไม่สมบูรณ์หรือทำงานร่วมกับ SPI ของบริษัทอื่นไม่ได้ ทำให้การเขียนแอพพลิเคชั่นในส่วนการสร้างการติดต่อด้วย WinSock2.0 ไม่สามารถทำงานได้ จึงมีความจำเป็นต้องใช้ ATM การ์ดของบริษัท Fore เพียงบริษัทเดียวในการทำงานโครงงานวิจัย

อย่างไรก็ดี แอพพลิเคชั่นที่ได้พัฒนาขึ้นในโครงงานนี้ ก็สามารถทำงานได้ตรงตามจุดประสงค์ในการใช้งานระบบเครือข่าย ATM เพราะสามารถนำเอาคุณลักษณะ และประสิทธิภาพที่มีในระบบเครือ ATM มาแสดงให้เห็นอย่างชัดเจน

ภาคผนวก ก

WinSock 2.0 ATM เฮดเดอร์ไฟล์

ไฟล์เฮดเดอร์ " ws2atm.h " ซึ่งใช้ในการกำหนดพารามิเตอร์ต่างๆของ ATM

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#ifndef _WS2ATM_H_
#define _WS2ATM_H_

#pragma pack(1)

#define ATMPROTO_AALUSER        0x00  /* User-defined AAL */
#define ATMPROTO_AAL1           0x01  /* AAL 1 */
#define ATMPROTO_AAL2           0x02  /* AAL 2 */
#define ATMPROTO_AAL34          0x03  /* AAL 3/4 */
#define ATMPROTO_AAL5           0x05  /* AAL 5 */

#define SAP_FIELD_ABSENT        0xFFFFFFFE
#define SAP_FIELD_ANY           0xFFFFFFFF
#define SAP_FIELD_ANY_AESA_SEL  0xFFFFFFFA
#define SAP_FIELD_ANY_AESA_REST 0xFFFFFFFB

/*
 * values used for AddressType in struct ATM_ADDRESS
 */
#define ATM_E164            0x01  /* E.164 addressing scheme */
#define ATM_NSAP            0x02  /* NSAP-style ATM Endsystem Address scheme */
#define ATM_AESA            0x02  /* NSAP-style ATM Endsystem Address scheme */

typedef struct {
  DWORD  AddressType;     /* E.164 or NSAP-style ATM Endsystem Address */
  DWORD  NumofDigits;     /* number of digits; */
  UCHAR  Addr[20];        /* IA5 digits for E164, BCD encoding for NSAP */
                          /* format as defined in the ATM Forum UNI 3.1 */
```

```
} ATM_ADDRESS;

/*
 * values used for Layer2Protocol in B-LLI
 */
#define BLLI_L2_ISO_1745        0x01   /* Basic mode ISO 1745                */
#define BLLI_L2_Q921            0x02   /* CCITT Rec. Q.921                   */
#define BLLI_L2_X25L            0x06   /* CCITT Rec. X.25, link layer        */
#define BLLI_L2_X25M            0x07   /* CCITT Rec. X.25, multilink         */
#define BLLI_L2_ELAPB           0x08   /* Extended LAPB; for half duplex operation */
#define BLLI_L2_HDLC_NRM        0x09   /* HDLC NRM (ISO 4335)                */
#define BLLI_L2_HDLC_ABM        0x0A   /* HDLC ABM (ISO 4335)                */
#define BLLI_L2_HDLC_ARM        0x0B   /* HDLC ARM (ISO 4335)                */
#define BLLI_L2_LLC             0x0C   /* LAN logical link control (ISO 8802/2)  */
#define BLLI_L2_X75             0x0D   /* CCITT Rec. X.75, single link procedure  */
#define BLLI_L2_Q922            0x0E   /* CCITT Rec. Q.922                   */
#define BLLI_L2_USER_SPECIFIED  0x10   /* User Specified                     */
#define BLLI_L2_ISO_7776        0x11   /* ISO 7776 DTE-DTE operation         */

/*
 * values used for Layer3Protocol in B-LLI
 */
#define BLLI_L3_X25             0x06   /* CCITT Rec. X.25, packet layer      */
#define BLLI_L3_ISO_8208        0x07   /* ISO/IEC 8208 (X.25 packet layer for DTE  */
#define BLLI_L3_X223            0x08   /* X.223/ISO 8878                     */
#define BLLI_L3_SIO_8473        0x09   /* ISO/IEC 8473 (OSI connectionless)  */
#define BLLI_L3_T70             0x0A   /* CCITT Rec. T.70 min. network layer  */
#define BLLI_L3_ISO_TR9577      0x0B   /* ISO/IEC TR 9577 Network Layer Protocol ID*/
#define BLLI_L3_USER_SPECIFIED  0x10   /* User Specified                     */

/*
 * values used for Layer3IPI in B-LLI
 */
#define BLLI_L3_IPI_SNAP        0x80   /* IEEE 802.1 SNAP identifier         */
```

```
#define BLLI_L3_IPI_IP          0xCC  /* Internet Protocol (IP) identifier      */

typedef struct {
   DWORD Layer2Protocol;              /* User information layer 2 protocol       */
   DWORD Layer2UserSpecifiedProtocol; /* User specified layer 2 protocol information */
   DWORD Layer3Protocol;              /* User information layer 3 protocol       */
   DWORD Layer3UserSpecifiedProtocol; /* User specified layer 3 protocol information */
   DWORD Layer3IPI;                   /* ISO/IEC TR 9577 Initial Protocol Identifier */
   UCHAR SnapID[5];                   /* SNAP ID consisting of OUI and PID       */
} ATM_BLLI;

/*
 * values used for the HighLayerInfoType field in ATM_BHLI
 */
#define BHLI_ISO                0x00  /* ISO                                     */
#define BHLI_UserSpecific       0x01  /* User Specific                           */
#define BHLI_HighLayerProfile   0x02  /* High layer profile (only in UNI3.0) */
#define BHLI_VendorSpecificAppId  0x03  /* Vendor-Specific Application ID      */

typedef struct {
   DWORD HighLayerInfoType;       /* High Layer Information Type     */
   DWORD HighLayerInfoLength;     /* number of bytes in HighLayerInfo */
   UCHAR HighLayerInfo[8];        /* the value dependent on the      */
                                  /*  HighLayerInfoType field        */
} ATM_BHLI;

struct sockaddr_atm {
   u_short satm_family;           /* address family should be AF_ATM */
   ATM_ADDRESS satm_number;       /* ATM address                     */
   ATM_BLLI satm_blli;            /* B-LLI                           */
   ATM_BHLI satm_bhli;            /* B-HLI                           */
};
```

```
typedef enum {
    IE_AALParameters,
    IE_TrafficDescriptor,
    IE_BroadbandBearerCapability,
    IE_BHLI,
    IE_BLLI,
    IE_CalledPartyNumber,
    IE_CalledPartySubaddress,
    IE_CallingPartyNumber,
    IE_CallingPartySubaddress,
    IE_Cause,
    IE_QOSClass,
    IE_TransitNetworkSelection,
} Q2931_IE_TYPE;

typedef struct {
    Q2931_IE_TYPE IEType;
    ULONG         IELength;
    UCHAR         IE[1];
} Q2931_IE;

/*
 * manifest constants for the AALType field in struct AAL_PARAMETERS_IE
 */
typedef enum {
    AALTYPE_5    = 5,   /* AAL 5 */
    AALTYPE_USER = 16,  /* user-defined AAL */
} AAL_TYPE;

/*
 * values used for the Mode field in struct AAL5_PARAMETERS
 */
#define AAL5_MODE_MESSAGE       0x01
#define AAL5_MODE_STREAMING     0x02
```

```
/*
 * values used for the SSCSType field in struct AAL5_PARAMETERS
 */
#define AAL5_SSCS_NULL            0x00
#define AAL5_SSCS_SSCOP_ASSURED   0x01
#define AAL5_SSCS_SSCOP_NON_ASSURED 0x02
#define AAL5_SSCS_FRAME_RELAY     0x04

typedef struct {
   ULONG ForwardMaxCPCSSDUSize;
   ULONG BackwardMaxCPCSSDUSize;
   UCHAR Mode;                /* only available in UNI 3.0 */
   UCHAR SSCSType;
} AAL5_PARAMETERS;

typedef struct {
   ULONG UserDefined;
} AALUSER_PARAMETERS;

typedef struct {
   AAL_TYPE AALType;
   union {
      AAL5_PARAMETERS    AAL5Parameters;
      AALUSER_PARAMETERS  AALUserParameters;
   } AALSpecificParameters;
} AAL_PARAMETERS_IE;

typedef struct {
   ULONG PeakCellRate_CLP0;
   ULONG PeakCellRate_CLP01;
   ULONG SustainableCellRate_CLP0;
   ULONG SustainableCellRate_CLP01;
```

```
    ULONG MaxBurstSize_CLP0;
    ULONG MaxBurstSize_CLP01;
    BOOL  Tagging;
} ATM_TD;

typedef struct {
    ATM_TD Forward;
    ATM_TD Backward;
    BOOL   BestEffort;
} ATM_TRAFFIC_DESCRIPTOR_IE;

/*
 * values used for the BearerClass field in struct ATM_BROADBAND_BEARER_CAPABILITY_IE
 */
#define BCOB_A              0x01   /* Bearer class A              */
#define BCOB_C              0x03   /* Bearer class C              */
#define BCOB_X              0x10   /* Bearer class X              */

/*
 * values used for the TrafficType field in struct ATM_BROADBAND_BEARER_CAPABILITY_IE
 */
#define TT_NOIND            0x00   /* No indication of traffic type    */
#define TT_CBR              0x04   /* Constant bit rate           */
#define TT_VBR              0x06   /* Variable bit rate           */

/*
 * values used for the TimingRequirements field in struct
 ATM_BROADBAND_BEARER_CAPABILITY_IE
 */
#define TR_NOIND            0x00   /* No timing requirement indication   */
#define TR_END_TO_END       0x01   /* End-to-end timing required         */
#define TR_NO_END_TO_END    0x02   /* End-to-end timing not required     */

/*
```

```
 * values used for the ClippingSusceptability field in struct
ATM_BROADBAND_BEARER_CAPABILITY_IE
 */
#define CLIP_NOT            0x00   /* Not susceptible to clipping      */
#define CLIP_SUS            0x20   /* Susceptible to clipping          */

/*
 * values used for the UserPlaneConnectionConfig field in
 * struct ATM_BROADBAND_BEARER_CAPABILITY_IE
 */
#define UP_P2P              0x00   /* Point-to-point connection        */
#define UP_P2MP             0x01   /* Point-to-multipoint connection   */

typedef struct {
   UCHAR BearerClass;
   UCHAR TrafficType;
   UCHAR TimingRequirements;
   UCHAR ClippingSusceptability;
   UCHAR UserPlaneConnectionConfig;
} ATM_BROADBAND_BEARER_CAPABILITY_IE;

typedef ATM_BHLI ATM_BHLI_IE;

/*
 * values used for the Layer2Mode field in struct ATM_BLLI_IE
 */
#define BLLI_L2_MODE_NORMAL     0x40
#define BLLI_L2_MODE_EXT        0x80

/*
 * values used for the Layer3Mode field in struct ATM_BLLI_IE
 */
#define BLLI_L3_MODE_NORMAL     0x40
#define BLLI_L3_MODE_EXT        0x80
```

```
/*
 * values used for the Layer3DefaultPacketSize field in struct ATM_BLLI_IE
 */
#define BLLI_L3_PACKET_16        0x04
#define BLLI_L3_PACKET_32        0x05
#define BLLI_L3_PACKET_64        0x06
#define BLLI_L3_PACKET_128       0x07
#define BLLI_L3_PACKET_256       0x08
#define BLLI_L3_PACKET_512       0x09
#define BLLI_L3_PACKET_1024      0x0A
#define BLLI_L3_PACKET_2048      0x0B
#define BLLI_L3_PACKET_4096      0x0C

typedef struct {
   DWORD Layer2Protocol;            /* User information layer 2 protocol        */
   UCHAR Layer2Mode;
   UCHAR Layer2WindowSize;
   DWORD Layer2UserSpecifiedProtocol;   /* User specified layer 2 protocol information */
   DWORD Layer3Protocol;            /* User information layer 3 protocol        */
   UCHAR Layer3Mode;
   UCHAR Layer3DefaultPacketSize;
   UCHAR Layer3PacketWindowSize;
   DWORD Layer3UserSpecifiedProtocol;   /* User specified layer 3 protocol information */
   DWORD Layer3IPI;                 /* ISO/IEC TR 9577 Initial Protocol Identifier */
   UCHAR SnapID[5];                 /* SNAP ID consisting of OUI and PID        */
} ATM_BLLI_IE;

typedef ATM_ADDRESS ATM_CALLED_PARTY_NUMBER_IE;

typedef ATM_ADDRESS ATM_CALLED_PARTY_SUBADDRESS_IE;

/*
 * values used for the Presentation_Indication field in
```

```
 * struct ATM_CALLING_PARTY_NUMBER_IE
 */
#define PI_ALLOWED              0x00
#define PI_RESTRICTED           0x40
#define PI_NUMBER_NOT_AVAILABLE  0x80

/*
 * values used for the Screening_Indicator field in
 * struct ATM_CALLING_PARTY_NUMBER_IE
 */
#define SI_USER_NOT_SCREENED    0x00
#define SI_USER_PASSED          0x01
#define SI_USER_FAILED          0x02
#define SI_NETWORK              0x03

typedef struct {
   ATM_ADDRESS ATM_Number;
   UCHAR       Presentation_Indication;
   UCHAR       Screening_Indicator;
} ATM_CALLING_PARTY_NUMBER_IE;

typedef ATM_ADDRESS ATM_CALLING_PARTY_SUBADDRESS_IE;

/*
 * values used for the Location field in struct ATM_CAUSE_IE
 */
#define CAUSE_LOC_USER              0x00
#define CAUSE_LOC_PRIVATE_LOCAL     0x01
#define CAUSE_LOC_PUBLIC_LOCAL      0x02
#define CAUSE_LOC_TRANSIT_NETWORK   0x03
#define CAUSE_LOC_PUBLIC_REMOTE     0x04
#define CAUSE_LOC_PRIVATE_REMOTE    0x05
#define CAUSE_LOC_INTERNATIONAL_NETWORK  0x06
#define CAUSE_LOC_BEYOND_INTERWORKING    0x0A
```

```
/*
 * values used for the Cause field in struct ATM_CAUSE_IE
 */
#define CAUSE_UNALLOCATED_NUMBER            0x01
#define CAUSE_NO_ROUTE_TO_TRANSIT_NETWORK   0x02
#define CAUSE_NO_ROUTE_TO_DESTINATION       0x03
#define CAUSE_VPI_VCI_UNACCEPTABLE          0x0A
#define CAUSE_NORMAL_CALL_CLEARING          0x10
#define CAUSE_USER_BUSY                     0x11
#define CAUSE_NO_USER_RESPONDING            0x12
#define CAUSE_CALL_REJECTED                 0x15
#define CAUSE_NUMBER_CHANGED                0x16
#define CAUSE_USER_REJECTS_CLIR             0x17
#define CAUSE_DESTINATION_OUT_OF_ORDER      0x1B
#define CAUSE_INVALID_NUMBER_FORMAT         0x1C
#define CAUSE_STATUS_ENQUIRY_RESPONSE       0x1E
#define CAUSE_NORMAL_UNSPECIFIED            0x1F
#define CAUSE_VPI_VCI_UNAVAILABLE           0x23
#define CAUSE_NETWORK_OUT_OF_ORDER          0x26
#define CAUSE_TEMPORARY_FAILURE             0x29
#define CAUSE_ACCESS_INFORMAION_DISCARDED   0x2B
#define CAUSE_NO_VPI_VCI_AVAILABLE          0x2D
#define CAUSE_RESOURCE_UNAVAILABLE          0x2F
#define CAUSE_QOS_UNAVAILABLE               0x31
#define CAUSE_USER_CELL_RATE_UNAVAILABLE    0x33
#define CAUSE_BEARER_CAPABILITY_UNAUTHORIZED  0x39
#define CAUSE_BEARER_CAPABILITY_UNAVAILABLE   0x3A
#define CAUSE_OPTION_UNAVAILABLE            0x3F
#define CAUSE_BEARER_CAPABILITY_UNIMPLEMENTED 0x41
#define CAUSE_UNSUPPORTED_TRAFFIC_PARAMETERS  0x49
#define CAUSE_INVALID_CALL_REFERENCE        0x51
#define CAUSE_CHANNEL_NONEXISTENT           0x52
#define CAUSE_INCOMPATIBLE_DESTINATION      0x58
```

```
#define CAUSE_INVALID_ENDPOINT_REFERENCE      0x59
#define CAUSE_INVALID_TRANSIT_NETWORK_SELECTION 0x5B
#define CAUSE_TOO_MANY_PENDING_ADD_PARTY      0x5C
#define CAUSE_AAL_PARAMETERS_UNSUPPORTED      0x5D
#define CAUSE_MANDATORY_IE_MISSING            0x60
#define CAUSE_UNIMPLEMENTED_MESSAGE_TYPE      0x61
#define CAUSE_UNIMPLEMENTED_IE                0x63
#define CAUSE_INVALID_IE_CONTENTS             0x64
#define CAUSE_INVALID_STATE_FOR_MESSAGE       0x65
#define CAUSE_RECOVERY_ON_TIMEOUT             0x66
#define CAUSE_INCORRECT_MESSAGE_LENGTH        0x68
#define CAUSE_PROTOCOL_ERROR                  0x6F

/*
 * values used for the Condition portion of the Diagnostics field
 * in struct ATM_CAUSE_IE, for certain Cause values
 */
#define CAUSE_COND_UNKNOWN                    0x00
#define CAUSE_COND_PERMANENT                  0x01
#define CAUSE_COND_TRANSIENT                  0x02

/*
 * values used for the Rejection Reason portion of the Diagnostics field
 * in struct ATM_CAUSE_IE, for certain Cause values
 */
#define CAUSE_REASON_USER                     0x00
#define CAUSE_REASON_IE_MISSING               0x04
#define CAUSE_REASON_IE_INSUFFICIENT          0x08

/*
 * values used for the P-U flag of the Diagnostics field
 * in struct ATM_CAUSE_IE, for certain Cause values
 */
#define CAUSE_PU_PROVIDER                     0x00
```

```
#define CAUSE_PU_USER                0x08

/*
 * values used for the N-A flag of the Diagnostics field
 * in struct ATM_CAUSE_IE, for certain Cause values
 */
#define CAUSE_NA_NORMAL              0x00
#define CAUSE_NA_ABNORMAL            0x04

typedef struct {
   UCHAR Location;
   UCHAR Cause;
   UCHAR DiagnosticsLength;
   UCHAR Diagnostics[4];
} ATM_CAUSE_IE;

/*
 * values used for the QOSClassForward and QOSClassBackward
 * field in struct ATM_QOS_CLASS_IE
 */
#define QOS_CLASS0           0x00
#define QOS_CLASS1           0x01
#define QOS_CLASS2           0x02
#define QOS_CLASS3           0x03
#define QOS_CLASS4           0x04

typedef struct {
   UCHAR QOSClassForward;
   UCHAR QOSClassBackward;
} ATM_QOS_CLASS_IE;

/*
 * values used for the TypeOfNetworkId field in struct
ATM_TRANSIT_NETWORK_SELECTION_IE
```

```
*/
#define TNS_TYPE_NATIONAL          0x40

/*
 *  values used for the NetworkIdPlan field in struct ATM_TRANSIT_NETWORK_SELECTION_IE
 */
#define TNS_PLAN_CARRIER_ID_CODE    0x01

typedef struct {
    UCHAR TypeOfNetworkId;
    UCHAR NetworkIdPlan;
    UCHAR NetworkIdLength;
    UCHAR NetworkId[1];
} ATM_TRANSIT_NETWORK_SELECTION_IE;

#pragma pack()

#endif  /* _WS2ATM_H_ */
```

บรรณานุกรม

1. Lewis Napper , " WinSock 2.0 " , Developing Robust Network Application for Windows, IDG Books Worldwide Inc., 1997, 543p

2. George C. Sackett ,Christopher Y.Melz , " ATM and Multiprotocol Networking ", McGraw – Hill , 1996, 341p

3. "Windows* Sockets2 Application Programming Interface" ,Revision 2.2.0 May 10, 1996, 268p

4. สานนท์ เคลือบกำเหนิด, ณัฐวุฒิ สุขเจริญนุกุล , " คู่มือการใช้ Microsoft Visual C++ " ซีเอ็ด , 1994, 678p

5. Microsoft Developer Network, "MSDN Library Visual Studio 6.0 " CD-ROM จำนวน 2 แผ่น

6. เอกสารของ DirectShow ในชุดพัฒนาโปรแกรมบนวินโดว์ส "Microsoft DirectX Media SDK 6.0 "