การควบคุมคอมพิวเตอร์ระยะไกลผ่าน TCP/IP

นายจรัญ คาร์น 38054110

นางสาวนิตยา จินดาทองประภา 38054128

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

REMOTE COMPUTER CONTROL OVER TCP/IP

Mr. Jarun	Khan	38054110
Miss Nittaya	Jindathongprapa	38054128

A Special Project Submitted in Partial Fulfillment of the

Requirement for the Bachelor Degree of Science

Department of Mathematics and Computer Science

Faculty of Science

King Mongkut Institute of Technology Chaokhuntaharn Ladkrabang

1998

ปัญหาพิเศษ	การควบคุมคอมพิวเตอร์ระยะไกลผ่าน TCP/IP	
โดย	นายจรัญ คาร์น	38054110
	นางสาวนิตยา จินดาทองประภา	38054128
อาจารย์ที่ปรึกษา	อาจารย์ไพรบูลย์ พันธรักษ์พงษ์	
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	
ปีการศึกษา	2541	

.....

(รองศาสตราจารย์ภัคคินี ชิตสกุล)

หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

คณะกรรมการโครงงานพิเศษ

.....

(อาจารย์กรกช ประชุมรักษ์)

ประธานกรรมการ

.....

(อาจารย์ธีรวัฒน์ ประกอบผล)

กรรมการ

.....

(อาจารย์ไพรบูลย์ พันธรักษ์พงษ์)

กรรมการและอาจารย์ที่ปรึกษา

ปัญหาพิเศษ การควบคุมคอมพิวเตอร์ระยะไกลผ่าน TCP/IP

โดย นายจรัญ คาร์น 38054110
นางสาวนิตยา จินดาทองประภา 38054128

อาจารย์ที่ปรึกษา อาจารย์ไพรบูลย์ พันธรักษ์พงษ์

ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์

ปีการศึกษา 2541

บทคัดย่อ

โครงงานพิเศษเรื่องการควบคุมคอมพิวเตอร์ระยะไกลผ่าน TCP/IP นี้เป็นการพัฒนาโปรแกรมขึ้นมา 2โปรแกรม คือโปรแกรมฝั่งควบคุม(Master)และโปรแกรมฝั่งถูกควบคุม (Puppet) โดยทั้งสองโปรแกรมนี้จะใช้ในการติดต่อสื่อสารกันระหว่างเครื่องคอมพิวเตอร์ 2 เครื่องที่มีการเชื่อมต่อกันผ่านโปรโตคอล TCP/IP ซึ่งซอฟแวร์ที่ใช้ในการพัฒนาโปรแกรมนี้คือ Visual C++ เวอร์ชั่น 6.0 โปรแกรมฝั่งถูกควบคุม ที่สร้างขึ้นมานั้นจะทำการส่งหน้าจอมาให้กับโปรแกรมฝั่งควบคุมให้คอยตรวจจับการเปลี่ยนแปลงของหน้าจอ และเปลี่ยนแปลงหน้าจอของเครื่องคอมพิวเตอร์ทั้งสองให้เหมือนกัน ส่วนโปรแกรมฝั่งควบคุมจะคอยตรวจจับการทำงานของเมาส์และคีย์บอร์ด และจะส่งการทำงานของเมาส์และคีย์บอร์ดเพื่อไปทำให้เกิดขึ้นบนเครื่องที่ถูกควบคุม

โครงงานพิเศษนี้จะช่วยให้เราสามารถควบคุมเครื่องคอมพิวเตอร์จากระยะไกลโดยผ่านโปรโตคอล TCP/IP ได้ ทำให้ประหยัดเวลาในการเดินทาง และค่าใช้จ่ายในการจ้างผู้ชำนาญหลายๆ คน นอกจากนี้ยังคาดว่าจะนำมาประยุกต์ในการสอนคอมพิวเตอร์ในห้องปฏิบัติการอีกด้วย

Special Topic	Remote Computer Control Over TCP/IP		
Students	Jarun	Khan	38054110
	Nittaya	Jindathongprapa	38054128
Advisor	Mr. Praiboon	Pantarakpong	
Department	Mathemetics and Computer Science		
Year	1998		

Abstract

The special project, "Remote Computer Control Over TCP/IP Links" is developing 2 application programs, the Master's application (Run on expert's PC) and the Puppet's application (Run on user's PC). Which these 2 applications must be run on Microsoft window operating systems, And it'll communicate with each other via TCP/IP protocol. These application programs are developed with Microsoft Visual C++ 6.0.

The communication between these 2 applications is when we catch clicked or keyboard pressed in the working area of the Master's application; we send it to the Puppet's application and make it happen on the Puppet's computer. Another main communication; since the event happens on the Puppet's computer. There might be some change of desktop happen. Then it must send that updated desktop to the Master's application to update its working area.

This special project will help we control the computer remotely via TCP/IP protocol. So it makes us save times and we don't have to have a large number of customer supports for servicing your clients.

กิตติกรรมประกาศ

ขอขอบคุณคุณพ่อ คุณแม่ ที่มอบชีวิต มอบกำลังใจ มอบความหวัง และมอบวันนี้ให้พวกเรา คง ไม่มีสิ่งใดทดแทนคุณของพ่อและแม่ได้ แต่สิ่งนี้ คือสิ่งที่พวกเราตั้งใจ มันคือส่วนหนึ่งของความสำเร็จที่พวกเราขอมอบแด่ท่าน

ขอขอบคุณอาจารย์นันทิกา เบญจเทพานันท์ ที่ให้คอยดูแล เป็นห่วงเป็นใย ให้คำปรึกษาและให้ข้อคิดกับพวกเรามาโดยตลอดระยะเวลา 4 ปี

ขอขอบคุณอาจารย์ไพรบูลย์ พันธรักษ์พงษ์ ที่ช่วยเสนอหัวข้อโครงงานนี้ พร้อมทั้งให้แนวคิด คำแนะนำ รวมถึงคำปรึกษาดีๆในการทำโครงงานนี้

และทุกๆท่านที่เป็นกำลังใจ คอยช่วยเหลือ และห่วงใยพวกเรามาโดยตลอด ขอบคุณมากๆ

โดยเฉพาะ:

พี่นพนันท์ พานิชเจริญ	คณะวิทยาศาสตร์ สาขาวิชาสถิติประยุกต์ (รุ่น 12) สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พี่ปริญญ์ แซกวานิช	คณะวิทยาศาสตร์ สาขาวิชาคณิตศาสตร์ประยุกต์ (รุ่น 13) สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พี่มงคล คาร์น (รุ่น 5)	คณะสถาปัตยกรรมศาสตร์ สาขาวิชาสถาปัตยกรรมศาสตร์ มหาวิทยาลัยรังสิต
พี่เฉลิมชัย เตตานนท์สกุล	คณะวิทยาศาสตร์ สาขาวิชาคณิตศาสตร์ประยุกต์ (รุ่น 2) สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พี่ประยุทธ์ เตตานนท์สกุล	บริษัท Business Soft

และเพื่อนๆ พี่ๆ น้องๆ ทุกคน สำหรับสิ่งดีๆมากมายที่เรามีให้กันตลอดมา และตลอดไป

และที่ขาดไม่ได้คือ

PRO-MARK ZX-5A และ stain less drum sticks คู่ใจ และ Carlos acoustic guitar ที่ช่วยให้หายเครียด

Axl 9จ-2099 ที่ช่วยอำนวยความสะดวก รู้ใจ ไปไหนไปด้วยกันตลอด รักเธอจริงๆ

Nuno 133 ที่เจ็บท้องอยู่ 1 ปี เพื่อที่จะคลอดผลงานนี้ออกมาให้

คณะผู้จัดทำ

สารบัญ

สารบัญรูป

หน้า

หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญ/ที่มาของปัญหา

เนื่องจากปัจจุบันมีการใช้คอมพิวเตอร์กันอย่างแพร่หลาย เมื่อผู้ใช้เกิดปัญหาบางครั้งอาจไม่สามารถแก้ปัญหาได้ด้วยตัวเอง จึงต้องปรึกษาผู้เชี่ยวชาญซึ่งบางครั้งผู้เชี่ยวชาญอาจไม่สะดวกในการเดินทางมาช่วยแก้ปัญหา จึงได้พัฒนาโปรแกรมเพื่อควบคุมคอมพิวเตอร์จากระยะไกลเพื่ออำนวยความสะดวกให้กับผู้เชี่ยวชาญในการช่วยแก้ปัญหากับผู้ใช้คอมพิวเตอร์

1.2 วัตถุประสงค์ของปัญหาพิเศษ

พัฒนาโปรแกรม 2 โปรแกรม คือ โปรแกรมฝั่งควบคุม (Master) และโปรแกรมฝั่งถูกควบคุม (Puppet)โดยโปรแกรมฝั่งถูกควบคุมจะทำการส่งหน้าจอมาให้กับโปรแกรมฝั่งควบคุม โดยจะคอยตรวจจับการเปลี่ยนแปลงของหน้าจอ และทำการปรับเปลี่ยนหน้าจอของทั้งสองฝั่งให้ตรงกัน นอกจากนี้โปรแกรมฝั่งควบคุมจะคอยตรวจจับการทำงานของเมาส์ และคีย์บอร์ด และจะทำการส่งการทำงานของเมาส์ และคีย์บอร์ด เพื่อไปทำให้เกิดขึ้นบนเครื่องที่ถูกควบคุม

1.3 ขอบเขตของปัญหาพิเศษ

1. พัฒนาโปรแกรมด้วย Microsoft Visual C++ 6.0
2. ควบคุมคอมพิวเตอร์ IBM – Compatible บนวินโดวส์ 95 โดยผ่านโปรโตคอล TCP/IP ระหว่างเครื่องสองเครื่องที่ทราบหมายเลข IP แล้ว โดยทั้งสองเครื่องต้องทำการรันโปรแกรมชุดนี้พร้อมกัน
3. ไม่พิจารณาระบบรักษาความปลอดภัยอื่นใดนอกจากซอฟแวร์ที่ได้ผลิตขึ้นนี้
4. พิจารณาข้อมูลนำเข้าเฉพาะเมาส์และคีย์บอร์ด

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. นำมาประยุกต์ใช้ในการสอนคอมพิวเตอร์ในห้องปฏิบัติการ
2. ช่วยให้ผู้เชี่ยวชาญสามารถแก้ไข และปรับเปลี่ยนค่า Configuration ต่าง ๆ จากระยะไกลได้
3. ประหยัดเวลาเดินทางของผู้เชี่ยวชาญ และประหยัดงบประมาณในการจ้างผู้ชำนาญ เพราะเราสามารถควบคุมเครื่องคอมพิวเตอร์โดยผ่าน TCP/IP ไม่ต้องเดินทางไปแก้ปัญหาถึง สถานที่จริง

1.5 ขั้นตอนการดำเนินงาน

<u>ภาคเรียนที่ 1</u>

1 ก.ค. - 10 ก.ค.	ศึกษาและรวบรวมข้อมูลด้านการส่งข้อมูลผ่าน TCP/IP
11 ก.ค. - 25 ก.ค.	ศึกษาและรวบรวมข้อมูลด้านการควบคุมการส่งและการรับข้อมูลข่าวสาร(message)ผ่าน Network
26 ก.ค. - 8 ส.ค.	ศึกษาขั้นตอนการส่ง message ของวินโดวส์
9 ส.ค. - 15ส.ค.	เขียน Dataflow Diagram และ Model การทำงาน
16 ส.ค. - 16 ก.ย.	ศึกษาการเขียนโปรแกรมด้วย Visual C++
17 ก.ย. - 30 ก.ย.	เขียนโปรแกรมในส่วนการติดต่อกันโดยใช้ TCP/IP
1 ต.ค. - 6 ต.ค.	จัดทำ Document

<u>ภาคเรียนที่ 2</u>

8 ต.ค. - 15 พ.ย.	เขียนโปรแกรมในส่วนของการคัดลอก และการส่งหน้าจอวินโดวส์ระหว่างเครื่องสองเครื่อง
16 พ.ย. - 31 ธ.ค.	เขียนโปรแกรมในส่วนของการควบคุมคอมพิวเตอร์ระยะไกล
1 ม.ค. - 31 ม.ค.	รวบรวมส่วนต่างๆ เข้าด้วยกันและพัฒนาโปรแกรมให้สมบูรณ์
1 ก.พ. - 28ก.พ.	ทดสอบการทำงานและปรับปรุงแก้ไข
1 มี.ค. – 4 มี.ค.	จัดทำ Document

ID	Task Name	July				August					September					October				November					December				January				February				March			
		5	12	19	26	2	9	16	23	30	6	13	20	27	30	4	11	18	25	1	8	15	22	29	6	13	20	27	3	10	17	24	31	7	14	21	28	7	14	21	28
1	ศึกษาและรวบรวมข้อมูลด้านการส่งข้อมูลด่าน TCP/IP																																							
2	ศึกษาและรวบรวมด้านการส่งและรับ Messageด่าน network																																							
3	ศึกษาขั้นตอนการส่ง message ของวินโดวส์																																							
4	เขียน Dataflow Diagram และ Model การทำงาน																																							
5	ศึกษาการเขียนโปรแกรมโดยใช้ Visual C++																																							
6	เขียนโปรแกรมในส่วนของการติดต่อกันโดยใช้ TCP/IP																																							
7	จัดทำ Document																																							
8	เขียนโปรแกรมในส่วนของการก๊อปออกและส่งหน้าจอวินโดวส์																																							
9	เขียนโปรแกรมในส่วนของการควบคุมคอมพิวเตอร์จากระยะไกล																																							
10	รวบรวมส่วนต่างๆ เข้าด้วยกันและพัฒนาโปรแกรมให้สมบูรณ์																																							
11	ทำการทดสอบและปรับปรุงแก้ไข																																							
12	จัดทำ Document																																							

1.6 อุปกรณ์ที่ใช้ในการทำปัญหาพิเศษ

1. เครื่องคอมพิวเตอร์รุ่น Pentium ที่เชื่อมต่อกันด้วย TCP/IP อย่างน้อยสองเครื่องที่มีอยู่แล้ว
2. Network Interface Card บนเครื่องคอมพิวเตอร์ทุกเครื่อง
3. สายต่อ LAN
4. โปรแกรม Microsoft Visual C++ 6.0

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 โปรโตคอล TCP/IP

2.1.1 ประวัติความเป็นมา

สืบเนื่องมาจากแนวความคิดของกระทรวงกลาโหมของประเทศสหรัฐอเมริกาที่ต้องการเชื่อมต่อเครื่อง คอมพิวเตอร์ที่มีอยู่มากมายหลายยี่ห้อเข้าด้วยกันเป็นระบบเครือข่ายเดียว นับเป็นจุดเริ่มต้นของชุดโปรโตคอล TCP/IP เพราะมีการแสดงให้เห็นถึงการเชื่อมต่อเครื่องคอมพิวเตอร์จาก 4 แห่งเข้าหากัน โดยเรียกเครือข่ายนี้ว่า ARPANET รูปแบบการสื่อสารข้อมูล ใช้เทคนิคของการสวิตช์กลุ่มข้อมูล (packet switching)

ในปี ค.ศ. 1972 เป็นการสาธิตความสามารถของระบบเครือข่ายเป็นครั้งแรก โดยมีเครื่องคอมพิวเตอร์จำนวน 50 เครื่องต่อเข้ากับระบบที่มีอุปกรณ์สวิตช์กลุ่มข้อมูลทำหน้าที่ ส่งผ่านข้อมูลระหว่างเครื่องคอมพิวเตอร์

ในช่วงต้นทศวรรษที่ 80 ชุดโปรโตคอล TCP/IP ได้ถูกรวมเข้าเป็น ความสามารถหนึ่งของระบบปฏิบัติการยูนิกซ์ภายใต้ชื่อ UNIX BSD 4.2 เพื่อความปลอดภัยของข้อมูลทางทหารจึงได้ทำการแบ่งเครือข่าย ARPANET ออกเป็น 2 เครือข่าย คือเครือข่ายทางด้านพลเรือน และงานวิจัย ซึ่งยังคงใช้ชื่อ ARPANET ตามเดิม และเครือข่ายทางทหารใช้ชื่อ MILNET นับตั้งแต่นั้นเป็นต้นมาจนถึงระบบปัจจุบัน ระบบสื่อสารข้อมูล โดยใช้ชุด

โปรโตคอล TCP/IP ก็เป็นที่ยอมรับแก่คนทั่วไป

2.1.2 โปรโตคอล TCP

โปรโตคอล TCP (มีชื่อเต็มว่า Transport Control Protocol) เป็นโปรตรคอลที่อยู่ในชั้นที่ 4 ของ OSI Model มีการให้บริการแบบ "Connection Oriented : CO"

โปรโตคอล TCP เป็นโปรโตคอลที่ทำให้เราสามารถเชื่อถือได้ว่าข้อมูลของเราต้องมีการส่งและรับระหว่างเครื่องคอมพิวเตอร์สองตัวได้โดยมีการเรียงตามลำดับแน่นอน

คำศัพท์ที่ใช้เรียก TCP แพคเก็ต คือ เซกเมนต์ (segment) รายละเอียดเขตต่างๆ ของเซกเมนต์ TCPแสดงในรูป ซึ่งอธิบายได้ดังนี้

รูปแบบของ TCP Packet

SOURCE PORT		DESTINATION PORT
SEQUENCE NUMBER		
ACKNOWLEDGEMENT		
HL RSV	CODE BIT	WINDOW
CHECKSUM		URGENT POINTER
OPTION		
DATA		

2.1.3 ส่วนต่างๆ ของ TCP packet

1). Source port

มีขนาด 16 บิต บอกหมายเลขพอร์ตของสถานีต้นทาง

2). Destination port

มีขนาด 16 บิต บอกหมายเลขพอร์ตของสถานีปลายทาง

3). Sequence number

มีขนาด 32 บิต บอกถึงตำแหน่งไบต์แรกของข้อมูลที่ปรากฏอยู่ในส่วนของเซต data

4). Acknowledgement number

มีขนาด 32 บิต โปรโตคอล TCP ใช้หลักการของ positive ack ในการตอบกลับไปยังต้นทางเพื่อบอกว่าปลายทางได้รับข้อมูลเรียบร้อยแล้ว โดยหมายเลขของ ack.no. จะต้องมีค่าเท่ากับ " ไบต์สุดท้ายบวกหนึ่ง" ของข้อมูลที่รับได้ Header length (HL) มีขนาด 4 บิต เป็นการบอกถึงความยาวส่วนหัวของ TCP เซกเมนต์

5). Reserved (SRV)

มีขนาด 4 บิต เป็นส่วนที่ถูกสำรองไว้ใช้ในอนาคต

6). Code bit

ประกอบด้วย 6 เขตย่อย ซึ่งแต่ละเขตย่อยมีขนาด 1 บิตมีรายละเอียด ดังนี้

* เขตย่อย URGent ถ้าบิตนี้เป็น 1 หมายความว่าเขต URGent pointer บรรจุข่าวสารที่มีความหมายจะต้องนำมาตีความด้วย

* เขตย่อย ACKnowledgement ถ้าบิตนี้เป็น 1 หมายความว่าเขต ACKnowledgement number มีความหมายต้องนำมาพิจารณาในการตรวจสอบลำดับในการรับส่งข้อมูลด้วย

* เขตย่อย PUSH ถ้าบิตนี้เป็น 1 หมายความว่าทันทีที่สถานีปลายทางรับ TCP เซกเมนต์ได้ ต้องรีบส่งข้อมูลนี้ไปยังแอพพลิเคชั่นทันที

* เขตย่อย RESET ถ้าบิตนี้เป็น 1 หมายความว่ามีการผิดพลาดเกิดขึ้นระหว่างคู่สถานีที่มีการเชื่อมต่อถึงกันอยู่ และจำเป็นต้องยกเลิกการเชื่อมต่อนี้

* เขตย่อย SYNchronize ถ้าบิตนี้เป็น 1 หมายความว่าขอเริ่มต้นการสถาปนาการเชื่อมต่อ

* เขตย่อย FIN ถ้าบิตนี้เป็น 1 หมายความว่า ขอยกเลิกการเชื่อมต่อระหว่างสถานี

7). window

ขนาด 16 บิต สถานีปลายทางใช้เขตนี้ในการบอกถึงขนาดของบัพเฟอร์ (จำนวนไบต์)ที่สามารถรับข้อมูลจากต้นทางได้

8). Checksum

มีขนาด 16 บิต เป็นเขตที่ใช้ในการตรวจสอบความผิดพลาดเฉพาะส่วนหัวของ TCP เซกเมนต์

9). Urgent pointer

มีขนาด 16 บิต บอกถึงตำแหน่งของไบต์สุดท้ายภายใน TCP เซกเมนต์ที่เป็นข้อมูลเร่งด่วนซึ่ง ค่าที่ถูกบรรจุในเขตนี้จะมีความหมายก็ต่อเมื่อเขตย่อย urgent มีค่าเป็น 1

10). Option

ส่วนนี้จะมีหรือไม่ หรือมีขนาดเท่าใดขึ้นอยู่กับชนิดของเซกเมนต์

11). Padding

มีขนาด 0-3 ไบต์ใช้เป็นส่วนที่ทำให้ขนาดของ option เป็นจำนวนเท่าของ 32 บิต

2.1.4 โปรโตคอล IP

โปรโตคอล IP เป็นโปรโตคอลที่ให้บริการแก่โปรโตคอลชั้นที่ 4 (TCP/IP) ในแบบที่เรียกว่า"Connectionless : CL" นั่นก็คือไม่มีการกำหนดหรือหาเส้นทางระหว่างต้นทางกับปลายทางเอาไว้ก่อนแต่ขึ้นกับอุปกรณ์หาเส้นทาง (router) โปรโตคอล IP มีรูปแบบดังนี้

รูปแบบของ IP packet

<table>
<tr><td colspan="3">VER</td><td>HEAD</td><td>TOS</td><td colspan="4">TOTAL LENGTH</td></tr>
<tr><td colspan="5">IDENTIFICATION</td><td>D</td><td>M</td><td></td><td>FRAG.OFFSET</td></tr>
<tr><td colspan="2">TTL</td><td colspan="3">PROTOCOL</td><td colspan="4">HEADER CHECKSUM</td></tr>
<tr><td colspan="9">SOURCE IP ADDRESS</td></tr>
<tr><td colspan="9">DESTINATION IP ADDRESS</td></tr>
<tr><td colspan="9">OPTION</td></tr>
<tr><td colspan="9">DATA</td></tr>
</table>

2.1.5 ส่วนต่างๆ ของ IP packet

1). Version (ver)

มีขนาด 4 บิต แสดงถึงรุ่นของโปรโตคอล IP

2). Header (head)

มีขนาด 4 บิต บอกถึงความยาวเฉพาะส่วนหัวของ IP packet ซึ่งมีหน่วยนับเป็นจำนวนเท่าของ 4 ไบต์

3). Type of Service (TOS)

มีขนาด 8 บิต แบ่งออกเป็น 6 เขตย่อย ดังรูป

PRECEDENCE	D	T	R	C	UNUSED

* precedence มีขนาด 3 บิตใช้ในการกำหนดลำดับความสำคัญของ IP packet

* D บิต มีขนาด 1 บิต ใช้บอกถึงความต้องการ การให้บริการจากเครือข่าย ถ้าบิตนี้มีค่าเป็น 1 หมายถึงต้องการเส้นทางที่มีการหน่วงเวลาต่ำที่สุดเท่าที่ระบบจะหาให้ได้

* T บิต มีขนาด 1บิต ถ้าบิตนี้มีค่าเป็น 1 หมายถึงต้องการเส้นทางที่มีความสามารถส่งผ่านข้อมูลได้ปริมาณ มากๆ ในหนึ่งหน่วยเวลา

* R บิต มีขนาด 1 บิต ถ้าบิตนี้มีค่าเป็น 1 หมายถึงต้องการเส้นทางหรือบริการที่มีความเชื่อถือได้สูง

* C บิต มีขนาด 1 บิต ถ้าบิตนี้มีค่าเป็น 1 หมายถึงต้องการเส้นทางที่มีค่าพารามิเตอร์ของ "cost" ต่ำ

* UNUSED มีขนาด 1 บิต ยังไม่มีการนำบิตนี้มาใช้งาน

4). Total length

มีขนาด 16 บิต ใช้บอกถึงความยาวของ IP packet ทั้งหมด

5). Identification

มีขนาด 16 บิต ใช้บอกถึงหมายเลขของ IP packet

6). D บิต

มีขนาด 1 บิต ถ้าบิตนี้มีค่าเป็น 1 หมายความว่า ห้ามตัดทอน IP packet ออกเป็น packet ย่อย แต่ถ้าบิตนี้ มีค่าเป็น 0 หมายความว่าสามารถตัดทอน IP packet ออกเป็น packet ย่อยได้

7). M บิต

มีขนาด 1 บิต ถ้าบิตนี้ถูกกำหนดเป็น 1 ใน packet ใด หมายความว่าเป็น packet ย่อยที่ถูกแบ่งออกมาจากpacket ใหญ่ โดยมีหมายเลข IP packet เหมือนกับ packet ใหญ่เพื่อที่ปลายทางจะสามารถรวบรวม packet ย่อยเหล่านี้ให้กลับคืนมาเป็น packet ใหญ่ตามเดิม แต่ถ้าบิตนี้เป็น 0 หมายความว่าเป็น packet เดียวโดดๆ หรืออาจจะเป็น packet ย่อยตัวสุดท้ายของ packet ใหญ่ก็ได้

8). Fragment offset (frag.offtset)

มีขนาด 13 บิต ใช้ในการพิจารณาร่วมกับเขต identification, total length, D บิต และM บิต เพื่อเรียงลำดับและหาขนาดความยาวของ packet ที่แท้จริงที่ถูกส่งออกมาจากสถานีต้นทาง

9). Time to live (TTL)

มีขนาด 8 บิตใช้ในการกำหนดอายุของ IP packet โดยปกติอายุของ IP packet เท่ากับ 255 ทุกครั้งที่ packet ผ่านอุปราณ์หาเส้นทางค่าของTTL จะลดลงหนึ่งเสมอ ถ้าค่าของ TTL เป็น 0 ก็หมายความว่าหมดอายุขัย router ก็จะนำ packet นั้นๆ ออกจากระบบทันที

10). Protocol

มีขนาด 8 บิต ใช้บอกชนิดของโปรโตรคอลชนิด TCP มีหมายเลขเป็น 6

11). Header checksum

มีขนาด 16 บิต ใช้ในการตรวจสอบความผิดพลาด การคำนวนจะใช้วิธีการบวกเลข 16 บิต แบบ 1'scomplement เมื่อได้แล้วให้ทำการผกผันผลลัพธ์อีกครั้งหนึ่งจึงจะได้ผลลัพธ์ที่แท้จริง

12). Source/Destinatiom address

มีขนาดอย่างละ 32 บิต ซึ่งก็คือหมายเลขเครือข่าย หรือสถานีนั้นเอง (IP address)

13). Option

ส่วนของ option นี้อาจจะมีหรือไม่มีก็ได้ ถ้าจะมีขนาดเท่าใดขึ้นอยู่กับชนิดของ IP packet

14). Padding

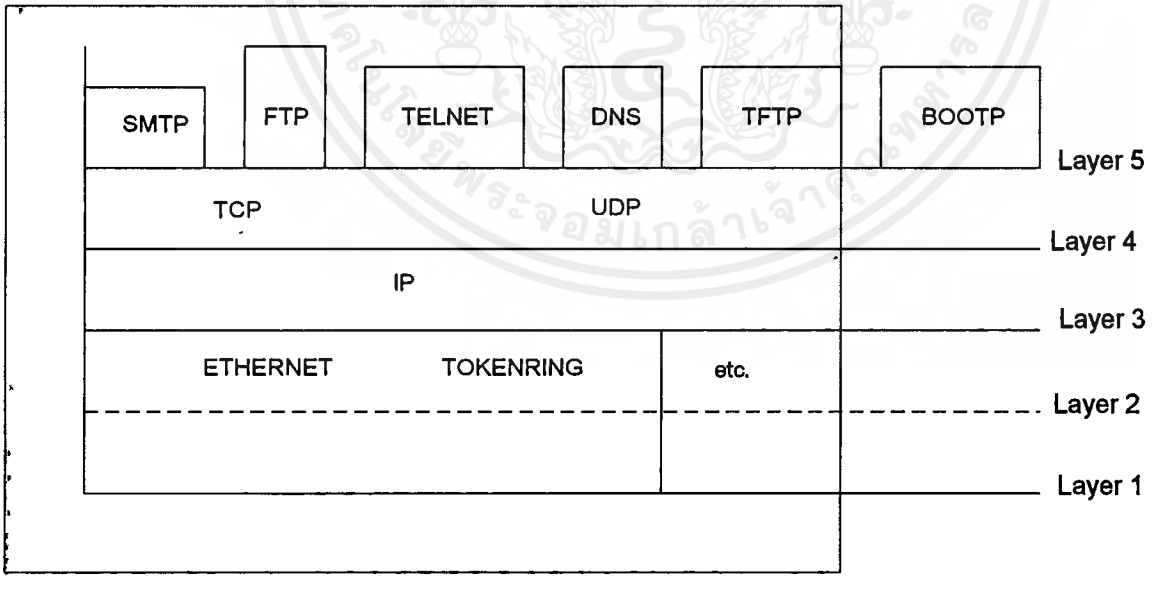
มีขนาด 0-3 บิต ใช้เป็นส่วนที่ทำให้ขนาดของ option เป็นจำนวนเท่าของ 32 บิต

15). Data

เป็นส่วนของข้อมูลหรือโปรโตคอลที่อยู่ในชั้นที่สูงกว่าหรือเท่ากับ IP packet ก็ได้

2.1.6 ความสำคัญและการทำงานร่วมกันของโปรโตคอล

จากรูปจะเห็นได้ว่าแอพพลิเคชั่นแต่ละประเภทเลือกใช้โปรโตคอลชั้นที่ 4 แตกต่างกัน และถัดจากโปรโตคอลชั้นที่ 3 ลงมาก็ขึ้นอยู่กับว่าผู้ใช้งานเลือกใช้เครือข่ายประเภทใดในการส่งผ่านชุดโปรโตคอล TCP/IP



ซึ่ง Layer ใน TCP/IP แบ่งเป็น 5 Layer คือ

1). Application Layer (Layer 5)

ที่เลเยอร์นี้จะเป็นพวกแอพพลิเคชั่นหรือโปรแกรมจริง ๆ ที่รันบนคอมพิวเตอร์ หรือที่ Enduser ที่มีการใช้ฟังก์ชันในการสื่อสารใน TCP/IP ซึ่งอาจมีการร้องขอการให้บริการจาก Layer ต่ำกว่าได้

2). Service Layer (Layer 4)

เลเยอร์นี้เป็นโปรแกรมมาตรฐานที่ให้บริการการส่งไฟล์ การติดต่อสู่โฮสต์ การส่งอีเมล์ ซึ่งเมื่อมีการร้องขอแอพพลิเคชั่นจะมีการให้บริการโดยทำการส่งแอพพลิเคชั่นนั้นไป

3). Transport Layer (Layer 3)

ที่ Service Layer จะร้องขอฟังก์ชันในการส่ง โดยข้อมูลจะถูกรวมเข้าเป็น Block เราจะเรียกว่า Message Transfer Units โดยที่ Transport Layer จะนับและแจกแจงการส่งของแต่ละ MTU และจะทำการเชื่อมต่อและทำการส่งข้อมูลให้กับเครื่องคอมพิวเตอร์ปลายทาง

4). Network Layer (Layer 2)

จะมีการส่งข้อมูลจริงและจะหาเส้นทางในการส่งข้อมูลโดยจะทำการติดต่อสื่อสารกันระหว่างโหนดในเนตเวิร์ค หรือระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง

5). Link Layer (Layer 1)

เป็นการติดต่อสื่อสารจริงระหว่าง circuit หรือระหว่าง LAN , MODEM ซึ่งเป็นตัวส่งข้อมูล ตัวอย่างที่ใช้ใน Link Layer นี้คือ Ethernet 802.3 เป็นต้น

2.1.7 บทสรุปของ TCP/IP

ถูกพัฒนาครั้งแรกเมื่อปี พ.ศ. 2516 เพื่อใช้ในเครือข่าย ARPANET เดิมได้รับการออกแบบสำหรับต่อเข้ากับเครือข่ายต่างๆ และเป็นโปรโตคอลอันหนึ่งในโปรโตคอลเก่าแก่สำหรับเครือข่าย WAN โปรโตคอลชนิดนี้ช่วยให้เครือข่ายสามารถ remote log-in, ส่งถ่ายไฟล์และรับส่งเมล์อิเล็กทรอนิกส์ได้ และมักพบ โปรโตคอลนี้ใช้งานร่วมกับอินเทอร์เน็ตเครือข่ายที่ใช้ TCP/IP จะทำการส่งถ่ายข่าวสารข้อมูลด้วยการแตกมันออกเป็นแพคเก็ตย่อยๆ ซึ่งสามารถส่งข่าวสารขนาดเท่าใดก็ได้

2.2 Windows Sockets

Window Socket คือ ส่วนติดต่อกับเน็ตเวิร์ค เสมือนเป็นเส้นทางผ่านระหว่างไมโครซอฟท์วินโดว์เน็ตเวิร์ค และช่วยให้แอพพลิเคชั่น สามารถติดต่อกันผ่านเน็ตเวิร์คได้ ซึ่งมีเน็ตเวิร์คแอพพลิเคชั่นมากมายที่ สนับสนุน Window Sockets ภายใต้ โปรโตคอลต่างๆ เช่น Transmission Control Protocol / Internet Protocol (TCP/IP), Xerox Network System(XNS), Digital Equipment Corporation's DECNet protocol, Internet Packet Exchange/Sequence Packet Exchange (IPX/SPX), ฯลฯ แต่โดยพื้นฐานแล้วจะต้องสนับสนุนTCP/IP หรือ โปรโตคอลใดๆที่สามารถทำงานร่วมกับ Window Sockets ได้โดยการ implement ไว้ใน DLL(Dynamic Link Library) ซึ่ง Socket จะช่วยให้โปรแกรมเมอร์ไม่จำเป็นต้องทราบว่าเน็ตเวิร์คทำงานอย่างไร เพราะ Sockets ได้จัดการส่วนนั้นให้หมดแล้ว

Microsoft Foundation Class Library (MFC) สนับสนุนการโปรแกรมกับ Window Sockets API โดยมี 2 class ให้ใช้ หนึ่งในนั้นคือ class CSocket มันคอยจัดการเกี่ยวกับการเขียนโปรแกรมการสื่อสารข้อมูลบนเน็ตเวิร์คให้

2.2.1 นิยามของ Socket

Socket คือปลายทางของการสื่อสารข้อมูล Socket ต้องมี type และต้องรวมอยู่ใน process ที่กำลังทำงานอยู่ มันอาจชื่อที่ใช้อ้างถึงมัน โดยทั่วไป Socket จะสื่อสารข้อมูลกับ Socket ตัวอื่นๆ (ที่กำลังทำงานอยู่) ใน Communication Domain เดียวกันโดยใช้ IP ช่วย Socket สามารถแบ่งชนิดได้เป็น 2 ชนิดคือ

1) **Socket แบบ Stream.**
   - ทำงานกับข้อมูลที่จะส่งไปโดยไม่กำหนดขอบเขตของเรคอร์ด หรือไม่เป็นเรคอร์ด (ข้อมูลที่ส่งเรียกว่า a stream of bytes) และ stream จะรับประกันว่าข้อมูลถูกส่งถึงปลายทางอย่างถูกต้องโดยข้อมูลไม่สลับกันและไม่มีการส่งซ้ำซ้อน
2) **Socket แบบ Datagram**
   - สนับสนุนการส่งข้อมูลเป็นเรคอร์ดแต่ไม่รับประกันว่าได้ส่งไปถึงและข้อมูลอาจจะมีการสลับกันหรืออาจมีการส่งข้อมูลซ้ำกันได้

หมายเหตุ ในบางเน็ตเวิร์ค เช่น XNS. Stream อาจสนับสนุนการส่งข้อมูลเป็นเรคอร์ด (Stream of record)

2.2.2 The Socket Data Type

Socket ของ MFC จะซ่อนตัว handle ของ Window Sockets ไว้ในแฮนเดิลที่ชื่อว่า SOCKET เหมือนกับที่ใช้กับ HWND ซึ่งรายละเอียดของ SOCKET จะมีอยู่ในข้อมูลของ Win32 SDK ทั่วไป

1. การใช้ Sockets

Sockets มักถูกใช้ในการสื่อสารข้อมูล 3 ลักษณะนี้

* Client/Server models
* Peer-to-Peer scenarios (เช่น Chat Application ต่างๆ)
* การทำ Remote Procedure Call (RPC) โดยให้แอพพลิเคชั่นที่รับข้อมูลแปลเมสเสจเป็นการเรียกใช้ฟังก์ชั่น ต่างๆ

2.2.3 Stream Sockets

Stream sockets ใช้ในการส่งข้อมูลในลักษณะที่ไม่เป็นเรคอร์ดซึ่ง stream of bytes นี้สามารถส่งได้ทั้งสองทาง(ส่งและรับใน socket เดียวกัน) stream นี้สามารถเชื่อถือได้ว่าจะถูกส่งอย่างถูกต้อง ไม่สลับกันและไม่ซ้ำ และส่งไปถึงอย่างแน่นอน โดย stream ยังสามารถจัดการส่งและรับข้อมูลขนาดใหญ่ได้ดีอีกด้วยเพราะโปรโตคอลในเน็ตเวิร์คเลเยอร์อาจแตกข้อมูลให้อยู่ในขนาดที่สมควรซึ่ง class CSocket จะจัดการแตกและรวมให้เอง

การสื่อสารข้อมูลแบบ Stream นั้นเป็นการสื่อสารข้อมูลแบบ Connection-Oriented. Stream นั้นจะต้องมีการเชื่อมต่อกันอย่างชัดเจน ถ้า Socket A ต้องการเชื่อมต่อกับ Socket B แล้ว Socket A จะต้องมีการร้องขอการเชื่อมต่อ ไปยัง Socket B แล้ว Socket B จะต้องยอมรับหรือปฏิเสธการร้องขอนั้นๆ เหมือนกับการโทรศัพท์ : ซึ่งจะต้องมีผู้เรียกแล้วผู้รับจะต้องรับสายจึงจะคุยกันได้ และการสนทนาจะไม่มีเสียงซ้ำกัน(ถ้าผู้พูดไม่ได้พูดซ้ำและลำดับของเสียงที่พูดจะตรงกับลำดับของเสียงที่ผู้ฟังได้ยินและที่สำคัญที่สุดคือเสียงที่ผู้พูดพูดไปนั้นจะถูกส่งไปถึงผู้รับแน่นอน ด้วยเหตุนี้จึงทำให้ stream socket เป็นที่นิยมกว่า datagram socket.

2.2.4 Datagram Sockets

Datagram sockets สนับสนุนการส่งข้อมูลสองทาง โดยไม่รับประกันว่าข้อมูลจะถูกส่งไปถึงอย่างถูกต้องหรือไม่และข้อมูลอาจมีการสลับกันหรือซ้ำซ้อนแต่ขนาดของเรคอร์ดจะได้รับการปกป้องรักษา แม้ว่าขนาดของเรคอร์ดจะใหญ่กว่าขนาดที่ผู้รับกำหนด

การสื่อสารข้อมูลแบบ datagram เป็นการสื่อสารข้อมูลแบบ connectionless ไม่จำเป็นต้องสร้างการเชื่อมต่อโดยชัดเจน เพียงส่ง datagram message ไปยัง socket ที่กำหนดก็จะสามารถสื่อสารข้อมูลกันได้แล้ว และ Datagram Socket ยังมีความสามารถในการส่งข้อมูลแบบกระจาย (Broadcasting message)

2.2.5 การใช้ Socket กับ Archive

ในส่วนนี้จะอธิบายเกี่ยวกับการลักษณะการเขียนโปรแกรมด้วย class CSocket ซึ่ง class CSocket นี้สนับสนุนการทำงานในระดับที่สูงกว่า class CAsyncSocket. Class CSocket ได้ใช้ระบบการ Serialization ของ MFC ในการส่งข้อมูลระหว่าง Object CSocket ผ่าน class CArchive

2.2.6 The CSocket Programming Model

การใช้ CSocket คือการสร้างและเชื่อมต่อ object ของ MFC เข้าด้วยกัน ตามขั้นตอนต่อไปนี้

1) สร้าง object CSocket
2) ใช้ object นี้สร้าง handle SOCKET

สำหรับตัว client โดยทั่วไปจะใช้ default parameter ในการสร้าง และสำหรับ object CSocket ที่เป็น server จะต้องกำหนดเลข port ให้ด้วย

โดยทั่วไปจะมองว่าตัว socket ที่เป็น server จะคอย Listen และตัว Client ไป connect กับ Server

3) ถ้า socket เป็น server ให้เรียก CAsyncSocket::Listen() เพื่อเริ่มการรอรับการเชื่อมต่อ และเมื่อมีร้องขอการเชื่อมต่อเข้ามาให้เรียก CAsyncSocket::Accept() เพื่อรับการร้องขอนั้นๆ

- หรือ –

ถ้า socket นั้นเป็น client ให้เรียก CAsyncSocket::Connect() เพื่อทำการเชื่อมต่อเข้ากับ socket ที่เป็น server

4) สร้าง object CSocketFile แล้วเชื่อมเข้ากับ object CSocket
5) สร้าง object CArchive สำหรับการ load (receiving) หรือ save (sending) แล้วเชื่อมเข้า object CSocket โดยที่ CArchive ไม่สามารถใช้กับ datagram sockets ได้
6) ใช้ object CArchive เพื่อส่งหรือรับข้อมูลระหว่าง socket ที่เชื่อมกัน
7) ทำลาย object CArchive, CSocketFile, CSocket เมื่อเสร็จสิ้นการใช้

2.2.7 Sequence of Operations

ในส่วนนี้จะกล่าวถึงลำดับขั้นตอนการสร้างและสื่อสารข้อมูลระหว่าง socket ทั้งด้าน server และ client

	Server	Client
สร้าง socket.	Csocket sockServer; SockServer.Create(nPort); *// *nPort เป็น UINT ที่ใช้กำหนดเลข port ที่จะ // มา connect ด้วย*	CSocket sockClient; SockClient.Create();
หา connection.		SockClient.Connect(strAddress, nPort); *// *strAddress คือ String ที่มีค่า IP Address อยู่* **nPort เป็น UINT ที่ได้กำหนดไว้ตอนที่ server ได้ // Create ขึ้น*
สร้าง socket ว่างๆ และ Accept connection.	Csocket sockReceive; SockServer.Accept(sockReceive);	
สร้าง file object.	CsocketFile file(&sockServer);	
สร้าง archive	Carchive arIn(&file, CArchive::load); - หรือ – Carchive arOut(&file, CArchive::store); Carchive arIn(&file, CArchive::load); - หรือ – Carchive arOut(&file, CArchive::store);	
ใช้ archive ในการส่งหรือรับข้อมูล	ArIn << dwValue; - หรือ – arOut >> dwValue;	

2.2.8 ตัวอย่างการใช้ Sockets กับ Archives

ในส่วนนี้จะแสดงตัวอย่างการ Serialize ข้อมูลผ่าน socket เข้าสู่ object CArchive.

```
void CBlabberView::PacketSerialize(long nPackets, CArchive& arData, CArchive& arAck)
{
  if (arData.IsStoring())
  {
    CString strText;

    for(int p = 0; p < nPackets; p++)
    {
      BYTE bValue = (BYTE)(rand()%256);
      WORD nCopies = (WORD)(rand()%32000);

      // send header information
      arData << bValue << nCopies;
      for(int c = 0; c < nCopies; c++)
      {
        // send data
        arData << bValue;
      }

      Text.Format("Received Packet %d of %d
        (Value=%d,Copies=%d)",p,nPackets,(int)bValue,nCopies);

      // send receipt string
      arData << strText;
      arData.Flush();

      // receive acknowledgment
      arAck >> strText;
```

```
        // display it
        DisplayMessage(strText);
      }
    }
    else
    {
      CString strText;
      BYTE bCheck;
      WORD nCopies;

      for(int p = 0; p < nPackets; p++)
      {
        // receive header information
        arData >> bCheck >> nCopies;
        for(int c = 0; c < nCopies; c++)
        {
          // receive data
          arData >> bValue;
          if (nCheck != bValue)
            AfxMessageBox("Packet Failure");
        }
      }

      // receive receipt string and display it
      arData >> strText;
      DisplayMessage(strText);

      Text.Format("Sent Packet %d of %d
        (Value=%d,Copies=%d)",p,nPackets,(int)bValue,nCopies);

      // send acknowledgment
```

```
    arAck << strText;
    arAck.Flush();
  }
}
```

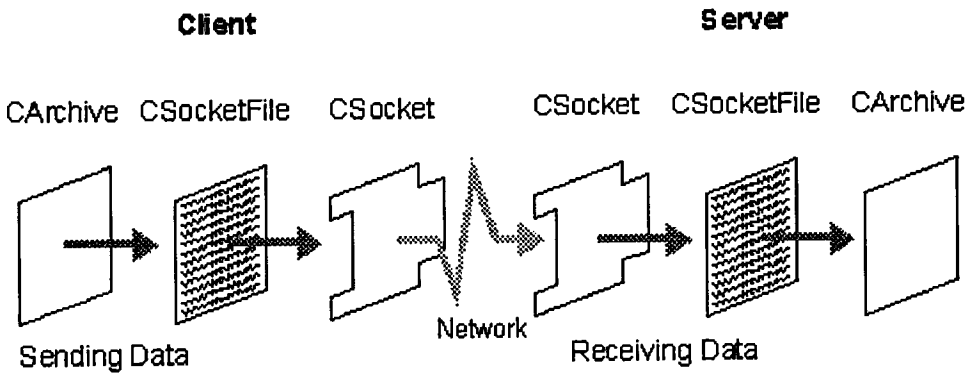
หมายเหตุ : Object CArchive arAck จะทำงานตรงข้ามกับ arData ถ้า arData ใช้ในการส่ง arAck จะใช้ในการรับ

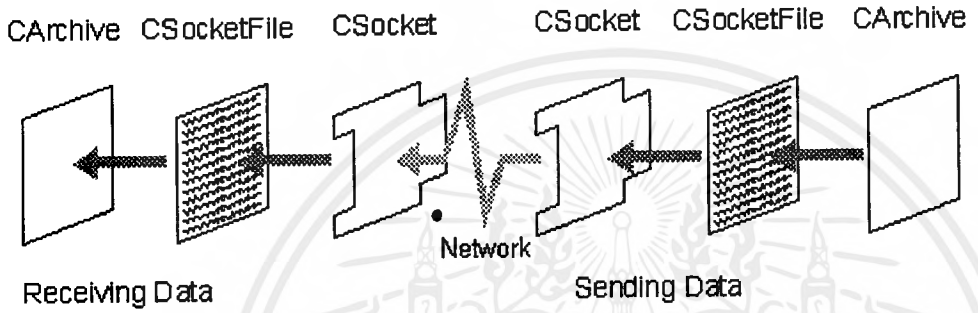
2.2.9 Sockets ทำงานกับ Archive อย่างไร

จากตัวอย่าง PacketSerialize เป็นการส่ง object ที่เหมือนกับการ Serialize ใน MFC ทั่วไป จุดแตกต่างกันก็คือข้อมูลไม่ได้ถูกเก็บไว้ใน object CFile หรือ disk ทั่วไปแต่เป็น object CSocketFile ที่เชื่อมกับ socket อีกที

Class CSocketFile สืบทอดคุณสมบัติมาจาก class CFile แต่ไม่สนับสนุนความสามารถบางอย่างของ CFile (เช่น Seek, GetLenght, SetLenght, ฯลฯ) การเก็บข้อมูลลง CSocketFile จะต้องเก็บเป็น sequence แต่เนื่องจากการที่ CSocketFile สืบทอดคุณสมบัติมาจาก CFile มันจึงต้องรับคุณสมบัติของ CFile มาทั้งหมด, เพื่อป้องกันปัญหาเหล่านี้ function ของ CFile ที่ CSocketFile ไม่สนับสนุนจะถูก override แล้ว throw exception ที่ชื่อว่า CNotSupportedException กลับมาแทน

CArchive, CSocketFile, Socket





การทำงานของมันช่วยให้ไม่ต้องจัดการรายละเอียดของ socket ด้วยตัวเอง เพียงสร้าง socket, file และ archive ขึ้นมาแล้วจึงส่ง/รับข้อมูลผ่าน archive.

โดยปกติ object CSocket เป็น two-state object คือมีสองสถานะ คือ asynchronous (สถานะปกติ) และ synchronous เมื่อ socket อยู่ในสถานะ asynchronous มันสามารถรับสัญญาณ asynchronous จาก framework แต่ในขณะที่มันกำลังส่งหรือรับข้อมูลมันจะเปลี่ยนสถานะเป็น synchronous นั่นหมายความว่า socket ไม่สามารถรับสัญญาณ asynchronous ได้อีกจนกว่าการทำงานของสถานะ synchronous จะเสร็จ

```
CMySocket::OnReceive( )
{
  // ...
  ar >> str;
  // ...
}
```

ถ้า CSocket ไม่ได้สร้างเป็นแบบ two-state object มันอาจสามารถรับสัญญาณอื่นได้อีก สำหรับการทำงานกับเหตุการณ์เดียวกับที่มันกำลังทำอยู่ จากตัวอย่างมันอาจรับรับสัญญาณให้ทำ OnReceive ในขณะที่มันกำลังทำ OnReceive อยู่เช่นการรับ ค่า str จาก archive นี้อาจทำให้เกิดการ recursion ได้ ดังนั้น CSocket จึงได้ป้องกันเหตุการณ์เช่นนี้โดยใช้วิธีสลับ state

2.2.10 การใช้ Class CAsyncSocket

class CAsyncSocket ได้ซ่อนการทำงานของ Windows Sockets API ไว้(ซึ่งได้ทำไว้ใน very low level) ทำให้โปรแกรมเมอร์ที่มีความรู้ด้านการเชื่อมต่อเครือข่ายสามารถจัดการกับ event ในเน็ตเวิร์คได้สะดวกยิ่งขึ้นโดยไม่จำเป็นต้องทราบรายละเอียดของ API แต่อย่างใด

วิธีใช้ CAsyncSocket

1) สร้าง object CAsyncSocket ซึ่งมีสองส่วนคือ

   ```
   CAsyncSocket sock;
   sock.Create( );   // Use the default parameters
   ```

   - หรือ -

   ```
   CAsyncSocket* pSocket = new CAsyncSocket;
   int nPort = 27;
   pSocket->Create( nPort, SOCK_DGRAM );
   ```

2) ถ้า socket นั้นเป็น server ให้มีการ Listen ด้วยการเรียกฟังก์ชั่น CAsyncSocket::Listen แล้วเมื่อมีการร้องขอการเชื่อมต่อเข้ามาให้เรียกฟังก์ชั่น CAsyncSocket::Accept เพื่อยอมรับการร้องขอ

   - หรือ -

   ถ้า socket นั้นเป็น client ใช้ฟังก์ชั่น CAsyncSocket::Connect เพื่อไปทำการเชื่อมต่อกับ server socket ที่ได้ listen รออยู่แล้ว

3) จัดการการสื่อสารข้อมูลด้วยฟังก์ชั่นของ CasyncSocket ที่ซ่อนการทำงานของ Windows Socket API ไว้

4) ทำลาย object CAsyncSocket

2.2.11 การจัดการกับ CAsyncSocket

เมื่อ object ของ CAsyncSocket ได้ถูกสร้างขึ้น object นั้นจะจัดการกับ handle SOCKET ให้ซึ่งโปรแกรมเมอร์สามารถจัดการกับลักษณะต่าง ๆ ของ Socket เช่น

- Blocking Mode
- Byte Ordering
- Converting String

ซึ่งจะกล่าวรายละเอียดในหัวข้อถัดไป

2.2.12 การสืบทอดคุณสมบัติจาก class socket ต่างๆ

การสืบทอดความสามารถของ class ใหม่จาก class CAsyncSocket หรือ CSocket นั้นทำให้โปรแกรมเมอร์สามารถเพิ่มคุณสมบัติของมันขึ้น และใน class เหล่านี้จะมี virtual ฟังก์ชั่นที่สามารถ override ได้นั่นคือ OnReceive, OnSend, OnAccept, OnConnect, และ OnClose ซึ่งเป็นลักษณะของ callback ฟังก์ชั่นเมื่อเกิด notification ของ socket event นั้นๆ และยังมี virtual function (advance overridable) ของ CSocket ซึ่ง MMFC จะเรียกฟังก์ชั่นนี้เมื่อ socket กำลังดึงเมสเสจมาจากวินโดว์

2.2.13 Socket Notifications

ในส่วนนี้จะอธิบายเกี่ยวกับฟังก์ชั่นที่ framework เรียกกลับมาที่ socket เมื่อเกิดเหตุการณ์ต่างๆดังนี้

- ◆ OnReceive : จะ notify เมื่อมีข้อมูลเข้ามาในบัฟเฟอร์จากการที่ socket อีกด้านส่งข้อมูลนั้นมาและสามารถรับข้อมูลนั้นได้โดยการเรียกฟังก์ชั่น Receive
- ◆ OnSend : จะ notify เมื่อ socket กำลังจะเริ่มส่งข้อมูล
- ◆ OnAccept : จะ notify เมื่อมี socket อื่นมาเชื่อมต่อกับ socket ที่ได้ Listen ไว้ และสามารถเรียก ฟังก์ชั่น Accept เพื่อยอมรับการร้องขอการเชื่อมต่อนั้นได้
- ◆ OnConnect : จะ notify เมื่อ socket นี้ได้เชื่อมต่อกับ socket ที่ Listen รอสำเร็จแล้ว
- ◆ OnClose : จะ notify เมื่อการเชื่อมต่อได้ถูกปิดลง

2.2.14 Blocking

Socket สามารถเป็นแบบ “blocking mode” หรือ “nonblocking mode” ก็ได้ ฟังก์ชั่นของ socket ที่อยู่ใน blocking mode จะไม่ return ค่าใดๆจนกว่าจะทำงานของมันสำเร็จ หรือถ้าเป็นแบบ “nonblocking mode” ทุกฟังก์ชั่นจะคืนค่ากลับทันที ซึ่งการตรวจสอบค่าความผิดพลาดจะดูได้จาก

ฟังก์ชั่น GetLastError ดังนั้นถ้าหาก socket เป็น block mode GetLastError จะ คืนค่า WSAEWOULDBLOCK กลับมาเมื่อ socket ถูก block หรือ กำลัง process อยู่

.

2.2.15 Byte Ordering

ในเครื่องคอมพิวเตอร์ที่มีสถาปัตยกรรมต่างกันอาจมีการจัดเรียงลำดับ byte (Byte Ordering) ที่ต่างกัน เช่นในเครื่องตระกูล Intel จะสลับ byte สูงกับ byte ต่ำ(เรียกว่า "little-Endian") แต่เครื่องตระกูลแมคอินทอชไม่สลับ (เรียกว่า "big-Endian") สำหรับ CAsyncSocket นั้น โปรแกรมเมอร์จะต้องจัดการ byte ordering เอง แต่สำหรับ CSocket ได้จัดการส่วนนี้ไว้ให้แล้ว

2.2.16 Converting Strings

การสื่อสารระหว่างแอพพลิเคชั่นที่ใช้ขนาดของแต่ละ byte ใน string ไม่เท่ากัน (เช่น ANSI, Unicode หรือ MultiByte Character Sets (MBCS)) Programmer จะต้องแปลง string เหล่านี้ ให้สามารถสื่อสารกันได้อย่างถูกต้อง (ถ้าโปรแกรมเมอร์ใช้ class CAsyncSocket แต่สำหรับ CSocket นั้นการแปลงจะถูกแปลงผ่าน class CString)

2.2.17 พอร์ต และแอดเดรสของ Socket

1). พอร์ต

พอร์ตเป็นเลขที่ใช้กำหนดค่าให้กับ socket แต่ละตัวโดยจะไม่มีเลขที่ซ้ำกันเกิดขึ้นพร้อมกัน เลข พอร์ตจะถูกเชื่อมกับแอพพลิเคชั่นที่สนับสนุน Window Sockets ดังนั้น Socket ของแอพพลิเคชั่นจะมีค่าพอร์ตที่ไม่ซ้ำกัน ทำให้สามารถมีแอพพลิเคชั่นที่ทำงานกับ Socket ได้หลายตัวพร้อมกัน

2). Socket แอดเดรส

Socket แต่ละตัวจะติดต่อกับ Internet Protocol (IP) Address ในเน็ตเวิร์ค โดยทั่วไปแอดเดรสจะเปรียบเสมือนชื่อเครื่อง เช่น "www.kmitl.ac.th" หรือ "161.246.10.21"

2.3 การจัดการกับวินโดวส์

2.3.1 เมสเสจ

เมสเสจเป็นอินพุตหรือข้อมูลเข้าเพียงทางเดียวของแอพพลิเคชั่น วินโดวส์เป็นตัวแทนต่อทุก ๆ เหตุการณ์ทั้งหมดที่ต้องการการตอบสนอง เมสเสจเป็นตัวแปรโครงสร้างชนิดหนึ่งที่ประกอบไปด้วยส่วนค่าอ้างอิงของเมสเสจนั้น กับส่วนพารามิเตอร์ของเมสเสจ โดยที่พารามิเตอร์จะขึ้นกับเมสเสจชนิดนั้น ๆ

1. การสร้างและการประมวลผลเมสเสจ

วินโดวส์จะมีการสร้างเมสเสจขึ้นทุกครั้งที่มีอินพุตเข้ามายังระบบ ไม่ว่าจะเป็นการเลื่อนเมาส์ การกดคีย์ หรืออื่น ๆ เพื่อบอกแก่แอพพลิเคชั่นหรือตัววินโดวส์เองถึงเหตุการณ์ (อินพุต) ที่เกิดขึ้น โดยวินโดวส์จะนำเมสเสจที่สร้างขึ้นเหล่านี้ใส่เข้าไปในคิวของระบบ จากนั้นก็ *ส่งผ่าน* ไปยังคิวของแอพพลิเคชั่นที่เหมาะสมต่อไป ลักษณะของคิวของแอพพลิเคชั่นนี้เป็นแบบเข้าก่อนออกก่อน โดยที่การเข้าก็คือ วินโดวส์ดึงเมสเสจจากคิวของระบบมาใส่ให้แก่คิวของแอพพลิเคชั่น ส่วนการออกคือ แอพพลิเคชั่นเป็นตัวดึงออกไปเองโดยเรียกใช้ฟังก์ชั่น GetMessage จากกนั้นก็แจกแจง และส่งไปยังฟังก์ชั่นประจำวินโดวส์ต่าง ๆ ด้วยฟังก์ชั่น DispatchMessage

มีเมสเสจบางอันที่วินโดวส์ต้องการส่งไปยังฟังก์ชั่นประจำวินโดวส์โดยตรง แทนที่จะวางลงในคิว อาจเรียกเมสเสจนี้ว่าเป็น เมสเสจลัดคิว เมสเสจลัดคิวนี้จะเป็นเมสเสจที่มีผลกับวินโดวส์นั้น ๆ โดยเฉพาะวินโดวส์ และแอพพลิเคชั่นจะส่งเมสเสจชนิดนี้ด้วยฟังก์ชั่น SendMessage ซึ่งจะเป็นการส่งเมสเสจไปยังฟังก์ชั่นประจำวินโดวส์โดยตรง อีกทั้งฟังก์ชั่นประจำวินโดวส์นั้นยังสามารถส่งค่าการประมวลผลเมสเสจกลับมาได้อีกด้วย

ตัวอย่างของเมสเสจลัดคิวอย่างหนึ่งก็คือ การใช้ฟังก์ชั่น CreateWindow นอกจากจะเป็นการสร้างวินโดวส์แล้ว ยังเป็นการบังคับให้วินโดวส์ส่งเมสเสจ WM_CREATE ซึ่งเป็นเมสเสจลัดคิวไปยังฟังก์ชั่นประจำวินโดวส์ แล้วรอจนกว่าฟังก์ชั่นนั่นจะจัดการกับเมสเสจเสร็จ เมสเสจ WM_CREATE จะไม่ผ่านแอพพลิเคชั่นเลย

แต่ก็ไม่ใช่ว่ามีเพียงวินโดวส์เท่านั้นที่สามารถสร้างเมสเสจขึ้นมาได้ ตัวแอพพลิเคชั่นเอง ก็สามารถสร้างเมสเสจแล้วส่งไปยังคิวของตนเอง และคิวของแอพพลิเคชั่นอื่น ๆ ได้เช่นกัน

แอพพลิเคชั่นจะใช้ฟังก์ชั่น GetMessage ในลูปฟังก์ชั่น WinMain เพื่อดึงเมสเสจออกจากคิวของตัวแอพพลิเคชั่นเอง ฟังก์ชั่น GetMessage จะทำงานโดยเริ่มจากดูว่าในคิวมีเมสเสจหรือไม่ หากมีก็จะดึงเมสเสจที่อยู่แรกสุดไป แต่หากไม่มีเมสเสจอยู่ในคิว ก็จะเกิดการรอเมสเสจขึ้น และ

ปล่อยการควบคุมกลับคืนไปให้วินโดวส์ เพื่อแบ่งการควบคุมให้แอพพลิเคชั่นอื่นสามารถทำงานได้ต่อไป

เมื่อฟังก์ชั่น WiinMain ของแอพพลิเคชั่นได้รับเมสเสจจากคิวแล้ว ก็ต้องใช้เรียกฟังก์ชั่น DispatchMessage เพื่อแจกแจงเมสเสจ และการส่งเมสเสจนั้นไปให้ฟังก์ชั่นประจำวินโดวส์ต่าง ๆ การทำเช่นนี้ จะเป็นการถ่ายการควบคุมให้กับฟังก์ชั่นประจำวินโดวส์ที่จะมวลผลเมสเสจนั้น หลังจากที่ทำงานเกี่ยวข้องกับเมสเสจนี้เรียบร้อยแล้ว ก็ถ่ายการควบคุมกลับไปให้ฟังก์ชั่นหลักในแอพพลิเคชั่น เพื่อดึงเมสเสจต่อไปในคิวมาประมวลผลต่อไป

ข้อควรระวัง ฟังก์ชั่นประจำวินโดวส์ไม่ควรคาดว่าเมสเสจใดจะเข้ามาก่อน เพราะวินโดวส์สามารถส่งเมสเสจมาในลำดับอย่างไรก็ได้

ส่วนการรับข้อมูลจากคีย์บอร์ดนั้นวินโดวส์จะรับเข้ามาในลักษณ์ของรหัสของคีย์ที่กด ดังนั้นจึงต้องมีการใช้ฟังก์ชั่น TranslateMessage เพื่อแปลงรหัสของคีย์เหล่านั้นให้เป็นรหัสของตัวอักษร เมื่อแปลงได้แล้วฟังก์ชั่นก็จะส่งเมสเสจของตัวเองเข้าไปในคิว พร้อมทั้งรหัสตัวอักษรที่ผู้ใช้กดเข้ามาเป็นพารามิเตอร์ของเมสเสจนั้น

2. การแปลและตีความเมสเสจ

โดยทั่วไปแล้วจะมีการใช้ฟังก์ชั่น TranslateMessage นี้กับทุก ๆ เมสเสจที่เข้ามา โดยที่หากไม่ใช่เมสเสจที่เกี่ยวกับคีย์บอร์ดก็จะไม่มีผลอันใด

ตัวอย่างต่อไปนี้ก็จะเป็นการแสดงถึงหน้าตาของเมสเสจลูปที่อยู่ในฟังก์ชั่น WinMain ของแอพพลิเคชั่น เมสเสจลูปจะดึงเมสเสจออกจากคิวและแจกจ่ายดังนี้

```
int PASCAL WinMain(hInt,hPrevInst,lpCmdLine,ShowCmd)
HINSTANCE hInst;
HINSTANCE hPrevInst;
LPSTR lpCmdLine;
Int ShowCmd;
{
        MSG msg;
        .
```

```
        .                                                    (ต่อหน้าถัดไป)
        While(GetMessage (&msg,NULL,0,0)) {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }
        return msg.wParam;
}
```

ส่วนแอพพลิเคชั่นที่มีการใช้คีย์ทันใจ (accelerator key) ของเมนู ก็ต้องมีการสร้างตารางคีย์ทันใจที่ใช้ไว้ในไฟล์รีซอร์สจากนั้นก็ต้องมีการโหลดตารางนี้เข้ามาโดยใช้ฟังก์ชั่น LoadAccelerator โดยจะทำให้ลักษณะของลูปในการรับและส่งเมสเสจเป็นดังนี้

```
While (GetMessage ((LPMSG)&msg, (HWND)NULL, 0 ,0 )))
{
    if (TranslateAccelerator (hWindow,hAccel, ((LPMSG)&msg) == 0)
    {
        TranslateMessage((LPMSG)&msg));
        DispatchMessage((LPMSG)&msg));
    }
}
```

ที่ต้องมีการตรวจสอบว่าค่าที่ได้กลับมาจากการเรียกใช้ฟังก์ชั่น TranslateAccerator นั้นก็เพราะว่าจากเมสเสจนั้นเป็นการกดคีย์ทันใจ ฟังก์ชั่น TranslateAccerator ก็จะทำการแปลงและส่งเมสเสจที่แปลงแล้วนั้น ไปยังฟังก์ชั่นประจำวินโดวส์เอง ดังนั้นจึงไม่ต้องมาเรียกใช้ฟังก์ชั่น TranslateMessage และฟังก์ชั่น DispatchMessage อีก

3. การตรวจสอบเมสเสจ

เราใช้ฟังก์ชั่น PeekMessage เพื่อมองหาและตรวจสอบเมสเสจบางเมสเสจในระหว่างการทำงานที่กินเวลา โดยไม่ต้องการมีผลกระทบกระเทือนต่อการเป็นอยู่ของเมสเสจ เมื่อเรียกใช้ฟังก์ชั่นนี้ ถ้ามีเมสเสจในคิวฟังก์ชั่นนี้จะให้ค่าที่ไม่เป็น 0 กลับมา ทำให้สามารถประมวลเมสเสจได้โดยไม่ต้องกลับไปที่ลูปหลักของฟังก์ชั่น WinMain ประโยชน์ของฟังก์ชั่นนี้คือ เมื่อมีการทำงาน

บางอย่างที่ต้องกินเวลามาก (ใช้เวลานานกว่าจะคืนการควบคุมไปให้ฟังก์ชั่นหลัก) เช่น การอ่านเขียนดิสก์ ก็สามารถใช้ฟังก์ชั่นนี้ในการตรวจสอบว่า ขณะนี้ผู้ใช้มีการกดคีย์เพื่อยกเลิกการทำงานหรือไม่

4. <u>การส่งและส่งผ่านเมสเสจ</u>

ฟังก์ชั่น PostMessage และ SendMessage มีหน้าที่ในการส่งเมสเสจไปยังฟังก์ชั่นประจำวินโดวส์ของตัวเองหรือฟังก์ชั่นประจำวินโดวส์ของแอพพลิเคชั่นอื่น นอกจากนี้ยังมีฟังก์ชั่น PostAppMessage ที่ทำงานแบบเดียวกับฟังก์ชั่น PostMessage แต่เป็นการส่งผ่านโดยอาศัยแฮนเดิลโมดูลของแอพลิเคชั่นแทน

ฟังก์ชั่น PostMessage จะส่งเมสเสจผ่านไปทางคิวของแอพพลิเคชั่น (เราจะขอเรียกการส่งแบบนี้ว่า *การส่งผ่าน*) โดยที่ผู้ส่งจะยังไม่เสียการควบคุม และผลของเมสเสจที่ส่งไปนั้นก็จะยังไม่เกิดขึ้นจนกว่าแอพพลิเคชั่นนั้น ๆ จะดึงเมสเสจที่ส่งไปนั้นขึ้นมา ส่วน SendMessage จะส่งเมสเสจลัดคิวไปยังฟังก์ชั่นประจำวินโดว์สที่ต้องการทันที (โดยเราจะขอเรียกการส่งแบบนี้ว่า *การส่ง*) โดยที่ไม่ต้องผ่านคิวของแอพพลิเคชั่น ซึ่งการส่งเมสเสจไปแบบนี้ ผู้ที่ส่งจะสูญเสียการควบคุมไปให้ฟังก์ชั่นประจำวินโดวส์ที่ส่งเมสเสจให้ และเมื่อฟังก์ชั่นทำงานเสร็จแล้วก็จะคืนการควบคุมกลับมา ส่วนค่าที่ได้กลับมาของการเรียกใช้ SendMessage นั้นคือค่าที่ฟังก์ชั่นประจำวินโดวส์นั้นส่งค่ากลับมา แต่ถ้าเป็นค่าที่ได้กลับมาจากฟังก์ชั่น PostMessage เป็นเพียงว่าสามารถส่งเมสเสจนันไปได้สำเร็จหรือไม่

5. <u>ฟังก์ชั่นที่ใช้กับเมสเสจ</u>

ฟังก์ชั่นในกลุ่มนี้มีหน้าที่ในการอ่านเมสเสจขึ้นมาจากคิว และประมวณผลเมสเสจในวินโดว์ ฟังก์ชั่นที่ใช้กับเมสเสจก็จะมีตารางดังนี้

ฟังก์ชั่น	หน้าที่
CallWindowProc	ส่งผ่านข้อมูลของเมสเสจไปยังฟังก์ชัน
DispatchMessage	แจกแจงเมสเสจและส่งเมสเสจไปยังฟังก์ชันประจำวินโดว์ใด ๆ
GetMessage	รับเมสเสจในช่วงของเมสเสจที่กำหนดไว้
GetMessagePost	ให้ค่าตำแหน่งของเมาส์ในขณะที่มีการรับเมสเสจล่าสุด
GetMessageTime	ให้ค่าเวลาในขณะที่มีการรับเมสเสจล่าสุด
PeekMessage	ตรวจสอบว่ามีเมสเสจในคิวหรือไม่โดยไม่ได้ดึงออกมา
PostAppMessage	ส่งเมสเสจไปยังแอพพลิเคชั่น
PostMessage	ส่งเมสเสจไปยังคิวของแอพพลิเคชั่น
PostQuitMessage	ส่ง WM_QUIT ไปยังแอพพลิเคชั่น
ReplyMessage	ตอบรับเมสเสจ
SendMessage	ส่งเมสเสจไปยังวินโดว์ต่าง ๆ
TranslateAccelerator	ประมวลผลคีย์ทันใจของเมนูให้เป็นเมสเสจและส่งเมสเสจให้แก่วินโดว์
TranslateMessage	แปลงค่ารหัสของคีย์ที่กดให้เป็นรหัสตัวอักษร
WaitMessage	ส่งการควบคุมไปให้แอพพลเคชั่นอื่น
WinMain	เป็นจุดที่เริ่มเข้าไปทำงานของ วินโดว์แอพพลิเคชั่น

2.3.2 การสร้างและการจัดการวินโดว์

ส่วนนี้จะเป็นเรื่องของการสร้าง ยกเลิก เปลี่ยนแปลง หรือดึงข้อมูลจากวินโดว์ เป็นเรื่องที่เรียกได้ว่าเป็นหัวใจของการสร้างแอพพลิเคชั่นบน Windows เลยทีเดียว

1. คลาสของวินโดว์

คลาสของวินโดว์ หมายถึง ลักษณะเฉพาะที่เป็นตัวกำหนดว่าวินโดว์นั้นจะมีรูปร่างหน้าตาเป็นอย่างไร มีความสามารถ และมีพฤติกรรมเป็นอย่างไร ก่อนที่แอพพลิเคชั่นจะสร้างวินโดว์ขึ้นมาให้ผู้ใช้เห็นบนหน้าจอได้ต้องมีการสร้าง "คลาส" ของวินโดว์เสียก่อน

การสร้างก็เพียงผ่านพารามิเตอร์ที่เหมาะสมไปให้ฟังก์ชัน Register Class โดยที่ สามารถสร้างคลาสของวินโดว์ได้มากเท่าที่ต้องการ จากนั้นก็สร้างวินโดว์โดยบอกว่าวินโดว์นั้นให้มีลักษณะเป็นไปตามคลาสใด ซึ่งคลาสหนึ่ง ๆ นั้น สามารถใช้โดยกี่วินโดว์ก็ได้ และคลาสของวินโดว์ที่สร้างไว้ก็จะคงอยู่จนกว่าจะจบแอพพลิเคชั่น

แม้ว่าข้อมูลที่ต้องใส่ตอนสร้างคลาสนั้นจะมีมากมาย แต่จะมีไม่กี่ตัวที่จำเป็นจริง ๆ สำหรับการสร้างคลาส เช่น ชื่อคลาส ฟังก์ชันที่ใช้ในการรับเมสเสจมาประมวล และแฮนเดิลที่เป็นค่าคงที่ของแอพพลิเคชั่นนั้นๆ ส่วนข้อมูลตัวอื่นๆ ก็จะเป็นพวกขนาด ตำแหน่งลักษณะเคอร์เซอร์และอื่นๆ

คลาสของวินโดว์สามารถแบ่งได้ตามลักษณะการสร้างและการคงอยู่ 3 ประเภทดังนี้

*** คลาสของระบบ (System Global Class)**

คลาสชนิดนี้ Windows จะสร้างขึ้นเมื่อเริ่มทำงาน ทุกแอพพลิเคชั่นสามารถเรียกใช้ได้ทันที คลาสชนิดนี้ไม่สามารถถูกสร้างหรือยกเลิกโดยแอพพลิเคชั่นได้ ตัวอย่างของคลาสเหล่านี้ก็เช่น คอนโทรล edit หรือ กรอบรายชื่อ

*** คลาสส่วนรวมของแอพพลิเคชั่น (Application Global Class)**

แอพพลิเคชั่นหรือไลบรารีสามารถสร้างคลาสที่ถูกเรียกใช้โดยแอพพลิเคชั่นอื่น ๆ ได้โดยกำหนด CS_GLOBALCLASS ในตอนสร้างคลาส โดยที่คลาสชนิดนี้จะหายไปก็ต่อเมื่อแอพพลิเคชั่นหรือไลบรารีนั้นเลิกทำงาน

*** คลาสของแอพพลิเคชั่น (Application Local Class)**

เป็นคลาสที่พบเห็นกันมากที่สุด คือ แอพพลิเคชั่นใดสร้างก็สามารถเรียกใช้งานได้ในแอพพลิเคชั่นนั้นเท่านั้น

เมื่อมีการสร้างวินโดว์โดยระบุคลาสที่ต้องการ Windows จะไปหาในคลาสของแอพพลิเคชั่นก่อน ถ้าไม่พบก็จะไปหาในคลาสที่ใช้ได้ทุกแอพพลิเคชั่น และถ้าไม่เจออีกก็จะไปหาในคลาสของระบบ การที่ Windows มีลำดับการหาคลาสอย่างนี้ ทำให้สามารถสร้างคลาสขึ้นมาทับคลาสเดิมโดยที่ไม่กระทบถึงแอพพลิเคชั่นที่ใช้คลาสเดิมนั้นอยู่

2. <u>ส่วนประกอบของคลาสของวินโดว์</u>

ส่วนประกอบต่าง ๆ ของคลาสของวินโดว์จะเป็นตัวที่บอกลักษณะของวินโดว์นั้น ซึ่งข้อมูลเหล่านี้จะถูกเก็บไว้ในโครงสร้างแบบ WNDCLASS จากนั้นก็จะส่งผ่านเป็นพารามิเตอร์ให้กับฟังก์ชัน Register Class ซึ่งข้อมูลเหล่านี้สามารถเรียกดูภายหลังได้โดยฟังก์ชัน Get Class Info ส่วนประกอบเหล่านี้มีดังตารางต่อไปนี้

ชื่อ	จุดประสงค์
- ชื่อคลาส	เป็นชื่อของคลาสซึ่งแตกต่างกันไปในแต่ละคลาส
- แอดเดรสฟังก์ชันประจำวินโดว์	เป็นการกำหนดแอดเดรสของฟังก์ชันที่จะใช้ในการรับเมสเสจที่ส่งมายังฟังก์ชันของแอพพลิเคชั่นนั้น
- แฮนเดิลอินแสตนต์	เป็นตัวบอกถึงแอพพลิเคชั่นที่ขึ้นทะเบียนคลาสนั้น
- เคอร์เซอร์ประจำคลาส	กำหนดรูปร่างของเคอร์เซอร์ที่จะปรากฏในวินโดว์ของคลาสนั้น
- ไอคอนประจำคลาส	กำหนดไอคอน เมื่อวินโดว์นั้นถูกลดขนาด
- สีพื้นหลังของคลาส	กำหนดสีและรูปแบบของพื้นที่ใช้งาน (client area) เมื่อแสดงวินโดว์เป็นครั้งแรก
- เมนูประจำของคลาส	กำหนดเมนูของวินโดว์ที่ไม่มีการกำหนดเมนูโดยเฉพาะ
- สไตล์ของคลาส	กำหนดว่าจะทำอย่างไรเมื่อมีการย้ายวินโดว์ เปลี่ยนขนาดวินโดว์ จะทำอย่างไรเมื่อมีการดับเบิ้ลคลิกเมาส์ และอื่น ๆ
- พื้นที่คลาสพิเศษ	กำหนดขนาดของหน่วยความจำ (เป็นไบต์) ที่ Windows ควรจองไว้ต่อจากโครงสร้างของคลาส
- พื้นที่วินโดว์พิเศษ	กำหนดขนาดของหน่วยความจำที่ Windows ควรจองไว้ต่อจากโครงสร้างของวินโดว์ที่แอพพลิเคชั่นสร้างขึ้น

3. สไตล์ของคลาส

สไตล์ของคลาสเป็นการกำหนดลักษณะของวินโดว์เพิ่มเติม โดยสามารถกำหนดได้ครั้งละมากกว่าหนึ่งสไตล์โดยใช้โอเปอร์เรเตอร์ OR (|) สไตล์ของคลาสมีดังตารางดังต่อไปนี้

สไตล์	คุณสมบัติ
CS_BYTEALIGNCLIENT	จัดวางพื้นที่ใช้งานตามไบต์(แนวนอน)
CS_BYTEALIGNWINDOW	จัดวางวินโดว์ตามไบต์(แนวนอน)
CS_CLASSDC	จองคอนเท็กซ์ดิสเพลย์เพียงหนึ่งสำหรับทุกวินโดว์ที่ใช้คลาสนี้
CS_DBCLKS	ส่งเมสเสจดับเบิ้ลคลิกไปยังฟังก์ชันประจำวินโดว์
CS_GLOBALCLASS	กำหนดว่าคลาสนั้นสามารถถูกใช้ได้กับทุกแอพพลิเคชั่น
CS_HREDRAW	กำหนดว่าจะต้องมีการวาดวินโดว์ใหม่เมื่อมีการย้ายหรือเปลี่ยนแปลงขนาดที่มีผลต่อความกว้างของวินโดว์
CS_NOCLOSE	ไม่ให้ใช้คำสั่ง Close ในเมนูของระบบ
CS_OWNDC	จองคอนเท็กซ์ดิสเพลย์สำหรับทุกวินโดว์ที่ใช้คลาสนั้น

สไตล์	คุณสมบัติ
CS_PARENTDC	ให้คลาสของวินโดว์นั้นใช้คอนเท็กซ์ดิสเพลย์ของวินโดว์ parent ของวินโดว์นั้น ๆ
CS_SAVBITS	กำหนดให้มีการเก็บภาพส่วนของวินโดว์ที่ถูกบัง เพื่อใช้ในการวาดใหม่เมื่อวินโดว์ถูกย้าย การทำเช่นนี้จะไม่มีการส่ง WM_PAINT ไปยังวินโดว์ (ถ้าหน่วยความจำที่เก็บรูปภาพส่วนนั้นไม่ถูกทิ้ง)
CS_VREDRAW	กำหนดว่าจะต้องมีการวาดวินโดว์ใหม่เมื่อมีการย้ายหรือเปลี่ยนแปลงขนาดที่มีผลต่อความสูงของวินโดว์

4. <u>ฟังก์ชั่นประจำวินโดวส์</u>

ฟังก์ชั่นประจำวินโดวส์เป็นฟังก์ชั่นที่ทำหน้าที่ประมวลเมสเสจของวินโดว์ เมสเสจจะถูกส่งมาโดยวินโดว์ เมื่อมีอินพุต (เช่นเมาส์ คีย์บอร์ด หรือไทเมอร์) เข้ามา หรือไม่วินโดว์ก็ต้องการให้วินโดวส์ทำงานบางอย่าง เช่น วาดพื้นที่ใช้งานใหม่ นอกจากมีหน้าที่ในการรับอินพุตเมสเสจต่าง ๆ แล้ว ฟังก์ชั่นประจำวินโดว์ยังมีหน้าที่ในการรับข้อมูลเมื่อระบบถูกเปลี่ยนแปลงโดยแอพพลิเคชั่นอื่น ๆ เช่น มีการแก้ไขไฟล์ win.ini หรือการทำตามคำร้องขอบางอย่างเช่น การเปลี่ยนแปลงเมนูก่อนการแสดงให้ผู้ใช้เห็น หรือการรับทราบว่าขณะนี้ตัวเองได้เป็นวินโดว์แอคทีฟแล้ว รวมทั้งข้อมูลอย่างอื่น ๆ อีกมากมาย

แม้ว่าเมสเสจส่วนใหญ่จะถูกส่งมาจากวินโดว์ แต่วินโดว์ด้วยกันเองก็สามารถส่งเมสเสจถึงกันได้(หรือจากตัวเองก็ได้) ทั้งนี้ก็เพื่อจะได้ทราบความเป็นไปของวินโดว์อื่น ๆ ตามปกติ แล้วการรับเมสเสจของฟังก์ชั่นประจำวินโดว์จะเป็นไปเรื่อย ๆ จนกว่าวินโดว์นั้นจะถูกทำลาย

เนื่องจากการทำงานของแอพพลิเคชั่นนั้นขึ้นกับเมสเสจที่แต่ละวินโดว์ได้รับ ดังนั้นตัวฟังก์ชั่นประจำวินโดว์จึงเป็นส่วนที่บังคับการดำเนินไปของวินโดว์นั้น ๆ เช่น ในแอพพลิเคชั่นที่มีการเปิดไฟล์ เมื่อผู้ใช้เลือกคำสั่ง Open ฟังก์ชั่นประจำวินโดว์ก็จะเป็นตัวสั่งการว่าต่อไปให้ทำการเปิดไฟล์ตามที่ผู้ใช้ต้องการ หรือผู้ใช้มีการเลื่อนสกรอล์บาร์ฟังก์ชั่นประจำวินโดวส์ก็ต้องเลื่อนข้อมูลที่อยู่นอกวินโดว์ขึ้นมาให้ผู้ใช้ดูตามต้องการ

ส่วนเมสเสจที่เข้ามายังฟังก์ชั่นประจำวินโดวส์แต่ไม่ได้ใช้ประโยชน์อะไรนั้น ต้องมีการส่งต่อไปให้ฟังก์ชั่น DefWindowProc เป็นตัวประมวลผลเมสเสจเหล่านั้น โดยเฉพาะอย่างยิ่งเมสเสจที่เกิดจากนอกพื้นที่ใช้งานวินโดว์ (ซึ่งเป็นงานของวินโดว์ เช่น การคลิกที่ไตเติ้ลบาร์ หรือการลดขนาดวินโดว์) ดังนั้นใน switch case ของทุกฟังก์ชั่นประจำวินโดว์ควรมี DefWindowProc นี้อยู่เสมอ

5. เมสเสจของวินโดวส์

เมสเสจเป็นกลุ่มของข้อมูลอย่างหนึ่งที่ถูกส่งไปยังฟังก์ชั่นประจำวินโดว์ต่าง ๆ ที่ทำงานในวินโดว์ขณะนั้นประกอบด้วย 4 ส่งคือแฮนเดิลของวินโดว์ที่ต้องการส่งไป ตัวเมสเสจIDพารามิเตอร์ขนาด 16 บิต และสุดท้ายคือพารามิเตอร์ขนาด 32 บิต สำหรับพารามิเตอร์สองตัวหลังนั้นจะเป็นข้อมูลของอะไรก็จะขึ้นกับชนิดของเมสเสจนั้น ซึ่งบางเมสเสจจะไม่ได้ใช้พารามิเตอร์ครบทั้งสองตัว

ลักษณะของฟังก์ชั่นประจำวินโดว์ที่มีเมสเสจเป็นอากิวเมนต์เป็นดังนี้

```
LONG FAR PASCAL WndProc( hWnd,wMsg,wParam,lparam)
HWND     hWnd;
WORD     wMsg;
WORD     wParam;
DWORD    lparam;
```

hWnd เป็นพารามิเตอร์ที่เก็บแฮนเดิลของวินโดว์ที่รับเมสเสจนี้ wMsg เป็นตัวเมสเสจที่ส่งมา wParam เป็นพารามิเตอร์เพิ่มเติมของเมสเสจที่ส่งมา โดยข้อมูล 16 บิต และ lparam เป็นพารามิเตอร์เพิ่มเติมของเมสเสจที่ส่งมา โดยเป็นข้อมูล 32 บิต ตัวอย่างของการใช้งานพารามิเตอร์สองตัวหลังเช่น ถ้ามีการเลื่อนเมาส์ก็จะเกิดเมสเสจ WM_MOUSEMOVE และ wParam ก็จะมีข้อมูลของปุ่มที่ถูกกด ส่วน lparam ก็จะเป็นตำแหน่งของเมาส์ที่ถูกเลื่อนมา

6. งานตอบสนองตามปกติ

ฟังก์ชั่น DefWindowProc เป็นฟังก์ชั่นที่ใช้ในการประมวลผลเมสเสจที่ผ่านเข้ามายังฟังก์ชันประจำวินโดว์แต่ไม่ได้ใช้งาน ซึ่งวินโดว์ได้กำหนดการตอบสนองตามปกติที่ฟังกชัน DefWindowProc ในตัววินโดว์ จะดำเนินการเมื่อได้รับเมสเสจต่าง ๆ งานเหล่านั้นมีดังตารางนี้

เมสเสจ	งานมาตรฐาน
WM_ACTIVATE	เซตหรือยกเลิกโฟกัส

เมสเสจ	งานมาตรฐาน
WM_CANCELMODE	ยกเลิกการประมวลผลของสกรอลล์บาร์ ยกเลิกการประมวลผลเมนู ยกเลิกการ capture เมาส์
WM_CLOSE	เรียกใช้งานฟังก์ชั่น DestroyWindow
WM_CTLCOLOR	เซตสีพื้นและสีตัวอักษร ให้ค่าแฮนเดิลของแปรงที่ใช้ระบายแบ๊คกราวน์กลับมา
WM_ERASEBKGND	ถ้ามีการกำหนดสีและรูปแบบของแบ๊คกราวน์ไว้ในคลาส ก็จะใช้ระบายพื้นที่ใช้งานทั้งหมด
WM_GETTEXT	ก๊อปปี้ไตเติ้ลของวินโดว์ลงบัฟเฟอร์ที่ต้องการ
WM_GETTEXTLENGTH	ให้ค่าความยาวของไตเติ้ลของวินโดว์
WM_ICONERASEBKGND	ระบายพื้นที่ใช้งานของไอคอนด้วยแปรงที่ใช้ระบายแบ๊คกราวด์ของวินโดว์ parent
WM_NCACTIVATE	ทำให้วินโดว์นั้นแอคทีฟหรือไม่แอคทีฟ และวาดไอคอนหรือไตเติ้ลบาร์ของวินโดว์ใหม่ด้วย
WM_NCCALCSIZE	คำนวณขนาดของพื้นที่ใช้งาน
WM_NCCREATE	ให้ค่าเริ่มต้นกับสกอลล์บาร์และset default ไตเติ้ลให้กับวินโดว์
WM_NCDDESTROY	ยกเลิกหน่วยความจำที่จองไว้สำหรับไตเติ้ลของวินโดว์
WM_NCHITTEST	พิจารณาว่าเมาส์อยู่ที่ใดในวินโดว์
WM_NCLBUTTONDBLCK	ทดสอบเพื่อหาตำแหน่งปัจจุบันของเมาส์
WM_NCLBUTTONDOWN	พิจารณาว่าปุ่มซ้ายถูกกดเมื่อเมาส์อยู่นอกพื้นที่ใช้งานหรือไม่
WM_NCLBUTTONUP	ทดสอบเพื่อหาตำแหน่งปัจจุบันของเมาส์
WM_NCMOUSEMOVE	ทดสอบเพื่อหาตำแหน่งปัจจุบันของเมาส์
WM_NCPAINT	วาดรูปลงนอกพื้นที่ใช้งาน
WM_PAINT	ทำให้ข้อมูลที่อยู่ในพื้นที่ใช้งานถูกต้อง แต่ไม่วาดใหม่
WM_PAINTICON	วาดไอคอนเมื่อวินโดว์ถูกลดขนาด
WM_QUERYENDSESSION	ให้ค่า TRUE
WM_QUERYOPEN	ให้ค่า TRUE
WM_SETREDRAW	บังคับให้มีการอัพเดตในวินโดว์ส่วนที่ถูกตัด
WM_SETTEXT	เซ็ตและแสดงไตเติ้ลของวินโดว์
WM_SHOWWINDOW	แสดงหรือไม่แสดงวินโดว์
WM_SYSCHAR	สร้างเมสเสจ WM_SYSCOMMAND
WM_SYSCOMMAND	ทำงานตามคำสั่งในเมนูของระบบ

เมสเสจ	งานมาตรฐาน
WM_SYSKEYDOWN	ตรวจสอบคีย์ที่ต้องการและสร้างเมสเสจ WM_SYSCOMMAND ถ้าคีย์ที่กดเป็น Tab หรือ Enter

7. สไตล์ของวินโดว์

เมื่อใช้ฟังก์ชัน Create Window ในการสร้างวินโดว์ใหม่ สามารถกำหนดสไตล์หรือลักษณะเฉพาะของวินโดว์นั้นได้ ซึ่งสไตล์เหล่านี้สามารถนำมาผสมกันในวินโดว์เดียวกันได้

สไตล์ของวินโดว์มีดังนี้

*** วินโดว์แบบซ้อนทับ (Overlapped Windows)**

วินโดว์แบบนี้จะใช้เป็นวินโดว์หลักของแต่ละแอพพลิเคชั่น คือจะอยู่ในระดับบนเสมอ ไม่เป็นวินโดว์ child ของวินโดว์อื่น ส่วนประกอบของวินโดว์แบบนี้สามารถมีได้ครบทุกอย่าง (ไม่ว่าจะเป็นเมนูของระบบ กล่องลดขนาด กล่องขยายขนาด สกรอลล์บาร์) หากมีการกำหนดว่าให้มีและถ้าวินโดว์นี้เป็นวินโดว์หลัก ขอแนะนำให้มีเมนูของระบบและกล่องขยาย/ลดขนาดเสมอ

การกำหนดให้วินโดว์มีสไตล์แบบนี้ให้ใช้ WS_OVERLAPPED (มีกรอบและ caption) หรือ WS_OVERLAPPEDWINDOW (มีกรอบหนา caption เมนูระบบ และ กล่องลด/ขยายขนาด) ในฟังก์ชั่น Create Window

*** วินโดว์แบบมีเจ้าของ (Owned Windows)**

วินโดว์แบบนี้ต้องมีวินโดว์ที่เป็นวินโดว์ parent และวินโดว์ parent นั้นต้องเป็นแบบซ้อนทับ (WS_OVERLAPPED) ด้วยการที่เป็นวินโดว์แบบนี้ทำให้เกิดข้อบังคับขึ้นดังนี้

- วินโดว์แบบมีเจ้าของนี้ จะต้องปรากฏอยู่บนวินโดว์ parent ของมันเสมอ และ ถ้าพยายามย้ายวินโดว์ parent ขึ้นมาข้างบนก็จะมีการเปลี่ยนตำแหน่งของวินโดว์แบบนี้เพื่อให้แน่ใจว่ายังอยู่บนวินโดว์ parent
- เมื่อยกเลิกวินโดว์ parent วินโดว์แบบมีเจ้าของนี้ก็จะถูกยกเลิกไปด้วย
- เมื่อวินโดว์ parent ถูกลดขนาด วินโดว์แบบมีเจ้าของก็จะหายไป

การสร้างวินโดว์ให้มีสไตล์แบบนี้ทำได้โดยกำหนดแฮนเดิลของวินโดว์ parent ไปที่ hWndParant ในพารามิเตอร์ของฟังก์ชัน Create Window ส่วนสไตล์ที่กำหนดให้ก็เป็น WS_OVERLAPPED (กรอบข้อความเป็นวินโดว์แบบนี้โดยอัตโนมัติอยู่แล้ว)

*** วินโดว์แบบป๊อปอัพ (Pop-up Windows)**

วินโดว์แบบป๊อปอัพเป็นชนิดหนึ่งของวินโดว์แบบซ้อนทับ มีคุณสมบัติต่าง ๆ เหมือนกันทั้งหมด เพียงแต่วินโดว์แบบป๊อปอัพจะสามารถเลือกได้ว่าให้มี caption หรือไม่ (ถ้าเป็นวินโดว์แบบซ้อนทับต้องมีเสมอ)

*** วินโดว์ Child (Child Windows)**

วินโดว์แบบนี้จะถูกจำกัดบริเวณอยู่ภายในพื้นที่ใช้งานของวินโดว์ parent เท่านั้น ส่วนมากจะใช้ในการแบ่งพื้นที่ใช้งานของวินโดว์ parent เป็นส่วนย่อย ๆ การสร้างวินโดว์ชนิดนี้ให้ใช้ WS_CHILD ในฟังก์ชัน Create Window ส่วนการแสดงหรือไม่แสดงวินโดว์ก็ใช้ฟังก์ชัน Show Window

ทุก ๆ วินโดว์ child ต้อมีวินโดว์ parent เสมอ วินโดว์ที่เป็น parent ของมันสามารถเป็นวินโดว์ซ้อนทับ วินโดว์แบบป๊อปอัพ หรือว่าเป็นวินโดว์ child เองก็ได้ เมื่อวินโดว์ parent มีวินโดว์ child อยู่ในพื้นที่ใช้งานของมัน ก็หมายความว่าพื้นที่ใช้งานส่วนนั้นก็จะไม่เป็นของวินโดว์ parent อีกต่อไป เมสเสจที่เกิดจากพื้นที่ใช้งานตรงนั้นก็จะถูกส่งไปยังฟังก์ชั่นวินโดว์ child นั้นวินโดว์ child ที่เข้ามาอยู่ในพื้นที่ใช้งานของวินโดว์ parent หนึ่ง ๆ นั้นไม่จำเป็นต้องมีคลาสเดียวกัน ทำให้สามารถรับผิดชอบงานต่าง ๆ กันในพื้นที่ใช้งานเดียวกันได้

ตัววินโดว์ child เองนั้นนอกจากพื้นที่ใช้งานแล้วจะไม่มีส่วนประกอบอื่นโดยอัตโนมัติจำเป็นต้องมีการกำหนดลงไปว่าต้องการอะไรบ้าง (กรอบ ไตเติ้ลบาร์ กล่องลด/ขยายขนาด ฯลฯ) โดยปกติแล้ววินโดว์ child นี้จะต้องมีค่าอ้างอิง(ID) ประจำตัว โดยกำหนดผ่านทาง Create Window เพื่อบอกกับวินโดว์ parent ว่าเมสเสจที่ส่งมานี้มาจากวินโดว์ child ใด

การกำหนดตำแหน่งของวินโดว์ child จะอ้างอิงเทียบกับมุมบนซ้ายของพื้นที่ใช้งานของวินโดว์ parent หากบางส่วนของวินโดว์ child ถูกเลื่อนออกไปนอกพื้นที่ใช้งาน ส่วนนั้นก็จะไม่มีการแสดง ส่วนเมสเสจนั้นก็สามารถส่งตรงไปยังวินโดว์ child ได้โดยตรงถ้าต้องการ เว้นเสียแต่ว่ามีการใช้ Enable Window ในกรณีนี้ Windows จะส่งเมสเสจนั้นไปยังวินโดว์ parent ของวินโดว์ child นั้น เพื่อเปิดโอกาสให้วินโดว์ parent มีการตรวจสอบเมสเสจ และอาจอีนาเบิ้ลวินโดว์ child ขึ้นมาอีกครั้ง

การกระทำส่วนมากที่เกิดขึ้นกับวินโดว์ parent ก็จะมีผลกับวินโดว์ child ด้วย ดังรายการในตารางดังต่อไปนี้

วินโดว์ parent	วินโดว์ child
ถูกแสดง	แสดงขึ้นมาตามวินโดว์ parent ด้วย
ถูกซ่อน	ก็ถูกซ่อนตามวินโดว์ parent
ถูกยกเลิก	ถูกยกเลิก
ถูกเคลื่อนย้าย	ถูกเคลื่อนย้ายโดยอิงกับตำแหน่งใหม่ของวินโดว์ parent
ถูกขยายขนาด (maximize)	มีการวาดส่วนที่ก่อนวินโดว์ parent ขยายเพิ่มหรือเพิ่มขนาด

8. <u>วงจรชีวิตของวินโดว์</u>

หน้าที่หลักของวินโดว์ คือ การรับอินพุตและการแสดงเอาต์พุต ดังนั้นวงจรชีวิตของวินโดว์จึงเริ่มขึ้นเมื่อแอพพลิเคชั่นต้องการรับอินพุตและเอาต์พุต และจะสิ้นสุดเมื่อไม่ต้องการหรือจบแอพพลิเคชั่น โดยปกติแล้ววินโดว์ก็จะคงอยู่จนกว่าจะจบแอพพลิเคชั่น แต่วินโดว์ที่ใช้งานเฉพาะบางอย่างก็จะมีชีวิตสั้น ๆ เช่น วินโดว์ของกรอบข้อความ

วินโดว์เริ่มขึ้นจากขั้นตอนการสร้าง โดยการส่งคลาสของวินโดว์ที่ทำการขึ้นทะเบียนไว้แล้วให้กับฟังก์ชัน Create Window จากนั้น Windows ก็จะไปเตรียมข้อมูลเกี่ยวกับวินโดว์ใหม่นั้นแล้วให้ค่าตัวเลขกลับมาสู่แอพพลิเคชั่นสำหรับใช้ในการอ้างถึงวินโดว์นี้ภายหลัง เรียกตัวเลขนี้ว่าแฮนเดิลของวินโดว์

เมสเสจแรกที่ถูกส่งไปยังฟังก์ชันประจำวินโดว์คือ WM_CREATE ซึ่งเป็นเมสเสจที่บอกว่าให้ทำการเตรียมตัวบางอย่าง ก่อนการแสดงวินโดว์ เช่น การจองหน่วยความจำ หรือ เปิดไฟล์ข้อมูลเตรียมไว้ ข้อมูลที่ส่งมาด้วยก็จะอยู่ใน lParam เป็นพอยน์เตอร์ไปยังโครงสร้าง CREATESTRUCT ซึ่งเก็บข้อมูลเกี่ยวกับวินโดว์นั้นอยู่ เมสเสจ WM_CREATE และ WM_NCCREATE นี้ถูกส่งให้วินโดว์โดยไม่ผ่านคิวของแอพพลิเคชั่น ซึ่งก็หมายความว่าจะทำงานก่อนลูปหลักของแอพพลิเคชั่น

การที่จะเริ่มใช้งานวินโดว์ได้นั้นจะต้องแสดงวินโดว์นั้นขึ้นมาเสียก่อน หากวินโดว์ไม่ได้ถูกสร้างด้วยสไตล์ WS_VISIBLE แล้ว ก็ต้องใช้ฟังก์ชัน Show Window ในการแสดง แต่สำหรับวินโดว์หลักของแอพพลิเคชั่นแล้ว ไม่ควรกำหนดให้เป็นสไตล์ WS_VISIBLE ควรใช้ Show Window แทน โดยใช้ตัวแปร nCmdShow เป็นพารามิเตอร์ของฟังก์ชันเพื่อบอกว่าควรแสดงหรือไม่

เมื่อวินโดว์สิ้นสุดการทำงานหรือจบแอพพลิเคชั่นแล้ว ก็ต้องยกเลิกหรือทำลายวินโดว์นั้นด้วยฟังก์ชัน Destroy Window โดยฟังก์ชันจะลบวินโดว์นั้นออกจากหน้าจอ และส่งเมสเสจ WM_DESTROY และ WM_NCDESTROY ไปยังฟังก์ชันประจำวินโดว์ (อีกทางหนึ่งที่ฟังก์ชันของวินโดว์จะได้รับเมสเสจนี้คือ มีเมสเสจ WM_CLOSE ส่งเข้าไปยังฟังก์ชัน DefWindowProc)

สำหรับวินโดว์ที่เป็นวินโดว์หลักของแอพพลิเคชั่นควรจะถูกทำลายเป็นวินโดว์สุดท้าย และควรมีการบอกให้จบแอพพลิเคชั่นด้วย โดยเมื่อวินโดว์หลักได้รับ wm_destroy แล้วก็ให้เรียกฟังก์ชัน PostQuitMessage เพื่อส่ง WM_QUIT ไปยังคิวของแอพพลิเคชั่น และเมื่อเมนูลูปอ่านเมสเสจนี้ขึ้นมาก็จะเป็นการจบแอพพลิเคชั่น

9. ฟังก์ชันสำหรับสร้างวินโดว์

ฟังก์ชันในกลุ่มนี้มีหน้าที่เกี่ยวกับ การสร้าง ทำลาย เปลี่ยนแปลง หรือดึงข้อมูลจากวินโดว์ เรื่องราวของฟังก์ชันในกลุ่มนี้ แสดงดังตารางดังต่อไปนี้

ฟังก์ชัน	หน้าที่
Adjust Window Rect	คำนวณขนาดของวินโดว์ให้เหมาะกับพื้นที่ใช้งาน
Adjust Window Rect Ex	คำนวณขนาดของวินโดว์แบบ extended ให้เหมาะกับพื้นที่ใช้งาน
Create Window	สร้างวินโดว์ชนิด child , ป๊อปอัพ, ทับกัน
Create Window Ex	สร้างวินโดว์ชนิด child, ป๊อปอัพ, ทับกัน แบบ extended
DefDigProc	กำหนดวิธีการที่จะจัดการกับเมสเสจของกรอบข้อความที่แอพพลิเคชั่นไม่ได้ประมวลผล
DefWindowProc	กำหนดวิธีการที่จะจัดการกับเมสเสจของฟังก์ชันของวินโดว์ที่แอพพลิเคชั่นไม่ได้ประมวลผล
Destroy Window	ยกเลิกวินโดว์
Get Class Info	ดึงข้อมูลเกี่ยวกับคลาสที่ต้องการ
Get Class Long	ดึงข้อมูลของคลาสของวินโดว์จากโครงสร้าง WNDCLASS
Get Class Name	ดึงข้อมูลชื่อของคลาส
Get Class Word	ดึงข้อมูลของคลาสของวินโดว์จากโครงสร้าง WNDCLASS
Get Last Active Popup	ดูว่าวินโดว์ใดแอคทีฟหลังสุด
Get Window Long	ดึงข้อมูลเกี่ยวกับวินโดว์
Get Windto Word	ดึงข้อมูลเกี่ยวกับวินโดว์
Register Class	รีจิสเตอร์คลาสใหม่
Set Class Long	เปลี่ยนแปลงข้อมูลโครงสร้างในแบบ WNDCLASS
Set Class Word	เปลี่ยนแปลงข้อมูลโครงสร้างในแบบ WNDCLASS
Set Window Long	เปลี่ยนลักษณะของวินโดว์
Set Window Word	เปลี่ยนลักษณะของวินโดว์
Unregister Class	ยกเลิกคลาสของวินโดว์จากตารางคลาส

2.3.3. การวาด

ในส่วนนี้จะอธิบายถึงรายละเอียดการแสดงผลของระบบ และการเตรียมวินโดว์สำหรับการวาด ตลอดจนถึงกระบวนการวาดภาพโดยทั่วไป

1. การจัดการเรื่องการแสดงผลของวินโดว์

จอภาพเป็นอุปกรณ์แสดงผลหลักของแอพพลิเคชั่นต่าง ๆ ไม่ว่าจะเป็นของดอส หรือของวินโดว์ แต่เรามักจะเคยชินกับการที่แอพพลิเคชันบนดอสจะได้หน้าจอไปเสียทั้งหมด แต่สำหรับวินโดว์แล้วนั้นจะถือว่าจอภาพเป็นทรัพยากรอย่างหนึ่งที่แต่ละแอพพลิเคชันที่ทำงานอยู่พร้อม ๆ กันในขณะนั้นต้องปันส่วนใช้ร่วมกันวินโดว์จึงมีการจัดการให้แอพพลิเคชันเหล่านั้นสามารถแสดงเอาต์พุตที่ตนเองต้องการได้ และต้องคอยดูและไม่ให้ล่วงล้ำพื้นที่ของแอพพลิเคชันอื่น ๆ

การจัดการกับการแสดงผลของวินโดว์นั้นจะใช้สิ่งที่เรียกว่า *คอนเท็กซ์ดิสเพลย์* ซึ่งเป็นชนิดหนึ่งของ *ดีไวคอนเทกซ์* ด้วยตัวคอนเทกซ์ดิสเพลย์นี้เองทำให้มองแต่ละวินโดว์ได้ว่าอยู่ที่พื้นผิวหรือพื้นที่ที่ต่างกัน แอพพลิเคชันที่ได้คอนเทกซ์ดิสเพลย์นี้ก็สามารถจัดการเกี่ยวกับวินโดว์นั้นได้ทุกอย่าง โดยฟังก์ชันที่เกี่ยวกับเอาต์พุตไปสู่วินโดว์นั้นอยู่ในการควบคุม (เรียกใช้งานผ่าน) ตัว GDI อีกทีหนึ่ง

2. ชนิดของคอนเทกซ์ดิสเพลย์

คอนเทกซ์ดิสเพลย์มีทั้งหมด 4 ชนิดด้วยกันคือ แบบทั่วไป (common) แบบคลาส (class) แบบส่วนตัว (private) และแบบวินโดว์ (window) ซึ่งสามแบบแรกนั้นยินยอมให้มีการวาดลงในพื้นที่ใช้งาน (client area) ของวินโดว์นั้น แต่อย่างสุดท้ายสามารถจะวาดลงไปที่ใดของวินโดว์ก็ได้ การกำหนดว่าวินโดว์จะได้ดิสเพลย์คอนเทกซ์ชนิดใดไปนั้น วินโดว์กำหนดให้โดยดูจากคลาสของวินโดว์นั้น

*** คอนเทกซ์ดิสเพลย์แบบทั่วไป**

คอนเทกซ์ดิสเพลย์ชนิดนี้จะถูกกำหนดให้กับวินโดว์ที่ใช้สไตล์ที่ไม่มีการระบุชนิดของคอนเทกซ์ดิสเพลย์ ซึ่งเมื่อวินโดว์มีคอนเทกซ์ดิสเพลย์ชนิดนี้แล้วก็สามารถที่จะวาดลงไปในพื้นที่ใช้งานของวินโดว์ได้ แต่ก่อนการเรียกใช้ต้องมีการร้องขอใช้ดิสเพลย์เสียก่อน โดยเรียกใช้ฟังก์ชัน GetDC หรือ BeginPaint ซึ่งค่าที่ได้กลับมานั้นเป็นแฮนเดิลของคอนเทกซ์ดิสเพลย์ของวินโดว์นั้น ๆ และค่าแฮนเดิลนี้จะต้องถูกส่งไปให้ฟังก์ชั่น GDI ด้วย เมื่อมีการวาดรูป หลังจากที่วาดเรียบร้อยแล้วก็ต้องใช้ฟังก์ชั่น ReleaseDC หรือ EndPaint เพื่อปล่อยแฮนเดิลของคอนเทกซ์ดิสเพลย์

เมื่อวินโดว์ได้คอนเทกซ์ดิสเพลย์แบบนี้มาใช้งาน วินโดว์จะให้อุปกรณ์ default สำหรับการวาดรูปต่าง ๆ มาให้ ดังนี้

อุปกรณ์	ค่า default
สีพื้น	สีพื้นที่ตั้งค่าจาก Control Panel
โหมดของการวาดพื้น	วาดทับ (OPAQUE)
รูปบิตแมพ	ไม่มี
แปรงสี	สีขาว (WHITE_BRUSH)
พิกัดดั้งเดิมของแปรง	(0,0)
ขอบเขตกำกับ(clipping region)	พื้นที่ใช้งานทั้งหมดกับพื้นที่ที่ควรมีการปรับปรุงหรือวาดใหม่ หากมีพื้นที่ที่อยู่ใต้วินโดว์ child ก็จะถูกคลิปออกไปด้วย
จานสี	จานสีปกติ (DEFAULT_PALETTE)
ตำแหน่งของปากกา	พิกัด (0,0)
พิกัดดั้งเดิมของอุปกรณ์	มุมซ้ายบนของพื้นที่ใช้งาน
โหมดการวาด	ลากเส้นทับ (R2_COPYPEN)
ฟอนต์	ฟอนต์ของระบบ โดยที่หากเป็นแอพพลิเคชันบน 3.1 จะเป็น SYSTEM_FONT แต่หากเป็นแอพพลิเคชันบน 3.0 จะเป็น SYSTEM_FIXED_FONT
ช่องระหว่างตัวอักษร (ช่องไฟ)	0 จุด
การวัดขนาดของจอภาพ	1 จุดต่อ 1 ปอยต์ (MM_TEXT)
ปากกา	สีดำ (BLACK_PEN)
โหมดการเติมสีรูปหลายเหลี่ยม	เติมสีระหว่างด้านคู่และด้านคี่(Alternate)
แฟล็กอ้างอิงการวาด	Absolute
โหมดการ Stretch	Black on White
สีของตัวอักษร	สีที่ตั้งค่าจาก Control Panel ซึ่งมักจะเป็นสีดำ
เนื้อที่เสริม viewport	(1,1)
พิกัดดั้งเดิม viewport	พิกัด (0,0)
เนื้อที่เสริมวินโดว์	(1,1)
พิกัดดั้งเดิมวินโดว์	พิกัด (0,0)

ค่าต่าง ๆ เหล่านี้แอพพลิเคชั่นสามารถเปลี่ยนแปลงได้ตามความพอใจ โดยใช้ฟังก์ชั่นเกี่ยวกับการเลือกวัตถุ ดังตัวอย่างในเรื่องของ " เอาต์พุตสู่วินโดว์ " และเนื่องจากตัวดิสเพลย์คอนเทกซ์แบบนี้จะต้องถูกแบ่งใช้ในหลายวินโดว์ จึงไม่สามารถเก็บค่าอุปกรณ์ต่าง ๆ ที่แต่ละวินโดว์กำหนดไว้สำหรับใช้เอง เมื่อปล่อยคอนเทกซ์ดิสเพลย์ไปแล้ว ครั้งต่อไปที่ร้องขออีกก็ต้องมีการสร้างวัตถุ หรืออุปกรณ์สำหรับการวาดภาพที่ต้องการใหม่

*** คอนเทกซ์ดิสเพลย์แบบคลาส**

คอนเทกซ์ดิสเพลย์แบบนี้เป็นของส่วนรวมระหว่างวินโดว์ที่ใช้คลาสเดียวกัน โดยอุปกรณ์ต่าง ๆ ที่วินโดว์ตั้งให้คอนเทกซ์ดิสเพลย์แบบนี้ก็จะยงคงตกทอดไปถึงวินโดว์ต่อไปที่ร้องขอใช้คอนเทกซ์ดิสเพลย์ด้วย การที่จะให้คลาสใช้คอนเทกซ์ดิสเพลย์แบบนี้ก็ทำได้โดยการกำหนด CS_CLASSDC ให้ตอนขึ้นทะเบียนคลาส

การใช้คอนเทกซ์ดิสเพลย์แบบคลาสนี้ วินโดว์ยังจำเป็นที่ต้องมีการร้องขอใช้ก่อนการวาดรูปใด ๆ โดยใช้ GetDC และ BeginPaint เช่นกัน แต่เมื่อสิ้นสุดการวาดรูปแล้วก็ไม่ต้องใช้ ReleaseDC หรือ EndPaint เพื่อปล่อยแฮนเดิล ส่วนค่า Default อุปกรณ์ต่าง ๆ ก็มีให้เหมือนกับแบบทั่วไป เพียงแต่เมื่อมีการเปลี่ยนอุปกรณ์ใด ๆ ไปแล้วค่านั้นยังคงอยู่ (นอกจากตำแหน่งและพื้นที่ส่วนที่ถูกคลิปเท่านั้นที่ต้องเปลี่ยนไป)ไม่เช่นนั้นแล้วการวาดรูปจะซ้ำอยู่ในวินโดว์ที่ได้คอนเทกซ์ดิสเพลย์เป็นวินโดว์แรกเสมอ) ทำให้วินโดว์ที่ใช้งานต่อไปไม่ต้องมีการเปลี่ยนหรือสร้างใหม่

*** คอนเทกซ์ดิสเพลย์แบบส่วนตัว**

การสร้างคอนเทกซ์ดิสเพลย์แบบส่วนตัวทำได้โดยกำหนด CS_OWNDC ในตอนสร้างวินโดว์ ทำให้วินโดว์มีคอนเทกซ์ดิสเพลย์ใช้ส่วนตัวไม่ปะปนกับของส่วนรวม เมื่อใช้คอนเทกซ์ดิสเพลย์แบบนี้แล้วก็มีการร้องขอเพียงครั้งแรกครั้งเดียวเท่านั้น (โดยใช้ GetDC หรือ BeginPaint) และสามารถใช้ไปได้ตลอด ไม่สามารถยกเลิกได้ (ReleaseDC หรือ EndPaint จะไม่มีผลต่อคอนเทกซ์ดิสเพลย์ชนิดนี้) ส่วนอุปกรณ์ default ต่าง ๆ ก็มีให้ใช้เหมือนกับสองแบบแรก

ดูเหมือนว่าคอนเทกซ์ดิสเพลย์ชนิดนี้จะใช้งานง่ายและสะดวก เพราะไม่ต้องมีการร้องขอทุกครั้งที่ต้องการวาดภาพ แต่ข้อเสียของมันก็มีคือ จะทำให้เปลืองหน่วยความจำในการที่ต้องเก็บข้อมูลของวินโดว์นั้น ๆ เพียงวินโดว์เดียวตลอดเวลา แม้ว่าวินโดว์นั้นไม่ได้อยู่ในกระบวนการวาดภาพอยู่

* คอนเทกซ์ดิสเพลย์แบบวินโดว์

คอนเทกซ์ดิสเพลย์จะยอมให้แอพพลิเคชั่นวาดที่ตำแหน่งใด ๆ บนวินโดว์ก็ได้ ไม่จำกัดเฉพาะในพื้นที่ใช้งานเหมือน 3 แบบแรก จุดอ้างอิงของคอนเทกซ์ดิสเพลย์แบบนี้จะอยู่ทีมุมบนซ้ายสุดของวินโดว์ การขอคอนเทกซ์ดิสเพลย์ชนิดนี้ให้ใช้ฟังก์ชั่น GetWindowDC และใช้ฟังก์ชัน ReleaseDC ในการปล่อยแฮนเดิล

การใช้งานคอนเทกซ์ดิสเพลย์ชนิดนี้ส่วนมากเป็นการวาดพิเศษที่อยู่นอกเหนือพื้นที่ใช้งานของวินโดว์เช่น ต้องการสร้างสกรอลล์บาร์รอบ ๆ วินโดว์นั้น ในกรณีแอพพลิเคชั่นจำเป็นต้องประมวลผลเมสเสจ WM_NCPAINT เอง ซึ่งเป็นเมสเสจที่ร้องขอการวาดนอกพื้นที่ใช้งานใหม่แทนที่จะส่งไปให้ฟังก์ชัน DefWindowProc และหากไม่ต้องการตรวจจับเมสเสจนี้ก็สามารถใช้ฟังก์ชั่น GetSystemMetrics เพื่อหาขนาดของส่วนต่าง ๆ นอกพื้นที่ใช้งาน เช่นเมนูบาร์หรือสกรอลล์บาร์ได้

3. เมสเสจ WM_PAINT

เมื่อมีการเปลี่ยนแปลงเกิดขึ้นกับวินโดว์ ไม่ว่าจะเป็นการย้าย การเปลี่ยนแปลงขนาดหรือถูกวินโดว์อื่นทับ วินโดว์จะบันทึกตำแหน่งที่ต้องมีการวาดใหม่เหล่านี้ไว้ และเมื่อมีโอกาสก็จะส่ง WM_PAINT มาเพื่อให้แอพพลิเคชั่นทำการวาดตรงพื้นที่ที่บันทึกไว้นี้ใหม่ (โดยใช้ BeginPaint และ EndPaint)

ฟังก์ชั่นที่เกี่ยวข้องกับการวาดใหม่นี้มีอีกสองฟังก์ชั่นคือ InvalidateRect และ InvalidateRgn ซึ่งจะใช้การบันทึกพื้นที่ที่ผู้ใช้ต้องการวาดใหม่ (Update region)โดยสามารถบันทึกไว้หลาย ๆ แห่งพร้อมกัน เมื่อฟังก์ชั่นประจำวินโดว์ได้รับ WM_PAINT ก็จะทำการวาดส่วนต่าง ๆ เหล่านั้น (ทั้งที่ผู้ใช้เป็นผู้มาร์กและที่ตัววินโดว์มาร์กเอง) ใหม่ในทีเดียว การทำเช่นนี้ทำให้วินโดว์ไม่ต้องวาดใหม่ทั้งพื้นที่ใช้งาน ซึ่งสามารถประหยัดเวลาในการอัพเดตหน้าจอได้มาก

การโมฆะบางพื้นที่ในส่วนของพื้นที่ที่ต้องถูกวาดใหม่ สามารถทำได้โดยใช้ฟังก์ชั่น ValidateRect และ ValidateRgn ส่วนถ้าต้องการขนาดของพื้นที่ที่ต้องการวาดใหม่ก็สามารถขอดูได้โดยฟังก์ชั่น GetUpdateRect และ GetUpdateRgn

ส่วนอีกฟังก์ชั่นที่สามารถสร้างเมสเสจ WM_PAINT และส่งไปยังวินโดว์ต่าง ๆ ได้ทันทีคือ UpdateWindows ซึ่งการส่ง WM_PAINT ของฟังก์ชั่นนี้จะส่งโดยตรงไปยังฟังก์ชั่นประจำวินโดว์ ทำให้ไม่ต้องเข้าแถวต่อจากเมสเสจในคิวอื่น ๆ ส่วนมากจะใช้ฟังก์ชั่นนี้กับวินโดว์ที่เพิ่งเริ่มสร้าง

4. ฟังก์ชั่นเกี่ยวกับการวาด

หน้าที่ของฟังก์ชั่นในกลุ่มนี้คือ การเตรียมวินโดว์สำหรับการวาดภาพ และทำงานต่าง ๆ ที่เกี่ยวกับด้านกราฟฟิกทั่ว ๆ ไป แม้ว่าส่วนมากของฟังก์ชั่นเหล่านี้จะใช้กับจอแสดงผล แต่ว่าก็ยังมีอีกหลายฟังก์ชั่นที่ใช้ได้กับอุปกรณ์เอาต์พุตทั่วไปดังตารางนี้

ฟังก์ชั่น	หน้าที่
BeginPaint	เตรียมวินโดว์สำหรับการวาด
DrawFocusRect	วาดกรอบเหลี่ยมที่ใช้สำหรับชี้โฟกัส
DrawIcon	วาดไอคอน
DrawText	แสดงข้อความ
EndPaint	สิ้นสุดการวาด
FillRect	ระบายสีในกรอบเหลี่ยมด้วยแปรงที่กำหนด
FrameRect	วาดรูปกรอบในกรอบเหลี่ยมที่ต้องการ
GetDC	ให้ค่าคอนเทกซ์ดิสเพลย์ของพื้นที่ใช้งาน
GetUpdateRgn	หาค่าขอบเขตบริเวณของวินโดว์ที่ควรวาด
GetWindowDC	ขอค่าแฮนเดิลของคอนเทกซ์ดิสเพลย์ของทั้งวินโดว์
Graystring	เขียนข้อความโดยใช้ตัวอักษรจาง
InvalidateRect	กำหนดพื้นที่ที่จะมีการวาดใหม่
InvalidateRgn	ทำให้พื้นที่เป็นโมฆะ ต้องวาดใหม่
InvertRect	กลับสีของภาพในพื้นที่ที่กำหนด
ReleaseDc	ปล่อยแฮนเดิลของคอนเทกซ์ดิสเพลย์
UpdateWindow	บอกแอพพลิเคชั่นว่าบางส่วนของวินโดว์ต้องการวาดใหม่
ValidateRect	เลิกการกำหนดพื้นที่ที่จะวาดใหม่

5. ฟังก์ชั่นเกี่ยวกับสารสนเทศ

ฟังก์ชั่นเกี่ยวกับสารสนเทศมีหน้าที่ในการรวบรวมข้อมูลในเรื่องจำนวน และตำแหน่งของวินโดว์ที่เปิดใช้งานอยู่ ฟังก์ชั่นในกลุ่มนี้มีดังตารางต่อไปนี้

ฟังก์ชั่น	หน้าที่
AnyPopup	ตรวจดูว่ามีวินโดว์แบบป๊อปอัพอยู่หรือไม่
ChildWindowFromPoints	ตรวจดูว่ามีวินโดว์ Child ใดที่อยู่ในพิกัดที่กำหนด
EnumChildWindows	นับจำนวนวินโดว์ Child ของวินโดว์ parent นั้น
EnumTaskWindows	นับจำนวนวินโดว์ที่เกี่ยวข้องกับงานหนึ่ง ๆ
EnumWindows	นับจำนวนวินโดว์ที่ใช้งานอยู่ทั้งหมด
FindWindow	ให้ค่าแฮนเดิลของวินโดว์ที่มีคลาสหรือ caption ที่ต้องการ
GetNextWindow	ให้ค่าวินโดว์ก่อนหน้าหรือต่อจากนี้
GetParent	ให้ค่าแฮนเดิลของวินโดว์ parent ของวินโดว์นั้น
GetTopWindow	ให้ค่าแฮนเดิลของวินโดว์ child ตัวแรกสุด
GetWindow	ให้ค่าแฮนเดิลของวินโดว์จากรายการของวินโดว์เมเนเจอร์
GetWindowTask	ให้ค่าแฮนเดิลของงานที่เกี่ยวข้องกับวินโดว์นั้น
ISChild	พิจารณาว่าวินโดว์เป็นวินโดว์ child หรือไม่
ISWindow	พิจารณาว่าวินโดว์นั้นยังอยู่หรือไม่
SetParent	เปลี่ยนวินโดว์ parent ให้กับวินโดว์ child
WindowFromPoint	ตรวจดูว่ามีวินโดว์ใดอยู่ในพิกัดที่กำหนด

6. ฟังก์ชั่นเกี่ยวกับระบบ

ฟังก์ชั่นเกี่ยวกับระบบจะให้ข้อมูลเกี่ยวกับค่าวัดขนาดของระบบ (System metrics) สีหรือเวลา ฟังก์ชั่นเหล่านี้แสดงดังตารางต่อไปนี้คือ

ฟังก์ชัน	หน้าที่
GetCurrentTime	ให้ค่าเวลาที่ผ่านมาตั้งแต่เริ่มเปิดเครื่อง
GetSysColor	ค่าสีของระบบ
GetSystemMetrics	ข้อมูลเกี่ยวกับค่าวัดขนาดของระบบ ซึ่งค่าวัดขนาดของระบบหมายถึง
SetSysColor	ขนาดของวัตถุต่างๆ ที่ปรากฏในวินโดว์ เปลี่ยนสีของระบบ

7. ฟังก์ชั่นเกี่ยวกับคลิปบอร์ด

คลิปบอร์ดมีกลไกซึ่งทำให้แอพพลิเคชั่นหนึ่งส่งแฮนเดิลของข้อมูลไปยังแอพพลิเคชั่นอื่นได้ ฟังก์ชั่นในกลุ่มนี้มีดังตารางดังต่อไปนี้

ฟังก์ชัน	หน้าที่
CloseClipboard	ปิดคลิปบอร์ด
EmptyClipboard	ลบข้อมูลในคลิปบอร์ดและให้ค่าเจ้าของคลิปบอร์ด
EnumClipboardFormats	ตรวจจำนวนรูปแบบทั้งหมดที่คลิปบอร์ดรู้จัก
GetClipboardData	ดึงข้อมูลจากคลิปบอร์ด
GetClipBoardFormatName	ดึงชื่อรูปแบบที่ใช้อยู่
GetClipboardOwner	ค่าแฮนเดิลของวินโดว์ที่เปิดใช้คลิปบอร์ดอยู่
GetPriorityClipboardFormat	ดูค่ารูปแบบที่มีความสำคัญมากที่สุดในคลิปบอร์ด
ISClipboardFormatAvailable	ให้ค่าเป็นจริงถ้ามีข้อมูลในรูปแบบที่ต้องการ
OpenClipboard	เปิดใช้คลิปบอร์ด
RegisterClipboardFormat	รีจิสเตอร์รูปแบบของข้อมูลในคลิปบอร์ดใหม่
SetClipboardData	ก๊อปปี้แฮนเดิลของข้อมูลในคลิปบอร์ด

8. ฟังก์ชั่นเกี่ยวกับข้อผิดพลาด

ฟังก์ชั่นในกลุ่มนี้จะแสดงข้อความว่ามีข้อผิดพลาดเกิดขึ้น และให้ผู้ใช้เลือกกระทำอย่างใดอย่างหนึ่ง ฟังก์ชั่นเหล่านี้แสดงดังตารางต่อไปนี้

ฟังก์ชั่น	หน้าที่
FlashWindow	ทำให้วินโดว์กะพริบโดยสลับสีของวินโดว์ที่แอคทีฟและที่ไม่แอคทีฟ
MessageBeep	สร้างเสียงบี๊ปทางลำโพงเครื่อง
MessageBox	สร้างกรอบข้อความแสดงข้อความ

9. <u>เมสเสจของวินโดว์</u> ที่ใช้งานในโปรแกรมนี้จะมีดังต่อไปนี้คือ

* เมสเสจเกี่ยวกับการเริ่มต้น

เมสเสจ	หน้าที่
WM_INITDIALOG	ถูกส่งทันทีหลังจากที่กรอบข้อความถูกแสดง
WM_INITMENU	จะถูกส่งก่อนการแสดงเมนูเพื่อเซตค่าเริ่มต้น
WM_INITMENUPOPUP	ถูกส่งทันทีหลังจากที่เมนูแบบป๊อปอัพถูกแสดง

* เมสเสจเกี่ยวกับระบบ

เมสเสจนี้ถูกส่งจากวินโดว์ไปยังแอพพลิเคชั่นเมื่อผู้ใช้เรียกใช้งานเมนูของระบบ (System Menu), Scroll Bar หรือกล่องกด / เพิ่มขนาด แม้ว่าแอพพลิเคชั่นสามารถจัดการกับเมสเสจเหล่านี้เองได้ แต่ส่วนมากก็จะนิยมผ่านไปให้ DefWindowProc เป็นตัวจัดการให้แทน

เมสเสจ	หน้าที่
WM_SYSCHAR	ถูกส่งเมื่อ WM_SYSKEYUP หรือ WM_SYSKEYDOWN ถูกแปล
WM_SYSCOMMAND	ถูกส่งเมื่อผู้ใช้เลือกคำสั่งในเมนูของระบบ
WM_SYSDEADCHAR	ถูกส่งเมื่อ WM_SYSKEYUP หรือ WM_SYSKEYDOWN ถูกแปล
WM_SYSKEYDOWN	ถูกส่งเมื่อผู้ใช้กด Alt ค้างไว้และกดคีย์ตัวอื่นตาม
WM_SYSKEYUP	ถูกส่งเมื่อผู้ใช้ปล่อยปุ่มอื่นขณะที่กด Alt ค้างไว้

* เมสเสจเกี่ยวกับคลิปบอร์ด

เมสเสจเกี่ยวกับคลิปบอร์ดจะส่งไปยงวินโดว์ที่ใช้คลิปบอร์ดอยู่เพื่อบอกว่ามีอีกแอพพลิเคชั่นกำลังต้องการใช้คลิปบอร์ดเช่นกัน เมสเสจในกลุ่มแสดงดังตารางต่อไปนี้

เมสเสจ	หน้าที่
WM_ASKCBFORMATNAME	ดึงฟอร์แมตของ CF_OWNERDISPLAY
WM_CHANGECBCHAIN	แจ้งให้ทราบว่าวินโดว์แรกในรายการของวินโดว์ที่ใช้คลิปบอร์ดนั้นกำลังจะถูกลบทิ้งจากรายการ
WM_DESTROYCLIPBOARD	ถูกส่งเมื่อกำลังจะลบข้อมูลในคลิปบอร์ด
WM_DRAWCLIPBOARD	แจ้งให้วินโดว์แรกในรายการทราบว่ามีการเปลี่ยนแปลงข้อมูลในคลิปบอร์ด
WM_HSCROLLCLIPBOARD	แจ้งว่ามีการใช้ Scroll Bar แบบนอนในคลิปบอร์ดที่มีข้อมูลแบบ CF_OWNERDISPLAY อยู่
WM_PAINTCLIPBOARD	แจ้งให้ทราบว่าต้องมีการวาดพื้นที่ใช้งานของคลิปบอร์ดที่มีข้อมูลแบบ CF_OWNERDISPLAY ใหม่
WM_RENDERALLFORMATS	แจ้งให้เจ้าของคลิปบอร์ดทราบว่าต้องมีการ render ข้อมูลในคลิปบอร์ดในทุก ๆ ฟอร์แมตที่เป็นไปได้
WM_REDERFORMAT	แจ้งให้เจ้าของคลิปบอร์ดทราบว่าต้องมีการจัดรูปแบบข้อมูลสุดท้ายที่ก๊อบปี้ไปยังคลิปบอร์ด
WM_SIZECLIPBOARD	แจ้งให้เจ้าของคลิปบอร์ดทราบว่ามีการเปลี่ยนขนาดของวินโดว์ของคลิปบอร์ด
WM_VSCROLLCLIPBOARD	แจ้งว่ามีการใช้ Scroll Bar แบบตั้งในคลิปบอร์ดที่มีข้อมูลแบบ CF_OWNERDISPLAY อยู่

*** เมสเสจแจ้งเหตุ**

เป็นเมสเสจที่แจ้งไปยังวินโดว์ parent ของคอนโทรลว่ามีการกระทำเกิดขึ้นกับคอนโทรลนั้น ๆ ซึ่งเมสเสจแบบนี้จะเป็นลักษณะของพารามิเตอร์ที่ถูกส่งไปกับ WM_COMMAND โดยที่พารามิเตอร์ตัวแรกคือ wParam นั้นจะเป็นค่า ID ของคอนโทรล ส่วนเวิร์ดต่ำของ lParam จะเป็นแฮนเดิลของคอนโทรล และเวิร์ดสูงของ lParam จะเป็นเมสเสจแจ้งเหตุนี้

เมสเสจ	หน้าที่
BN_CLICKED	แจ้งว่าปุ่มกดนั้นถูกคลิก
BN_DOUBLECLICKED	แจ้งว่าผู้ใช้ดับเบิ้ลคลิกที่ปุ่มที่สร้างเองหรือปุ่มวิทยุ

2.3.4. ช่องเกี่ยวโยง (Hook)

ช่องเกี่ยวโยงนับเป็นกลไกการจัดการเมสเสจอย่างหนึ่งของวินโดว์ ที่สามารถให้แอพพลิเคชั่นสามารถเข้าถึงเมสเสจที่ส่งกันไปมา วินโดว์มีช่องเกี่ยวโยงให้ใช้กันหลายแบบ ซึ่งแต่ละแบบก็จะมีความสามารถในการเข้าถึงเมสเสจต่างกันไป การใช้ประโยชน์จากช่องเกี่ยวโยงคือการติดตั้งฟังก์ชันกรองเมสเสจ (filter function) ซึ่งจะจัดการกับเมสเสจจากช่องเกี่ยวโยงก่อนที่เมสเสจนั้นจะถูกส่งไปยังฟังก์ชั่นประจำวินโดว์ที่อยู่ปลายทาง

1. ลูกโซ่ของฟังก์ชั่นการกรองเมสเสจ

ลูกโซ่ของฟังก์ชั่นกรองเมสเสจเกิดจากการต่อฟังก์ชั่นกรองเมสเสจหลาย ๆ ฟังก์ชั่นเข้ากับช่องเกี่ยวโยงเดียวกันของระบบ โดยที่ตำแหน่งแรกสุดของลูกโซ่คือ ช่องเกี่ยวโยงนั่นเอง และปลายของลูกโซ่ก็จะไปอยู่ที่ฟังก์ชั่นประจำวินโดว์ เมื่อเราติดตั้งฟังก์ชันกรองเมสเสจ ฟังก์ชั่นดังกล่าวจะเกี่ยวข้องกับช่องเกี่ยวโยงอันถือว่าเป็นจุดเริ่มต้นของลูกโซ่ รับเมสเสจแล้วส่งเมสเสจที่กรองแล้วโยงเข้ากับฟังก์ชันกรองเมสเสจที่เคยติดตั้งอยู่ (ซึ่งตอนนี้อยู่ในตำแหน่งถัดไป) ฟังก์ชันกรองถัดไปนั้นก็จะทำเช่นนี้ต่อไปเรื่อย ๆ เป็นลูกโซ่ที่จะโยงเขาหาฟังชันประจำวินโดว์ในที่สุด

ฟังก์ชั่นกรองเมสเสจจะสามารถโยงตัวเองเข้ากับช่องเกี่ยวโยงด้วยฟังก์ชัน SetWindowHook ในแต่ละครั้งที่เรียก ฟังก์ชันกรองเมสเสจเหล่านั้นก็จะถูกโยงไปยังจุดเริ่มต้นของลูกโซ่ (ซึ่งอาจมีฟังก์ชันกรองเมสเสจเกี่ยวอยู่) นั่นคือโยงต่อเข้ากับช่องเกี่ยวโยง และทุกครั้งที่แอพพลิเคชั่นติดตั้งฟังก์ชันกรองเมสเสจเข้ากับช่องเกี่ยว ก็จะต้องสำรองพื้นที่สำหรับเก็บตำแหน่งแอดเดรสของฟังก์ชันกรองเมสเสจทีเคยอยู่เดิม เพื่อส่งเมสเสจที่กรองแล้วโยงเข้ากับฟังก์ชันกรองที่อยู่ถัดไป

เมื่อฟังก์ชั่นกรองเมสเสจทำงานของตนเรียบร้อย ก็จะเรียกฟังก์ชัน DefHookProc ฟังก์ชันนี้จะใช้ค่าแอดเดรสที่เคยเก็บไว้เพื่อเรียกฟังก์ชันที่ควรจะอยู่ถัดไปในลูกโซ่

การจะถอดฟังก์ชั่นกรองเมสเสจออกจากลูกโซ่ คือแอพพลิเคชั่นเรียกใช้ฟังก์ชัน UnHookWindowHook ด้วยชนิดของช่องเกี่ยวโยงและตัวชี้ไปยังฟังก์ชันกรองนั้น

ช่องเกี่ยวโยงวินโดว์และช่องเกี่ยวโยงดีบักมีดังตารางนี้

ชนิด	จุดประสงค์
WH_CALLWNDPROC	ติดตั้งฟังก์ชั่นกรองเมสเสจของวินโดว์
WH_CBT	ติดตั้งฟังก์ชั่นกรองเมสเสจสำหรับช่วยการสอน
WH_DEBUG	ติดตั้งฟังก์ชั่นกรองเมสเสจสำหรับการดีบัก
WH_GETMESSAGE	ติดตั้งฟังก์ชั่นกรองเมสเสจ (เฉพาะวินโดว์รุ่น ดีบัก)
WH_HARDWARE	ติดตั้งฟังก์ชั่นกรองเมสเสจสำหรับฮาร์ดแวร์ที่ไม่เป็นมาตรฐานของวินโดว์
WH_JOURNALPLAYBACK	ติดตั้งฟังก์ชั่นกรองเมสเสเล่นกลับ
WH_JOURNALRECORD	ติดตั้งฟังก์ชั่นกรองเมสเสจบันทึก
WH_KEYBOARD	ติดตั้งฟังก์ชั่นกรองเมสเสจคีย์บอร์ด
WH_MOUSE	ติดตั้งฟังก์ชั่นกรองเมสเสจเมาส์
WH_MSGFILTER	ติดตั้งฟังก์ชั่นกรองเมสเสจทั่วไป
WH_SYSMSGFILTER	ติดตั้งฟังก์ชั่นกรองเมสเสจทั้งระบบ

เพิ่มเติม ช่องเกี่ยวโยงระบบ WH_CALLWNDRROC และ WH_GETMESSAGE จะมีผลกระทบต่อประสิทธิภาพของระบบ จึงควรใช้เฉพาะในการดีบักเท่านั้น

2. <u>การติดตั้งฟังก์ชั่นกรองเมสเสจ</u>

การติดตั้งฟังก์ชันกรองเมสเสจ มีกระบวนการต่อไปนี้

1. ประกาศฟังก์ชั่นกรองเมสเสจให้เป็นฟังก์ชั่นส่งออกในไฟล์กำหนดลักษณะโมดูล (ไฟล์ .DEF)
2. รับค่าตำแหน่งแอดเดรสด้วยการเรียกใช้ฟังก์ชั่น GetProcAddress (ฟังก์ชั่น MakeProcInstance) จะใช้ในกรณีที่ฟังก์ชั่นนั้นมิได้อยู่ในไลบรารี (DLL)
3. เรียกฟังก์ชั่น SetWindowHook กำหนดชนิดของช่องเกี่ยวโยง และตำแหน่งแอดเดรสของฟังก์ชั่น (ได้จากฟังก์ชัน GetPorcAddress)
4. รับค่าที่ได้จากการเรียกใช้ฟังก์ชั่น SetWindowHook เก็บไว้ในตำแหน่งที่สำรองไว้ ค่าที่ได้นี้คือแฮนเดิลของฟังก์ชั่นกรองเมสเสจเดิม

เพิ่มเติม

ฟังก์ชันกรองเมสเสจจะต้องบรรจุอยู่ในไลบรารีที่มีโค้ดเซกเมนต์และดาต้าเซกเมนต์แบบอยู่กับที่ ซึ่งจะทำให้ช่องเกี่ยวโยงและฟังก์ชั่นกรองเมสเสจนี้สามารถทำงานได้ในหน่วยความจำ EMS แบบ large frame

3. <u>ฟังก์ชันเกี่ยวกับการเชื่อมโยง</u> แสดงไว้ดังตารางต่อไปนี้

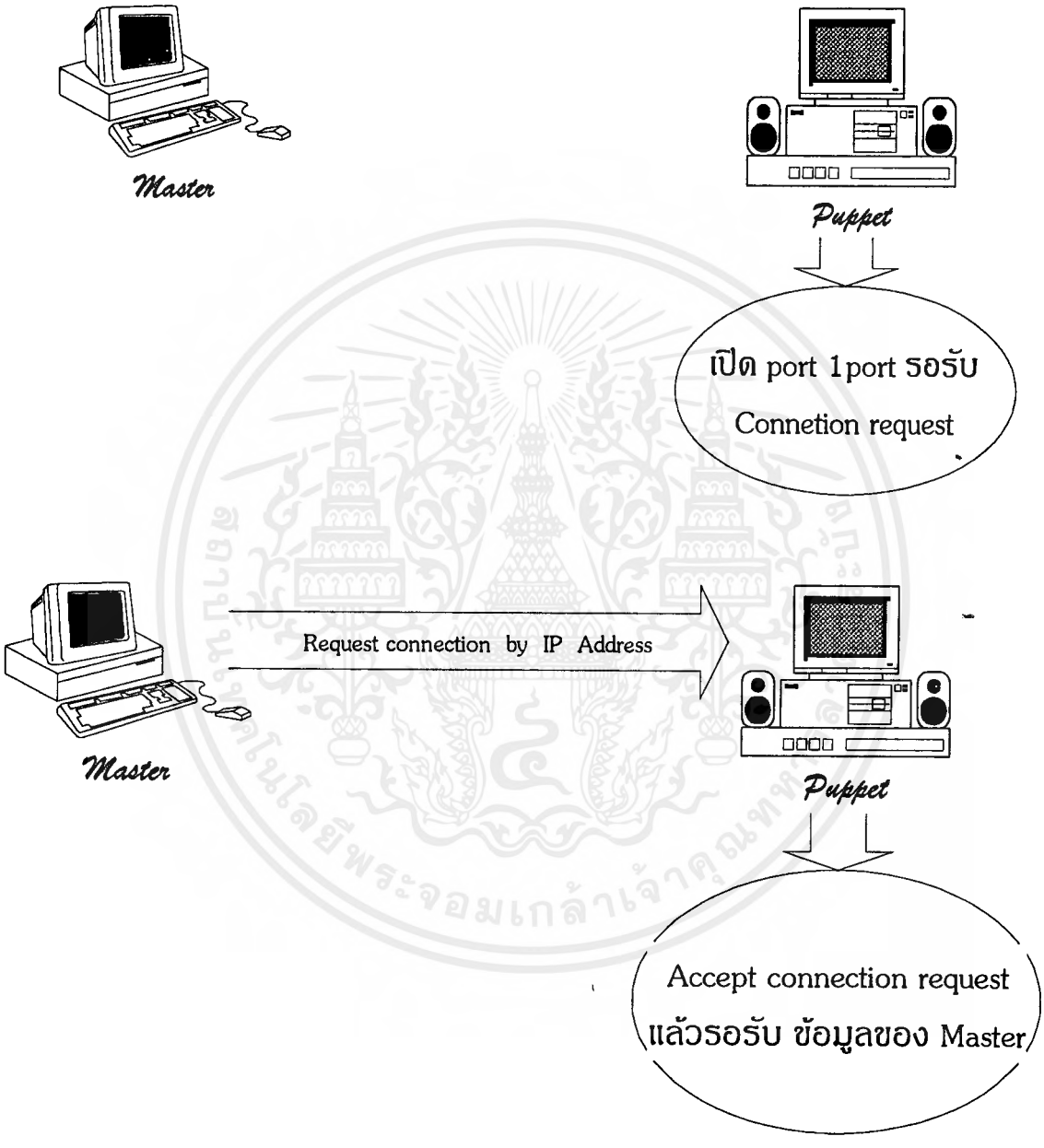
ฟังก์ชัน	สรรพคุณ
CallMsgFilter	ส่งเมสเสจและข้อมูลอื่น ๆ ไปให้ฟังก์ชันกรองเมสเสจ
CallNextHookEx	ส่งเรื่องราวเกี่ยวกับช่องเกี่ยวโยงไปยังลูกโซ่ฟังก์ชันกรองเมสเสจ
DefHookProc	เรียกฟังก์ชันกรองเมสเสจถัดไปในลูกโซ่ฟังก์ชันกรองเมสเสจ
SetWindowHookEx	ติดตั้งฟังก์ชันกรองเมสเสจของระบบ หรือฟังก์ชันกรองเมสเสจของแอพพลิเคชัน หรือทั้งคู่
UnHookWindowsHookEx	กำจัดฟังก์ชันกรองออกจากลูกโซ่ฟังก์ชันกรองเมสเสจ

บทที่ 3
การออกแบบโปรแกรม

ปัญหาพิเศษ เรื่อง "การควบคุมคอมพิวเตอร์ระยะไกลผ่านTCP/IP" คือการควบคุมเครื่องคอมพิวเตอร์ที่อยู่ห่างไกล ซึ่งในที่นี้ขอเรียกเครื่องที่ถูกควบคุมว่าเครื่อง " Puppet " ซึ่งเครื่องที่ทำการควบคุมนั้นเราจะขอเรียกว่าเครื่อง " Master " ซึ่งเครื่อง Masterจะสามารถรับรู้หน้าจอที่เปลี่ยนไปของเครื่อง Puppet และสามารถส่ง Input event (Mouse Event และKeyboard Event) จากเครื่อง Master ไปให้เกิดขึ้นบนเครื่อง Puppet ได้ ซึ่งการทำงานและการสื่อสารข้อมูลระหว่าง Master และ Puppet ได้ถูกออกแบบขั้นต้นไว้ดังนี้

3.1 การสื่อสารระหว่าง Master และ Puppet ช่วง Start up

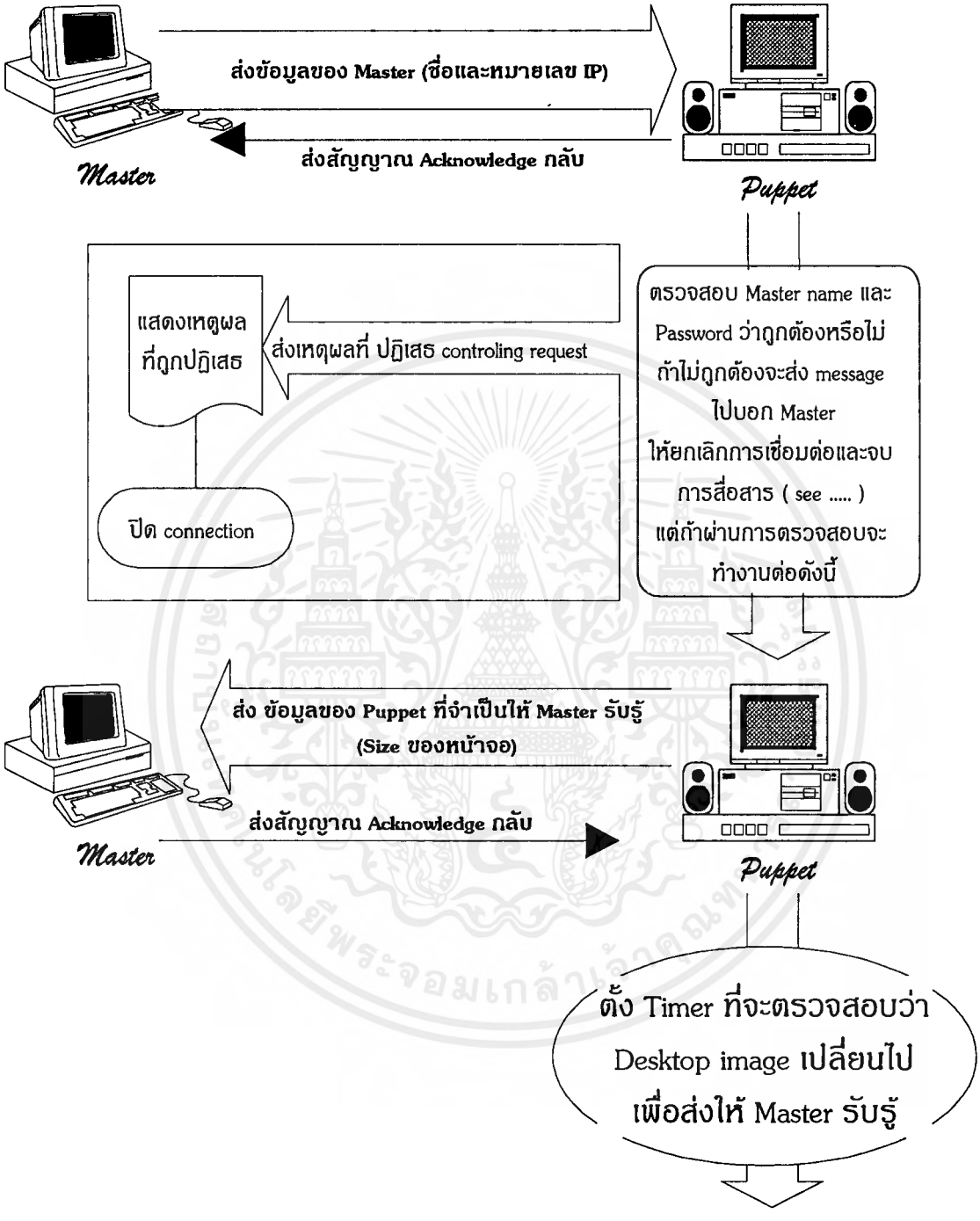
เริ่มต้นเมื่อรันโปรแกรมฝั่ง Master และโปรแกรมฝั่ง Puppet แล้วให้โปรแกรมฝั่ง Puppet ทำการเปิดพอร์ต 1 พอร์ตรอรับการเชื่อมต่อ หลังจากนั้นเมื่อฝั่ง Master ทำการร้องขอการเชื่อมต่อโดยส่งหมายเลข IP ไปให้ฝั่ง Puppet แล้วฝั่ง Puppet จะทำการตัดสินใจว่าจะเลือกยอมรับการเชื่อมต่อหรือปฏิเสธการเชื่อมต่อ ซึ่งจะได้แสดงแผนภาพให้เห็นถึงการสื่อสารระหว่าง Master และ Puppet ไว้ดังรูปที่ 3.1 นี้



รูปที่ 3.1 แผนภาพแสดงการสื่อสารระหว่าง Master และ Puppet (ในช่วง Start up)

3.2 การสื่อสารระหว่าง Master และ Puppet ระหว่างการเริ่มการเชื่อมต่อ

เมื่อฝั่ง Puppet ทำการยอมรับการเชื่อมต่อจากฝั่ง Master แล้วฝั่ง Master ต้องทำการส่งชื่อและหมายเลข IP ของ Master ไปให้ฝั่ง Puppet และเมื่อฝั่ง Puppet ได้รับข้อมูลจะทำการตรวจสอบ Master Name และ Password ว่าถูกต้องหรือไม่ ถ้าไม่ถูกต้องจะส่งข้อความไปปฏิเสธการเชื่อมต่อและทำการปิดการเชื่อมต่อนั้น แต่ถ้าข้อมูลถูกต้องฝั่ง Puppet จะทำการส่งข้อมูลเกี่ยวกับขนาดหน้าจอและหน้าจอของฝั่งตนไปให้ฝั่ง Master และจะทำการกำหนดเวลาในการตรวจจับการเปลี่ยนแปลงของหน้าจอ เมื่อถึงเวลาที่ได้กำหนดไว้จะทำการส่งหน้าจอปัจจุบันไปให้ฝั่ง Master และทุกครั้งที่ฝั่งหนึ่งส่งข้อมูลไป อีกฝั่งหนึ่งเมื่อได้รับข้อมูลแล้วต้องส่งสัญญาณ Acknowledge กลับเพื่อบอกว่าได้รับข้อมูลถูกต้องเรียบร้อยแล้ว ซึ่งได้แสดงแผนภาพแสดงการสื่อสารระหว่าง Master และ Puppet ระหว่างเริ่มการเชื่อมต่อดังรูปที่ 3.2 นี้

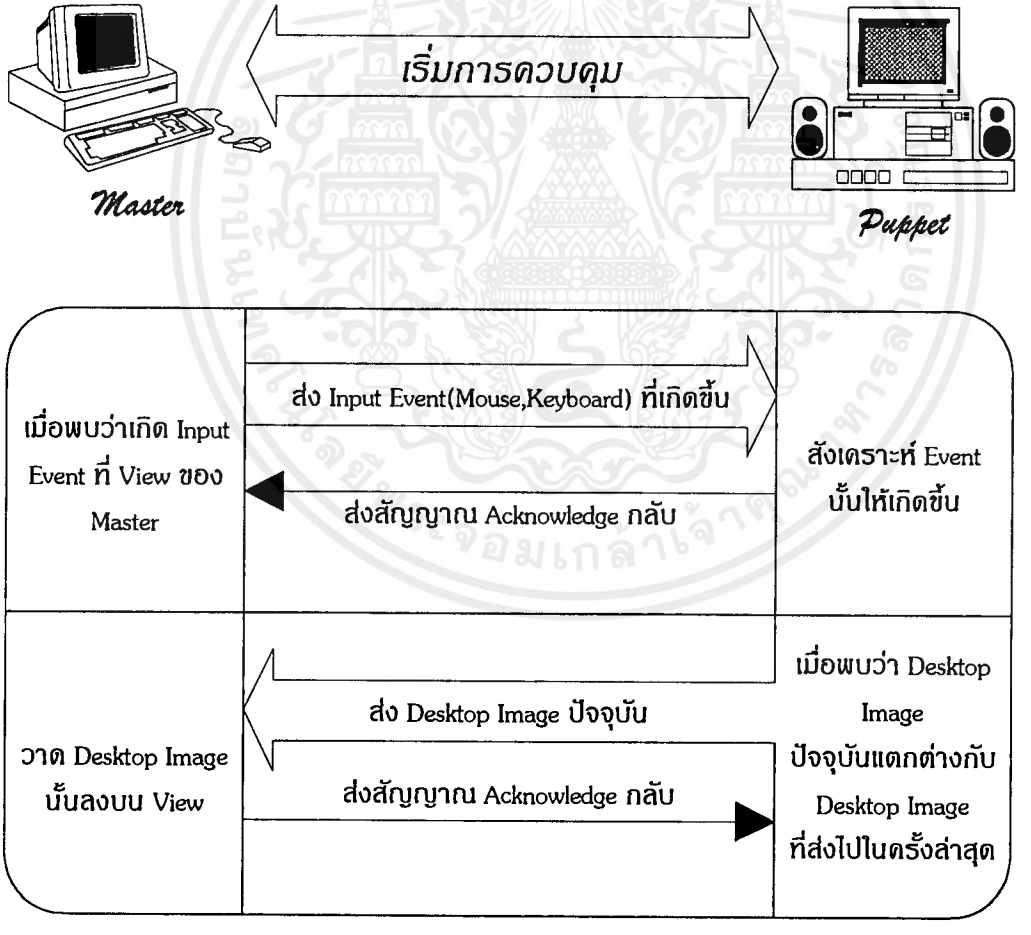


รูปที่ 3.2 แผนภาพแสดงการสื่อสารระหว่าง Master และ Puppet (ระหว่างการเริ่มการเชื่อมต่อ)

3.3 การสื่อสารระหว่าง Master และ Puppet ระหว่างการควบคุม

ระหว่างที่มีการควบคุมฝั่ง Master จะทำการส่ง Event ของเมาส์และคีย์บอร์ดที่เกิดขึ้นไปให้ฝั่ง Puppet เมื่อฝั่ง Puppet ได้รับ Event จากฝั่ง Master แล้วจะทำการส่งสัญญาณ Acknowledge กลับและจะทำการสังเคราะห์ Event ที่ได้รับมาให้เกิดที่เครื่องของตน

ส่วนทางฝั่ง Puppet ระหว่างที่มีการควบคุมจะทำการส่งภาพ Desktop Image ของตนไปให้ฝั่ง Master เมื่อ Master ได้รับข้อมูลแล้วจะทำการส่งสัญญาณ Acknowledge กลับเพื่อบอกว่าได้รับข้อมูลเรียบร้อยแล้วและทุกครั้งเมื่อถึงเวลาที่ได้กำหนดไว้ฝั่ง Puppet จะทำการส่งหน้าจอปัจจุบันไปให้ฝั่ง Master เพื่อให้ฝั่ง Master ได้เห็นหน้าจอที่เปลี่ยนแปลงล่าสุดเพื่อจะทำการควบคุมได้ถูกต้อง จึงได้แสดงแผนภาพการสื่อสารระหว่าง Master และ Puppet ระหว่างที่มีการควบคุมดังรูปที่ 3.3 นี้



รูปที่ 3.3 แผนภาพแสดงการสื่อสารระหว่าง Master และ Puppet (ระหว่างการควบคุม)

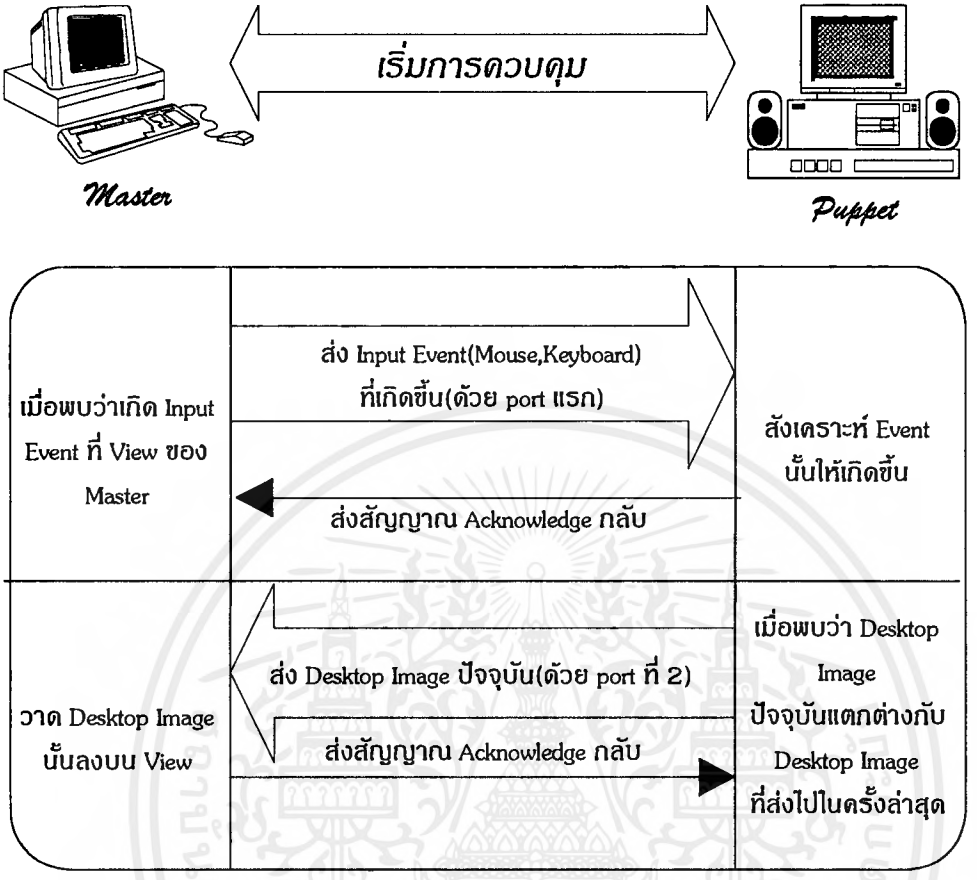
***หมายเหตุ :**

เมื่อเครื่องคอมพิวเตอร์ใดๆ ได้ทำการส่งแพกเก็ตออกไป จะไม่มีการส่งแพกเก็ตใดๆ อีกจนกว่าจะได้รับสัญญาณ Acknowledge จากเครื่องคอมพิวเตอร์อีกฝั่งหนึ่ง ซึ่งสัญญาณ Acknowledge นี้เราส่งไปเพื่อยืนยันว่าแพคเก็ตนั้นได้ถูกส่งไปถึงแล้ว และถ้าขณะที่รอสัญญาณ Acknowledge ถ้าเครื่องคอมพิวเตอร์ของตนต้องการส่งแพกเก็ตใด ๆ ออกมา จะมีการนำแพกเก็ตนั้นเก็บไว้ในคิว และเมื่อได้รับสัญญาณ Acknowledge จากเครื่องคอมพิวเตอร์อีกฝั่งหนึ่งแล้ว จะนำแพกเก็ตที่อยู่ในคิวออกมาและทำการส่งแพกเก็ตนั้นออกไป และเมื่อเครื่องคอมพิวเตอร์ได้รับแพกเก็ตเมื่อใด จะต้องทำการส่งสัญญาณ Acknowledge ออกไปทันที

เนื่องจากหากปล่อยให้ส่งแพกเก็ตโดยไม่มีการควบคุมจะพบว่าแพกเก็ตที่รับไปไม่ถูกต้องซึ่งคณะผู้จัดทำสันนิษฐานว่าเป็นการซ้อนทับกันของแพคเก็ตจึงได้ทำ Handshake (การรับสัญญาณ Acknowledge) ซึ่งทำให้การส่ง/รับแพคเก็ตถูกต้องตามต้องการ แต่วิธีนี้มีปัญหาเมื่อเกิด Critical section คือ เมื่อ Puppet ส่ง Desktop Image ไปให้ Master ในขณะที่ Master กำลังส่ง event มาให้ puppet ซึ่งต่างฝ่ายก็ต่างรอ สัญญาณ Acknowledge แต่ก็ไม่สามารถส่งได้เพราะมันไม่สามารถส่งแพคเก็ตใดๆได้ถ้าหากยังไม่ได้รับสัญญาณ Acknowledge จึงจะได้กล่าวถึงวิธีแก้ปัญหานี้ต่อไป

3.4 การสื่อสารระหว่าง Master และ Puppet ระหว่างการควบคุม (ที่แก้ไขแล้วเพื่อแก้ปัญหา Critical Section)

Puppet จะเปิด port ไว้ 2 port เพื่อให้ Master มาทำการเชื่อมต่อ โดยที่ Port แรกจะมีไว้ให้ Master ส่ง Input Event (และรอรับ Acknowledge ของ event นั้นๆ) และ Port ที่ 2 เปิดไว้เพื่อให้ Puppet ส่ง Desktop Image (และรอรับ Acknowledge ของ Desktop Image นั้นๆ) ซึ่งจะทำให้แผนภาพแสดงการสื่อสารระหว่าง Master และ Puppet (ระหว่างการควบคุม) เปลี่ยนไปดังรูปที่ 3.4 นี้



รูปที่ 3.4 แผนภาพแสดงการสื่อสารระหว่าง Master และ Puppet (ระหว่างการควบคุม ที่แก้ไขแล้ว)

บทที่ 4
การพัฒนาโปรแกรมการควบคุมคอมพิวเตอร์ระยะไกลผ่าน TCP/IP

4.1 อุปกรณ์ที่ใช้ในการพัฒนาโปรแกรม

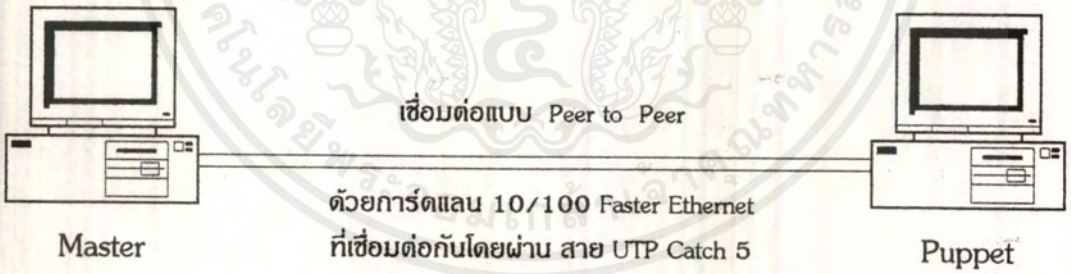
4.1.1 ด้านฮาร์ดแวร์

เครื่องคอมพิวเตอร์สองเครื่องรุ่น Pentium 133 ขึ้นไป ที่มีหน่วยความจำ 64 MB ขึ้นไป(ถ้าเครื่องมีประสิทธิภาพดีจะยิ่งทำให้การทำงานสมบูรณ์ และมีประสิทธิภาพมากขึ้น)

4.1.2 ด้านซอฟท์แวร์

ระบบปฏิบัติการวินโดว์ส 95

4.1.3 ด้านเนตเวิร์ค



4.2 ขั้นตอนการพัฒนาโปรแกรม

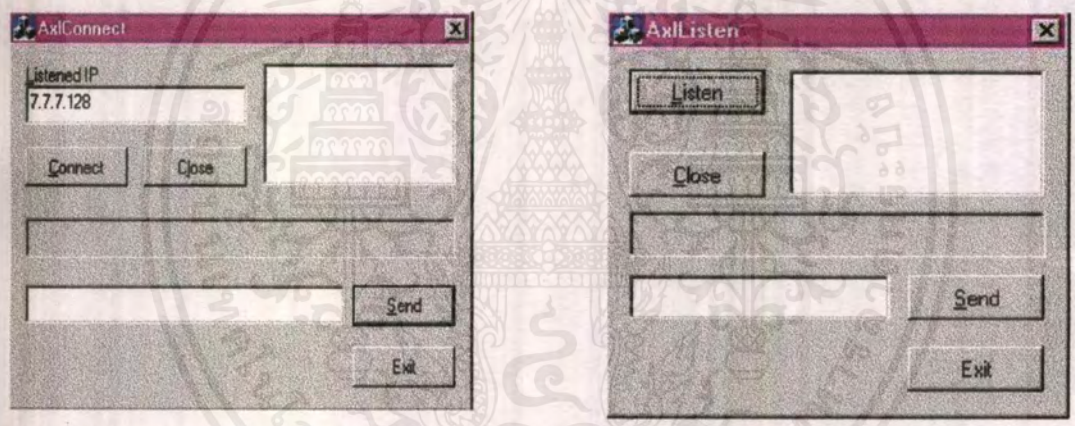
4.2.1 <u>การทดลองที่ 1</u> เขียนโปรแกรมที่ส่งข้อความเพื่อทดสอบการเชื่อมต่อเครื่องคอมพิวเตอร์สองเครื่องโดยผ่านโปรโตคอล TCP/IP

<u>การทดสอบ</u> เริ่มต้นด้วยการรันโปรแกรมสองโปรแกรมพร้อมกันคือ

1. โปรแกรมฝั่ง Connect (โปรแกรมชื่อ **AxlConnect**)
2. โปรแกรมฝั่ง Listen (โปรแกรมชื่อ **AxlListen**)

จะพบกับหน้าจอการทำงานทั้งสองฝั่งดังต่อไปนี้

<u>ฝั่ง Connect</u> <u>ฝั่ง Listen</u>

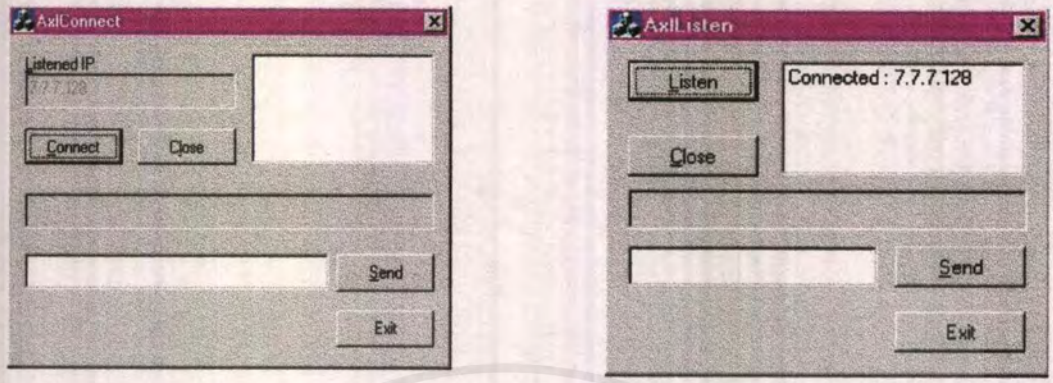


รูปที่ 4.1 แสดงหน้าจอของฝั่ง Connect และฝั่ง Listen ขณะเริ่มต้นรันโปรแกรม

<u>ขั้นที่ 1</u> ที่ฝั่ง Listen ต้องกดปุ่ม Listen เมื่อกดปุ่ม Listen จะมีการเรียกใช้ฟังก์ชั่น OnButtonListen ที่ฟังก์ชั่นนี้จะมีการเปิด Port รอรับการเชื่อมต่อ

<u>ขั้นที่ 2</u> ที่ฝั่ง Connect เมื่อใส่หมายเลข IP ของฝั่ง Listen เสร็จเรียบร้อยแล้วให้กดที่ปุ่ม Connectเมื่อมีการกดที่ปุ่ม Connect จะมีการเรียกใช้ฟังก์ชั่น OnButtonConnect ที่ฟังก์ชั่นนี้จะมีการเปิด Port หมายเลขเดียวกับฝั่ง Listen และทำการเชื่อมต่อกับฝั่ง Listen ด้วยหมายเลข IPที่รับเข้ามาจาก IDC_EDIT_IP เมื่อกดปุ่ม Connect แล้วจะปรากฏหน้าจอของทั้งสองฝั่งดังนี้

ฝั่ง Connect **ฝั่ง Listen**

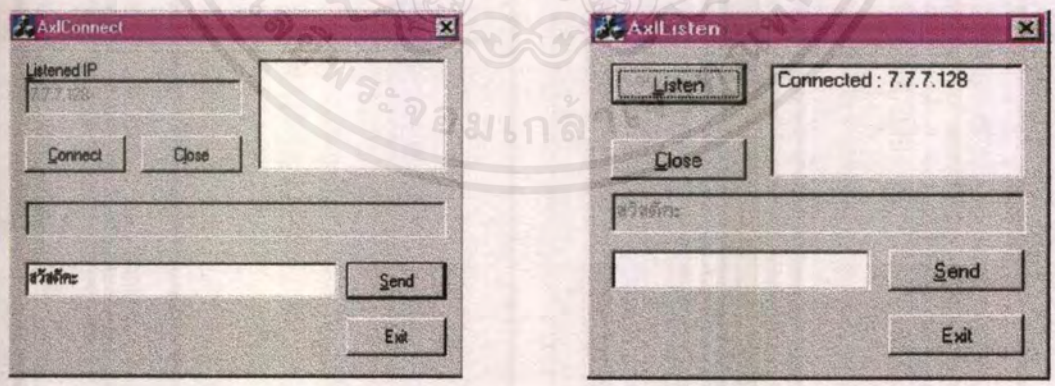


รูปที่ 4.2 แสดงหน้าจอของฝั่ง Connect และฝั่ง Listen ขณะมีการเชื่อมต่อกันด้วยหมายเลข IP

ขณะนี้เครื่องที่ฝั่ง Connect ได้เชื่อมต่อกับฝั่ง Listen เรียบร้อยแล้วด้วยหมายเลข IP 7.7.7.128 โดยผ่านพอร์ตที่ได้ตั้งไว้ แสดงว่าทั้งฝั่ง Coonnect และฝั่ง Listen สามารถเชื่อมต่อกันด้วย TCP/IP

ขั้นที่ 3 เพื่อแน่ใจว่าทั้งสองฝั่งได้เชื่อมต่อกันได้จริงจึงลองส่งข้อความจากฝั่งหนึ่งไปอีกฝั่งหนึ่งซึ่งแสดงให้เห็นดังรูปที่ 4.3 จะเห็นว่าให้ฝั่ง Connect ลองส่งข้อความว่า "สวัสดีคะ" เมื่อกดปุ่ม Sendจะมีการเรียกฟังก์ชั่นOnButtonSend ซึ่งจะทำการรับข้อความจาก IDC_EDIT_SEND และทำการส่งข้อความนั้นไปให้ฝั่ง Listen ซึ่งฝั่ง Listen จะปรากฏข้อความ"สวัสดีคะ"ที่IDC_EDIT_RECEIVE

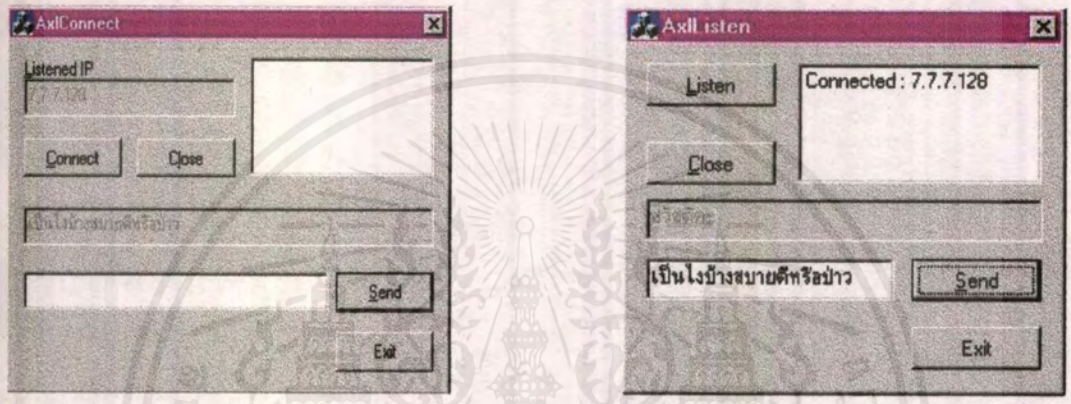
ฝั่ง Connect **ฝั่ง Listen**



รูปที่ 4.3 แสดงหน้าจอของฝั่ง Connect และฝั่ง Listen ขณะที่ฝั่ง Connect ส่งข้อความไปยังฝั่ง Listen

ขั้นที่ 4 ให้ฝั่ง Listen ลองส่งข้อความว่า "สบายดีหรือป่าวคะ" เมื่อกดปุ่ม Send จะมีการเรียกฟังก์ชั่นOnButtonSend ซึ่งจะทำการรับข้อความจาก IDC_EDIT_SEND และทำการส่งข้อความนั้นไปให้ฝั่ง Connect ซึ่งฝั่ง Connect จะปรากฏข้อความ "สบายดีหรือป่าวคะ" ที่ IDC_EDIT_RECEIVE จะปรากฏหน้าต่างการทำงานของทั้งสองฝั่งดังนี้

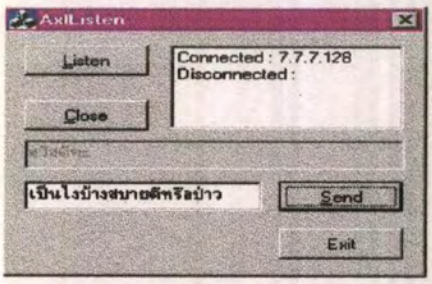
ฝั่ง Connect **ฝั่ง Listen**



รูปที่ 4.4 แสดงหน้าจอของฝั่ง Connect และฝั่ง Listen ขณะที่ฝั่ง Listen ส่งข้อความไปยังฝั่ง Connect

ขั้นที่ 5 ที่ฝั่ง Connect เมื่อกดปุ่ม Close จะมีการเรียกใช้ฟังก์ชั่น OnButtonClose ที่ฟังก์ชั่นนี้จะ ยกเลิกการเชื่อมต่อระหว่างเครื่องฝั่ง Connect และฝั่ง Listen และทำการปิดหน้าต่างการทำงานโดยอัตโนมัติ ส่วนฝั่ง Listen จะปรากฏหน้าต่างการทำงานดังต่อไปนี้

ฝั่ง Listen



รูปที่ 4.5 แสดงหน้าจอของฝั่ง Listen ภายหลังที่มีการยกเลิกการเชื่อมต่อจากฝั่ง Connect

จากผลการทดสอบแสดงว่าเราสามารถเชื่อมต่อเครื่องคอมพิวเตอร์สองเครื่องโดยผ่านโปรโตคอล TCP/IP ได้สมบูรณ์ และสามารถส่งข้อความสื่อสารระหว่างเครื่องคอมพิวเตอร์สองเครื่องได้อีกด้วย

4.2.2 การทดลองที่ 2 ทดสอบการจับหน้าจอและแสดงหน้าจอ (Capture & Display Desktop Image) โดยใช้ Port เพียงพอร์ตเดียว

ซึ่งในการจับหน้าจอนั้น ได้ออกแบบวิธีการต่างๆไว้ใน class "CDib" คำว่า DIB นั้นย่อมาจาก Device-Independent Bitmap หมายถึง bitmap ที่ไม่ขึ้นกับ device (มักหมายถึง Output device เช่น Screen, Printer) ซึ่ง operation ที่สำคัญๆก็คือ Capture(), PasteImage(), Read(), Write() ซึ่งมีรายละเอียดดังนี้

int CDib::Capture(**const CRect** &rect, **CDC** *pdcSrc = NULL);

จะทำการจับหน้าจอจาก argument DC (pdcSrc) ถ้าเป็น NULL จะใช้ DC ของ desktop โดยจับหน้าจอในขอบเขตของ (rect.left, rect.top, rect.right, rect.bottom)

Capture จะสร้าง Handle ของ bitmap ขึ้นมา แล้วนำไปใส่ไว้ใน DC ว่างๆ หลังจากนั้นก็จะทำ BitBlt (Bit Block Transfer) จาก pdcSrc มาใส่ไว้ใน DC ที่สร้างขึ้นมา แล้วจึงสร้าง BITMAPINFO เพื่อใช้กับ operation ต่างๆ เช่น การ Save การ Draw และนำ BITMAPINFO นั้นไปใช้ในการรับค่า DIBits (คือตัว bytes ของ image จริงๆ) อีกที

int CDib::PasteImage(**const CSize** &sizeSource, **CDib** &dib, **CRect** &rectPaste);

จะทำการวาด argument dib ลงไปใน ตัวมัน โดยวาดลงในกรอบ rectPaste ซึ่ง sizeSource จะใช้สำหรับกำหนดว่า DIB ปัจจุบันมีขนาดเท่าใด

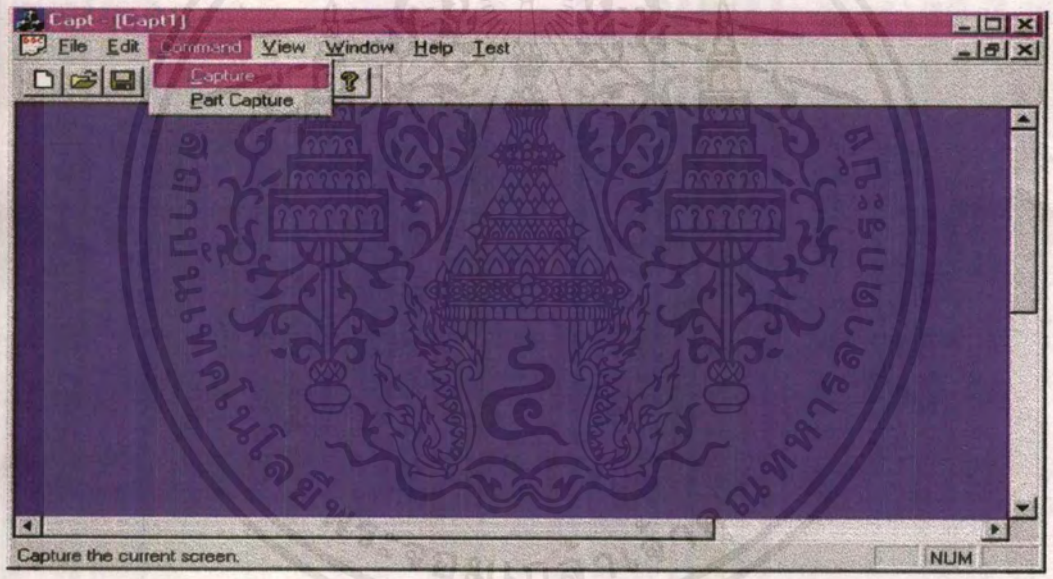
PasteImage จะสร้าง Handle ของ bitmap ขึ้นมา แล้วนำไปใส่ไว้ใน DC ว่างๆ หลังจากนั้นก็จะทำการวาดตัวเอง และ argument dib ลงบน dc นั้น แล้วจึงทำการจับภาพจาก dc ที่สร้างขึ้นนั้น

การ Write / Read

ใน bitmap file จะมี format ที่แตกต่างกันหลายชนิด ซึ่งในการทำ persistant ของ bitmap image นี้จะมี format ดังนี้

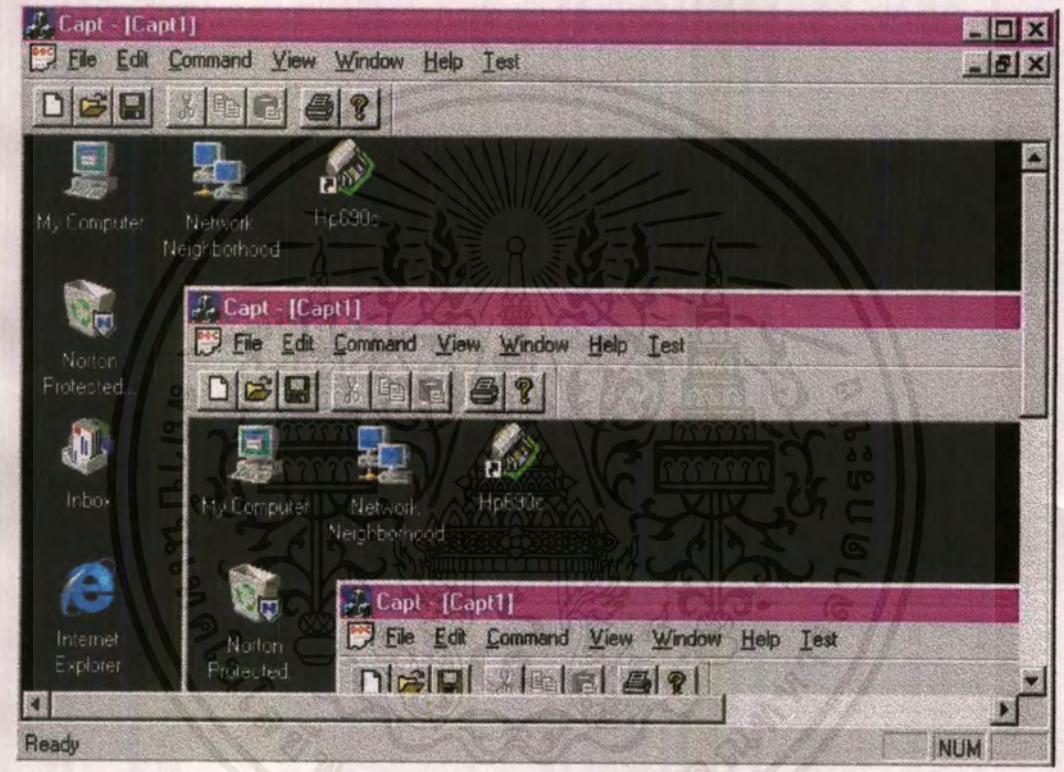
1. ส่วน BITMAPFILEHEADER จะเก็บข้อมูลเกี่ยวกับชนิด, ขนาด, และลักษณะของข้อมูลที่จะเก็บ
2. ส่วน BITMAPINFOHEADER จะเก็บข้อมูลเกี่ยวกับ มิติ, format ของสี ของ DIB
   ส่วน Image คือตัวค่าสีจริงๆของ image นั้นๆ

การทดสอบ เริ่มต้นด้วยการรันโปรแกรมที่ชื่อ Capt จะปรากฏหน้าต่างการทำงานดังนี้



รูปที่ 4.6 แสดงหน้าจอเริ่มต้นเมื่อรันโปรแกรม Capt.exe

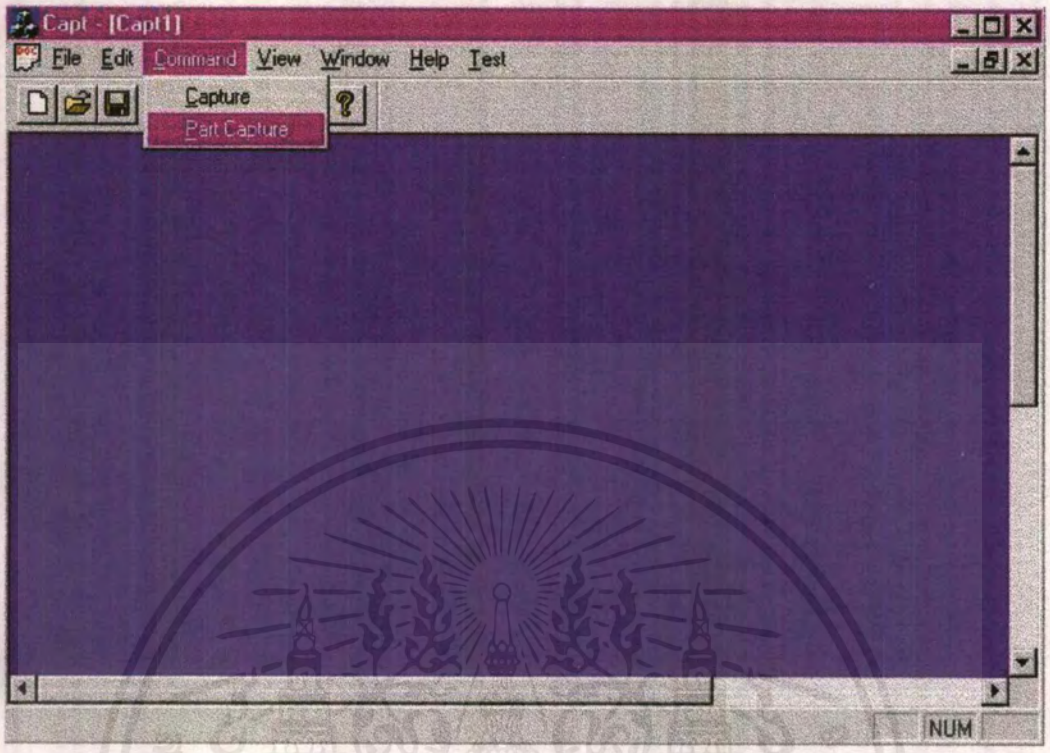
ถ้าไปที่เมนู Command เลื่อนแถบสีไปที่ Capture จะเป็นการเรียกฟังก์ชั่น OnCommandCaputre ที่ฟังก์ชั่นนี้จะมีการกำหนดขอบเขตในการ Capture และจะทำการ Capture Desktop Image ภายในขอบเขตที่ได้กำหนดไว้ ในที่นี้ได้กำหนดให้ Capture ทั้ง Screen ซึ่งจะปรากฏหน้าต่างการทำงานดังนี้



รูปที่ 4.7 แสดงหน้าจอที่เปลี่ยนแปลงเมื่อเลือกทำคำสั่ง Capture

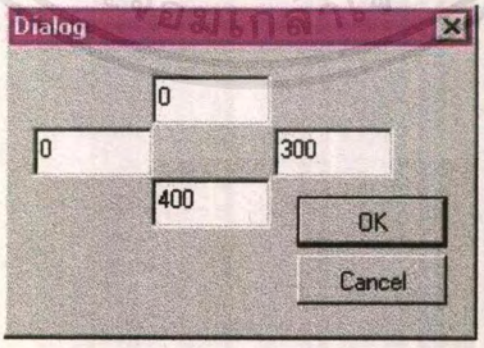
หรือ

ถ้าทำการเลื่อนแถบสีไปที่ Part Capture ดังรูป



รูปที่ 4.8 แสดงหน้าจอที่เปลี่ยนแปลงเมื่อเลือกทำคำสั่ง Part Capture

จะเป็นการเรียกใช้ฟังก์ชั่น OnCommandPartCapture ที่ฟังก์ชั่นนี้ขั้นแรกจะมีการเรียกให้แสดง Dialog ดังนี้



รูปที่ 4.9 แสดง Dialog ให้ใส่ค่าขอบเขตของจอภาพที่ต้องการจะ Capture

ที่ Dialog นี้เป็นการใส่ค่าขอบเขตของหน้าจอที่ต้องการ Capture โดยที่

- สี่เหลี่ยมด้านบนให้ใส่จุดบนสุดของขอบเขตที่ต้องการ Capture
- สี่เหลี่ยมด้านซ้ายให้ใส่จุดซ้ายสุดของขอบเขตที่ต้องการ Capture
- สี่เหลี่ยมด้านขวาให้ใส่จุดขวาสุดของขอบเขตที่ต้องการ Capture
- สี่เหลี่ยมด้านล่างให้ใส่จุดล่างสุดของขอบเขตที่ต้องการ Capture

ในที่นี้ได้กำหนดขอบเขต (0 , 0 , 300 , 400) ซึ่งเมื่อกด OK จะมีการเรียกฟังก์ชั่น PastImage เพื่อทำการวาดภาพใน view ซึ่งจะปรากฏหน้าต่างการทำงานดังนี้

รูปที่ 4.10 แสดงภาพหน้าจอที่ Capture โดยมีขนาดของภาพเท่ากับที่ได้กำหนดขอบเขตเอาไว้

จากผลการทดสอบเราสามารถจับหน้าจอของวินโดวส์ได้ทั้งแบบเต็มหน้าจอและแบบกำหนดขอบเขตหน้าจอ ซึ่งภาพที่ได้ทำการ Capture ไว้เราสามารถแสดงภาพ Desktop Image ได้อย่างถูกต้อง และมีประสิทธิภาพ

4.2.3 <u>การทดลองที่ 3</u> ทดสอบการส่ง Mouse Event และ Keyboard Event โดยใช้ Port เพียงพอร์ตเดียว

จากการที่เราต้องไปควบคุมการทำงานของเมาส์ และคีย์บอร์ด ของเครื่องฝั่งถูกควบคุม ซึ่งเป็นการสังเคราะห์ Input event ให้เกิดขึ้นที่ฝั่งถูกควบคุม ซึ่งการสังเคราะห์ Input event ในที่นี้จะหมายถึงการสร้าง Input Event ที่ไม่ได้เกิดตามธรรมชาติของมัน (เช่นการ click mouse หรือ กด keyboard) ให้ Windows เป็นไปอย่างที่มันควรจะเป็นถ้าเกิด event นั้นๆขึ้น ซึ่งพบว่าการที่เราจะเข้าไปควบคุมการทำงานของเมาส์นั้นเราจะใช้ Mouse_Event และการที่เราจะเข้าไปควบคุมการทำงานของคีย์บอร์ดเราจะใช้ฟังก์ชั่น Keyboard_Event ซึ่งทั้งสองฟังก์ชั่นมีการเรียกใช้และมีพารามิเตอร์ดังต่อไปนี้

<u>Mouse_Event</u>

```
void mouse_event(DWORD dwFlags, DWORD dx, DWORD dy,
                 DWORD dwData, DWORD dwExtraInfo );
```

Parameter:

dwFlags → เป็น flags ที่บ่งบอกถึง event ของ mouse ที่จะสังเคราะห์ เช่น MOUSEEVENTF_LEFTDOWN หมายถึงการกดปุ่มซ้ายของ mouse

dx → ค่าผลต่างระหว่างตำแหน่ง x ใหม่กับตำแน่ง x เดิม

dy → ค่าผลต่างระหว่างตำแหน่ง y ใหม่กับตำแน่ง y เดิม

dwData → ใช้เมื่อ dwFlags เป็น MOUSEEVENTF_MOUSEWHEEL ใช้บอกถึงจำนวนการเคลื่อนไหวของการ wheel

dwExtraInfo → เป็นค่าเพิ่มเติมให้กับ event นี้ซึ่ง application สามารถรับค่านี้ได้จาก function GetMessageExtarInfo()

ด้วยฟังก์ชั่นนี้ทำให้เราสามารถเข้าไปควบคุมการทำงานของเมาส์ของเครื่องคอมพิวเตอร์อีกฝั่งหนึ่งได้อย่างถูกต้อง ตรงตามตำแหน่งที่ต้องการ

Keyboard_Event

void keybd_event(BYTE bVk, BYTE bScan, DWORD dwFlags, DWORD dwExtraInfo);

Parameter:

bVk → ค่า Virtual-Key code ซึ่งมีค่าอยู่ระหว่าง 1 – 254

bScan → ค่า hardware scancode

dwFlags → ค่าเพิ่มเติมสำหรับ event นั้นๆ เช่น KEYEVENTF_KEYUP เป็นการบอกว่า key นี้ได้ถูกปล่อยแล้ว

dwExtraInfo → เป็นค่าเพิ่มเติมให้กับ event นี้ซึ่ง application สามารถรับค่านี้ได้จาก function GetMessageExtarInfo()

ด้วยฟังก์ชั่นนี้ทำให้เราสามารถเข้าไปควบคุมการทำงานของคีย์บอร์ดของเครื่องคอมพิวเตอร์อีกฝั่งหนึ่งโดยสามารถพิมพ์ข้อความจากฝั่งควบคุมไปยังฝั่งที่ถูกควบคุมได้อย่างถูกต้องตรงตามความต้องการ

4.2.4 การทดลองที่ 4 ทดสอบการส่ง Desktop Image , Mouse Event และ Keyboard Event พร้อมกัน โดยผ่านพอร์ต 1 พอร์ต

การทดสอบ

ได้ลองทำการส่ง Desktop Image, Mouse Event และ Keyboard Event พร้อมกันซึ่งเกิดปัญหาคือขณะที่ฝั่ง Puppet ทำการส่ง Desktop Image ที่มีการเปลี่ยนแปลงมาให้กับฝั่ง Master พร้อมกับที่ฝั่ง Master มีการส่งการควบคุมคีย์บอร์ด และควบคุมเมาส์มาให้ฝั่ง Puppet จะเกิด Critical Section (ดังที่อธิบายในบทที่ 3) ซึ่งโปรแกรมทำงานได้ไม่สมบูรณ์ไม่สามารถควบคุมเมาส์ และคีย์บอร์ดพร้อมกับที่มีการส่ง Desktop Image ได้ จึงได้ทำการศึกษาพบว่าเกิดจากการที่เราใช้พอร์ตเพียงพอร์ตเดียวในการส่งแพคเก็ต จึงได้ลองทำการเปิดพอร์ตสองพอร์ตดังที่จะศึกษาในขั้นต่อไป

4.2.5 <u>การทดลองที่ 5</u> ทดสอบการส่ง Desktop Image , Mouse Event และ Keyboard Event พร้อมกันโดยผ่านพอร์ต 2 พอร์ต โดยให้พอร์ตแรก ส่งการทำงานของ Input Event (Mouse Event และ Keyboard Event ที่ส่งจากฝั่ง Master ไปยังฝั่ง Puppet) และพอร์ตที่สองส่ง Desktop Image ที่มีการเปลี่ยนแปลงจากฝั่ง Puppet ไปยังฝั่ง Master

<u>การทดสอบ</u> โปรแกรมที่พัฒนาขึ้นนี้มีสองส่วนคือ

1. โปรแกรมฝั่งควบคุม (โปรแกรมชื่อว่า **Master**)
2. โปรแกรมฝั่งถูกควบคุม(โปรแกรมชื่อว่า **Puppet**)

<u>ขั้นที่ 1</u> เริ่มต้นด้วยการรันโปรแกรมที่ชื่อ Master และโปรแกรมที่ชื่อว่า Puppet พร้อมกัน

<u>ฝั่ง Master</u> จะปรากฏหน้าต่างการทำงานดังต่อไปนี้

รูปที่ 4.11 แสดงหน้าจอของฝั่ง Master เมื่อทำการรันโปรแกรมที่ชื่อว่า Master

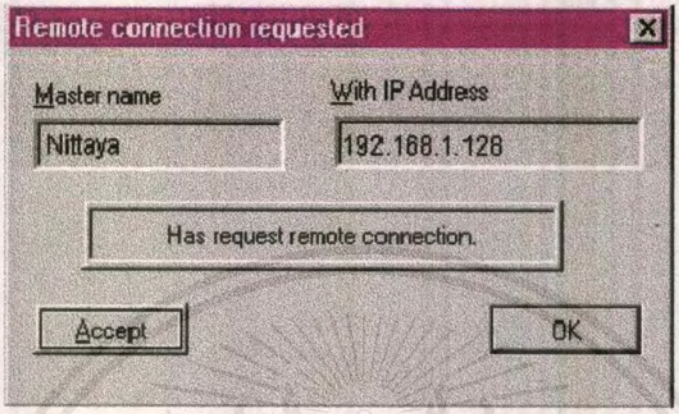
ฝั่ง Puppet เมื่อทำการรันโปรแกรมที่ชื่อ Puppet แล้วจะเปิดพอร์ทรอรับการเชื่อมต่อแต่จะยังไม่ปรากฏหน้าต่างการทำงานใดๆ จนกว่าจะมีการร้องขอการเชื่อมต่อเข้ามา

ขั้นที่ 2 ให้ฝั่ง Master เลือกทำการเชื่อมต่อกับเครื่องฝั่ง Puppet ในที่นี้จะขอเลือก "Nuno" เมื่อคลิกที่ไอคอนของ Puppet ที่ชื่อ "Nuno" จะปรากฏหน้าต่างการทำงานดังต่อไปนี้

ฝั่ง Master

รูปที่ 4.12 แสดงหน้าจอของฝั่ง Master ภายหลังที่ร้องขอการเชื่อมต่อกับฝั่ง Puppet ที่ชื่อว่า "Nuno"

ฝั่ง Puppet เมื่อฝั่ง Masterเข้ามาทำการร้องขอการเชื่อมต่อกับฝั่ง Puppet แล้วจะปรากฏหน้าต่างการทำงานดังต่อไปนี้



รูปที่ 4.13 แสดงหน้าจอของฝั่ง Puppet ขณะที่มีการร้องขอการเชื่อมต่อจากฝั่ง Master ที่ชื่อว่า Nittaya

จากรูปนี้ จะบอกถึง

Master Name ชื่อผู้ที่เข้ามาร้องขอการเชื่อมต่อ

IP Address หมายเลข IP ของฝั่ง Master ที่ต้องการเข้ามาควบคุมเครื่องฝั่ง Puppet

กดปุ่ม **Accept** แล้วกดปุ่ม **OK** จะเป็นการยอมรับการร้องขอการเชื่อมต่อนี้ ซึ่งจะทำให้เครื่องฝั่ง Master เข้ามาควบคุมเครื่องเราได้

แต่ที่ปุ่ม **Accept** เมื่อกดอีกครั้งจะเป็นปุ่ม **Decline** และจะเกิด **Edit Box** ให้ใส่เหตุผลที่ไม่ยอมรับการร้องขอการเชื่อมต่อได้ และเมื่อกดปุ่ม **OK** จะยกเลิกการร้องขอและออกจากโปรแกรม

ขั้นที่ 3 เมื่อฝั่ง Puppet ได้ยอมรับการร้องขอการเชื่อมต่อโดยกด Accept แล้ว จะปรากฏหน้าต่างการทำงานของฝั่ง Puppet ที่ view ของฝั่ง Master จะพบว่าหน้าต่างการทำงานของ Master และ Puppet เปลี่ยนแปลงไปดังต่อไปนี้

ฝั่ง Master จากรูปที่เราจะพบว่าที่ Menu เมื่อเลื่อนไปที่ Control Option จะสามารถเลือกได้ว่าเราจะควบคุม Mouse หรือ Keyboard หรือจะควบคุมทั้งสองอย่าง ซึ่งจะทำให้เราเสมือนว่าอยู่หน้าเครื่องคอมพิวเตอร์ที่ฝั่ง Puppet และสามารถทำการ Click Mouse หรือพิมพ์ข้อความใด ๆ ผ่าน Keyboard ก้อได้

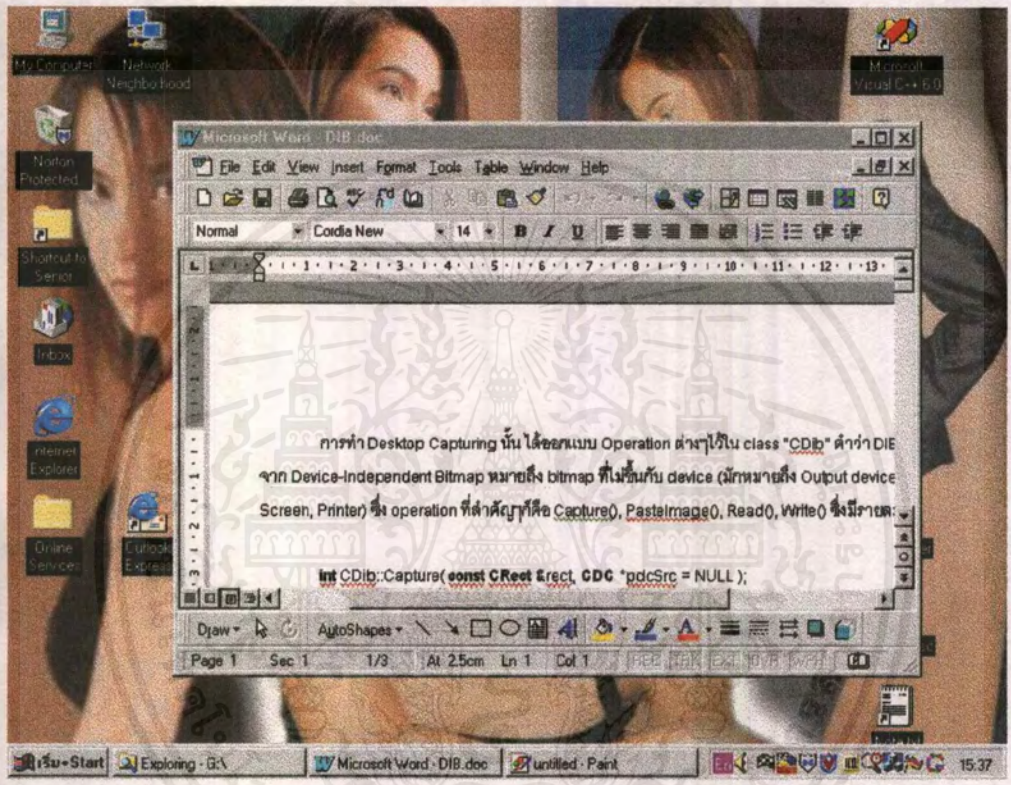
รูปที่ 4.14 แสดงหน้าจอของฝั่ง Master ภายหลังที่ฝั่ง Puppet กด Accept ยอมรับการเชื่อมต่อ

ฝั่ง Puppet เมื่อเครื่องฝั่ง Puppet ตอบรับการร้องขอการเชื่อมต่อจากฝั่ง Master แล้วจะปรากฏไอคอนของโปรแกรม Puppet ที่ System Tray ซึ่งจะบอกว่ามี Master ใดเข้ามาควบคุม และจะบอกหมายเลข IP ของMaster ที่เข้ามาควบคุม ดังตัวอย่างต่อไปนี้

รูปที่ 4.15 แสดงไอคอนของโปรแกรมที่ฝั่ง Puppet ที่อยู่ใน System Tray

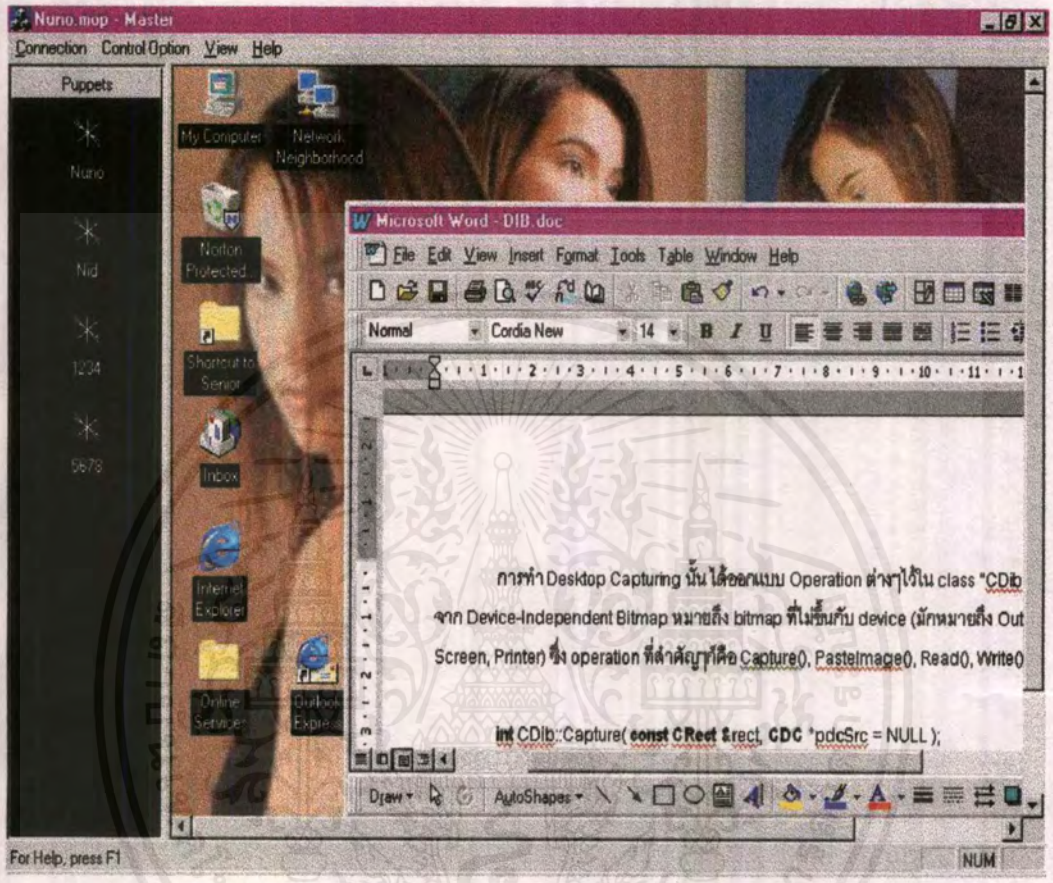
ขั้นที่ 4 ถ้ามีการเปลี่ยนแปลงหน้าต่างจอของฝั่ง Puppet จะมีการเปลี่ยนแปลงดังต่อไปนี้

ฝั่ง Puppet



รูปที่ 4.16 แสดงหน้าจอของฝั่ง Puppet เมื่อมีการเปลี่ยนแปลงไปจากเดิม

ฝั่ง Master



รูปที่ 4.17 แสดงหน้าจอของฝั่ง Master เมื่อหน้าจอของฝั่ง Puppet มีการเปลี่ยนแปลง

เมื่อเราต้องการยกเลิกการเชื่อมต่อกับฝั่ง Puppet ให้ไปที่Menu เลื่อนแถบสีไปที่ Connection และให้เลื่อนแถบสีไปที่ Disconnect จะเป็นการยกเลิกการเชื่อมต่อกับฝั่ง Puppet เป็นอันจบโปรแกรม

จากการทดสอบการทำงานของโปรแกรมทั้งฝั่ง Puppet และฝั่ง Master เราพบว่าฝั่ง Master สามารถควบคุมการทำงานของเมาส์และคีย์บอร์ดของฝั่ง Puppet ได้เป็นอย่างดีมีประสิทธิภาพ และที่ฝั่ง Puppet นั้นทุกครั้งที่มีการเปลี่ยนแปลงหน้าจอจะมีการส่งภาพที่มีการเปลี่ยนแปลงแล้วมายังฝั่ง Master ได้อย่างถูกต้อง ไม่มีผิดเพี้ยน.

จอภาพโปรแกรมฝั่ง Puppet

รูปที่ 4.18 แสดงหน้าจอของฝั่ง Puppet เมื่อต้องการเพิ่ม, ลบ, เปลี่ยนแปลง รายชื่อและรหัสผ่านของ Master

จากรูปนี้เป็นหน้าต่างการทำงานฝั่ง Puppet ที่ฝั่งนี้เราสามารถเพิ่มรายชื่อผู้ที่จะเข้ามาทำการควบคุมเครื่อง โดยใส่

Master Name คือ ชื่อของผู้ต้องการจะมาควบคุม

Password คือ รหัสผ่านของ Master Name นั้น

กดปุ่ม **Add** เราสามารถเพิ่ม Master ที่สามารถเข้ามาทำการติดต่อกับเครื่องฝั่ง Puppet ได้

กดปุ่ม **Remove** เราสามารถลบชื่อของ Master ออกจาก List ได้

กดปุ่ม **Update** เราสามารถเปลี่ยนแปลงรหัสผ่านของ Master Name นั้นได้

กดปุ่ม **Close** เป็นการออกจากโปรแกรม

บทที่ 5
บทสรุปและข้อเสนอแนะ

5.1 บทสรุป

ปัญหาพิเศษนี้ จัดทำเพื่อควบคุมคอมพิวเตอร์ระยะไกลผ่านเครือข่าย TCP / IP โดยใช้ Visual C++ 6.0 ในการเขียนโปรแกรม โดยสร้างโปรแกรมขึ้นมา 2 โปรแกรม คือ

1. โปรแกรมฝั่งควบคุม (Master)
2. โปรแกรมฝั่งถูกควบคุม (Puppet)

ได้ทำการทดสอบการทำงานของโปรแกรมทั้ง 2 โปรแกรมโดยเริ่มต้นให้ทำการรันโปรแกรม Master ที่ฝั่ง Master และให้รันโปรแกรม Puppet ที่ฝั่ง Puppet โดยฝั่ง Puppet จะเป็นตัวคอยรับการเชื่อมต่อจากตัว Master เมื่อฝั่ง Master ทำการร้องขอการเชื่อมต่อไปที่ฝั่ง Puppet ฝั่ง Puppet จะสามารถปฏิเสธการร้องขอโดยกด Decline แล้วจะยกเลิกการร้องขอนั้นแล้วออกจากโปรแกรม หรือสามารถกด Accept เพื่อยอมรับการร้องขอการเชื่อมต่อโดยที่ฝั่ง Puppet จะทำการ Capture Desktop Image และทำการส่งไปให้ฝั่ง Master ทำให้ฝั่ง Master เห็นหน้าจอการทำงานของฝั่ง Puppet เสมือนอยู่หน้าจอคอมพิวเตอร์ที่ฝั่ง Puppet ทำให้สามารถเข้าไปควบคุมเมาส์และคีย์บอร์ดของฝั่ง Puppet ได้ นอกจากนี้เมื่อ Desktop Image มีการเปลี่ยนแปลง จะมีการส่ง Desktop Image ที่มีการเปลี่ยนแปลงแล้วไปยังฝั่ง Master เพื่อให้ Master ได้เห็นหน้าจอที่ถูกต้อง ซึ่งจากการทดสอบการทำงานในขั้นแรกได้ทดสอบด้วยการใช้การ์ดแลนแบบ 10 MBPS เราพบว่าเราไม่สามารถส่งภาพหน้าจอการทำงานได้ แต่สามารถส่งการทำงานของเมาส์ได้ จึงได้ลองเปลี่ยนมาใช้การ์ดแลนแบบ 100 MBPS เราพบว่าเราสามารถส่งภาพ Desktop Image ได้อย่างถูกต้อง ,สามารถส่งการทำงานของเมาส์และคีย์บอร์ดได้อย่างถูกต้อง แต่มีปัญหาเมื่อเราทำการส่งภาพ Desktop Image ,การทำงานของเมาส์และคีย์บอร์ดพร้อมกัน เราพบว่าเกิดการชนกันของข้อมูล จึงได้ทำการศึกษาหาวิธีแก้ปัญหาโดยลองสร้างพอร์ตขึ้นมาสองพอร์ต (จากเดิมที่ใช้เพียงหนึ่งพอร์ต) โดยให้พอร์ตแรกเป็นพอร์ตที่ส่งการทำงานของเมาส์ และส่งการทำงานของคีย์บอร์ดจากฝั่ง Master ไปยังฝั่ง Puppet ส่วนพอร์ตที่สองจะจัดการส่งภาพ Desktop Image จากฝั่ง Puppet ไปยัง Master เมื่อเราได้ทำการทดลองส่งภาพ เมาส์และคีย์บอร์ดพร้อมๆ กันแล้วปรากฏว่าสามารถส่งได้ดีไม่พบกับปัญหาใด ๆ และยังได้ทดลองแก้ไขค่า Configuration ของเครื่องฝั่ง Puppet โดยทำการแก้ไขโดยเครื่องฝั่ง Master โดยลองเปลี่ยนภาพ Screensaver

ซึ่งสามารถแก้ไข เปลี่ยนแปลงได้ นอกจากนี้ยังได้ทดลองติดตั้งโปรแกรมจากฝั่ง Master ซึ่งพบว่าติดตั้งได้อย่างสมบูรณ์ ดังนั้นจะเห็นว่าปัญหาพิเศษนี้จะสามารถนำไปพัฒนาเพื่อนำมาประยุกต์ใช้ในการสอนคอมพิวเตอร์ในห้องปฏิบัติการทางคอมพิวเตอร์ได้

5.2 ข้อเสนอแนะ

1. ในการพัฒนาโปรแกรมนี้ควรใช้อุปกรณ์ที่มีประสิทธิภาพ เช่นควรใช้การ์ดแลนที่มีความเร็วสูง จากที่ได้ทำการทดลอง เดิมได้ใช้การ์ดแลนที่ความเร็ว 10 MBPS แต่ไม่สามารถส่งภาพได้ จึงได้ลองเปลี่ยนมาใช้การ์ดแลนที่ความเร็ว 100 MBPS พบว่าสามารถส่งข้อมูลต่างๆ ได้ดีรวดเร็วยิ่งขึ้น และควรจะใช้เครื่องที่มีหน่วยความจำ (RAM) สูง ๆ เพราะการส่ง Desktop Image ในแต่ละนั้นจะมีขนาดใหญ่ ถ้าใช้เครื่องคอมพิวเตอร์ที่มีหน่วยความจำน้อย ๆ จะทำให้ส่งภาพได้ช้ามาก ดังนั้นควรมีหน่วยความจำอย่างน้อย 64 MB

2. ผู้ที่ทำการควบคุมเครื่องคอมพิวเตอร์ควรมีจรรยาบรรณ ไม่ควรไปละเมิดสิทธิ์ในการเข้าไปจัดการระบบเกินสิทธิ์ที่ได้รับ

3. โปรแกรม Master และโปรแกรม Puppet จากเดิมที่ทำการเชื่อมต่อกันเพียงสองเครื่อง เราสามารถที่จะทำการพัฒนาเพื่อใช้ในการทำงานในระบบ LAN, ระบบ WAN หรืออาจจะพัฒนาโดยผ่านอินเตอร์เนตเพื่อติดต่อกันข้ามประเทศได้โดยติดต่อกันผ่าน TCP/IP

บรรณานุกรม

1. **Jesse Liberty , Teach Yourself More C++ in 21 days , SAMS Publishing,1996**
2. **Charles Ptzold, Programming Window, Microsoft Press ,1990**
3. **Richard C.Leinecker & Jamie Nye , Visual C++ Toolkit cutting – edge tools& technique for programming , Ventana Press, Inc 1995**
4. **David A.Holzgang, Teach Yourself ... Visual C++ 5.0 , MIS : Press and M&T Books,1997**
5. จิรพัฒน์ จันทร์เจิดศักดิ์และวีระ นพนิราพาธ , การเขียนโปรแกรมบน Microsoft Windows, สำนักพิมพ์ ซีเอ็ดยูเคชั่น จำกัด (มหาชน), พ.ศ.2521