โปรแกรมติดต่อสื่อสารแบบกราฟิกผ่านโปรโตคอล IPX

GRAPHICAL-INTERFACE COMMUNICATION PROGRAM VIA IPX

โดย

นายภาคภูมิ สุกษมรัตน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

โปรแกรมติดต่อสื่อสารแบบกราฟิกผ่านโปรโตคอล IPX

GRAPHICAL-INTERFACE COMMUNICATION PROGRAM VIA IPX

โดย

นายภาคภูมิ สุกษมรัตน์ 37014315

อาจารย์ที่ปรึกษา

อ.บุญชัย เรืองสุขนุกูล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมติดต่อสื่อสารแบบกราฟิกผ่านโปรโตคอล IPX

GRAPHICAL-INTERFACE COMMUNICATION PROGRAM VIA IPX

ผู้จัดทำ

นายภาคภูมิ สุกษมรัตน์ 37014315

อาจารย์ที่ปรึกษา

(อ. บุญชัย เรืองสุขนุกูล)

โปรแกรมติดต่อสื่อสารแบบกราฟิกผ่านโปรโตคอล IPX

Graphical-Interface Communication Program via IPX

โดย นายภาคภูมิ สุกษมรัตน์ 37014315

อาจารย์ที่ปรึกษา อาจารย์บุญชัย เรืองสุขนุกูล

บทคัดย่อ

โปรแกรมติดต่อสื่อสารแบบกราฟิกผ่านโปรโตคอล IPX เป็นโปรแกรมที่ทำให้ผู้ใช้บนระบบเครือข่ายท้องถิ่นโดยทั่วไป สามารถติดต่อสื่อสารกันได้โดยใช้โปรโตคอล IPX และการติดต่อกันแบบใช้รูปภาพระหว่างผู้ใช้ด้วยกันและระหว่างผู้ใช้กับโปรแกรม ผู้ใช้สามารถทำการสื่อสารได้โดยใช้การกดคีย์บอร์ดและเมาส์ ด้วยการลากเส้นหรือวาดรูปร่างต่างๆ ขึ้น เพื่อเป็นข่าวสารที่จะส่งไปยังผู้ใช้อีกคนหนึ่ง เส้นหรือรูปร่างที่ผู้ใช้วาดขึ้นนั้นจะไปปรากฏยังหน้าจอแสดงผลของอีกฝ่ายหนึ่ง โดยการจัดการรับส่งข้อมูลผ่านเครือข่ายของโปรแกรมทั้งสองฝั่ง โดยการรับส่งข้อมูลจะมีการตรวจสอบลำดับของข้อมูล และควบคุมการไหลของข้อมูล เพื่อให้ฝั่งรับสามารถทำงานได้ทัน การรับส่งข้อมูลสามารถทำได้ทั้งรับและส่งพร้อมกันได้

ABSTRACT

Graphical-Interface Communication Program via IPX is a program that provides communication among users in simple Local Area Network (LAN) via IPX protocol, with graphical interface between computer and user. Any user can communicate by using keyboard and mouse for drawing line as a message sent to another user. Lines or shapes written by user will be appear on the remote screen on ends, with the communication control and management of the both programs. Communication between two ends is controlled by data sequencing, error checking and flow control procedure of both sides. In this program, user can both receive and send data at the same time.

สารบัญ

สารบัญรูปภาพ

บทที่ 1
บทนำ

เนื่องจากในปัจจุบัน ระบบเครือข่ายคอมพิวเตอร์ได้มีการพัฒนามากขึ้นและถูกใช้กันอย่างกว้างขวาง ไม่ว่าจะในบริษัทขนาดใหญ่ ขนาดเล็ก ในหน่วยงานของรัฐบาลและเอกชน รวมไปถึงในสถาบันการศึกษาต่างๆ ซึ่งระบบเครือข่ายที่เชื่อมโยงระหว่างคอมพิวเตอร์ภายในพื้นที่เล็กๆ เช่นภายในหน่วยงานๆ หนึ่ง เรียกว่า **ระบบเครือข่ายท้องถิ่น** (LAN:Local Area Network)

ระบบเครือข่ายท้องถิ่นที่ใช้กันส่วนใหญ่ภายในองค์กรนั้น จะใช้ระบบไคลเอนท์เซิร์ฟเวอร์ (Client-Server) ของ Novell ซึ่งเรียกว่า Novell Netware โดยระบบจะประกอบด้วยเซิร์ฟเวอร์ ซึ่งเป็นเครื่องที่เก็บข้อมูลและเป็นตัวกลางในการติดต่อสื่อสาร และ ไคลเอนท์ ซึ่งเป็นเครื่องลูกข่ายซึ่งมีจำนวนมาก ในระบบ Netware นี้ **เครื่องคอมพิวเตอร์แต่ละเครื่องจะติดต่อสื่อสารกันโดยใช้โปรโตคอล IPX/SPX**

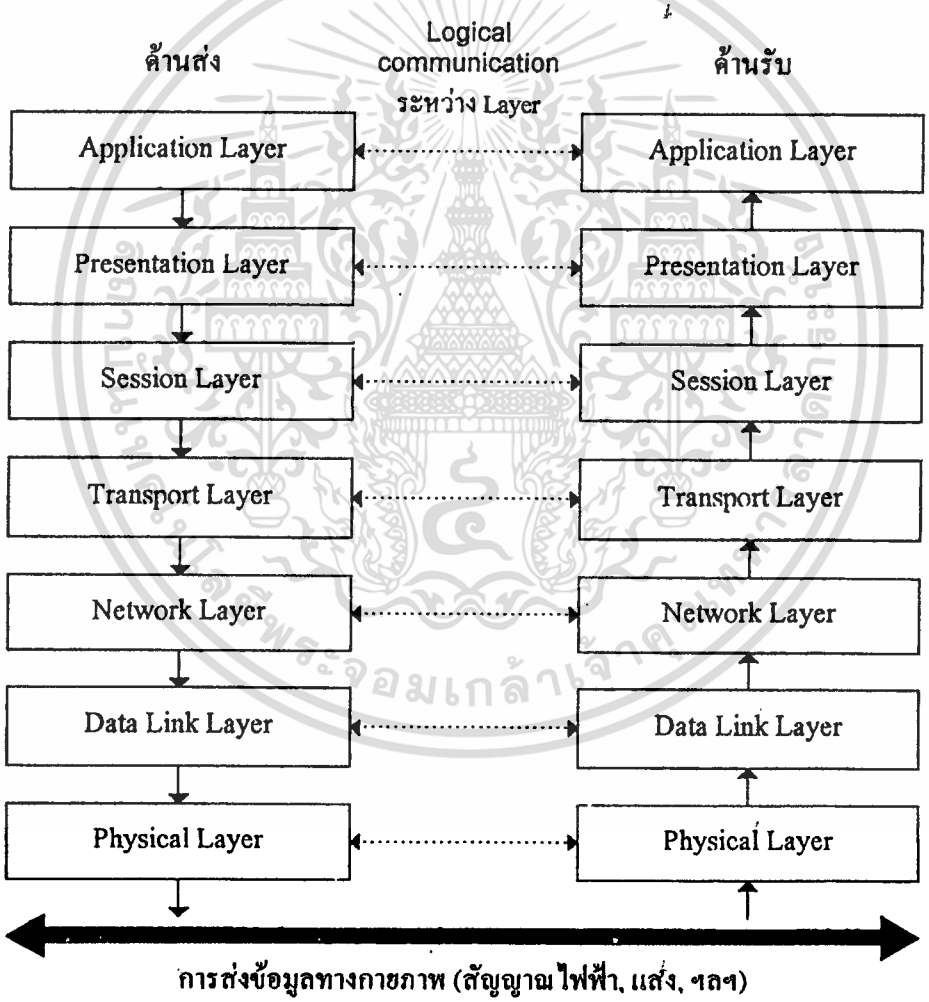
ในความเป็นจริง คอมพิวเตอร์ลูกข่ายแต่ละเครื่องในเครือข่าย สามารถติดต่อสื่อสารกันเองได้ ทั้งโดยการติดต่อสื่อสารกันเองและรับการช่วยเหลือการติดต่อจากเซิร์ฟเวอร์ โดยใช้โปรโตคอลดังกล่าว ซึ่งในกรณีติดต่อสื่อสารโดยการรับการช่วยเหลือจากเซิร์ฟเวอร์นั้น เซิร์ฟเวอร์จะมีแอพพลิเคชันที่ช่วยในการติดต่ออยู่แล้ว แต่ในกรณีการติดต่อกันเองนั้น โปรแกรมที่ใช้ยังมีอยู่น้อย

จุดประสงค์ของโครงงานนี้ก็เพื่อที่จะสร้างแอพพลิเคชันที่สามารถอำนวยความสะดวกให้กับผู้ใช้ภายในเครือข่าย ให้สามารถติดต่อสื่อสารกันได้อย่างมีประสิทธิภาพมากยิ่งขึ้น โดยใช้การแสดงผลและการติดต่อกับผู้ใช้แบบกราฟิก นอกไปจากนั้น ยังเป็นการศึกษาการสร้างการติดต่อบนเครือข่าย ขั้นตอนการติดต่อต่างๆ และการส่งสัญญาณ เนื่องจากโปรโตคอล IPX เป็นโปรโตคอลแบบ connectionless และไม่มีหมายเลขลำดับข้อมูล และในบางครั้งข้อมูลก็สูญหาย ในการทำโครงงานนี้จึงจำเป็นจะต้องสร้างรูปแบบการส่งข้อมูลโดยใช้โปรโตคอล IPX ที่ดีกว่าโปรโตคอล IPX เองขึ้นมาอีกทีหนึ่ง เพื่อให้การสื่อสารมีประสิทธิภาพมากยิ่งขึ้น

บทที่ 2
ทฤษฎีและการสร้าง

2.1 โอเอสไอโมเดล (OSI Model)

โอเอสไอโมเด็ล (OSI : Open System Interconnect) ถูกกำหนดเป็นมาตรฐานขึ้นมาโดยไอเอสโอ (ISO : International Standard Organization) เพื่อลดปัญหาเกี่ยวกับการสื่อสารกันระหว่างอุปกรณ์ต่างๆ และการติดต่อสื่อสารระหว่างเครือข่าย โอเอสไอโมเดล แบ่งออกเป็น 7 ชั้น โดยแต่ละชั้นจะทำงานตามหน้าที่ของมันและติดต่อกับชั้นที่อยู่สูงขึ้นไปและล่างลงมา ชั้นที่อยู่สูงจะทำหน้าที่ในการติดต่อกับผู้ใช้ แอพพลิเคชัน และการปฏิบัติงานต่างๆ ในขณะที่ชั้นล่างๆ จะทำหน้าที่เกี่ยวกับการส่งและควบคุมการส่งข้อมูล



รูป 2.1 - โครงสร้างของโอเอสไอโมเด็ล 7 ชั้น

จุดประสงค์ในการแบ่งโปรโตคอลออกเป็นชั้นๆ เช่นนี้ก็เพื่อแบ่งแยกงานของแต่ละส่วนอย่างชัดเจน ทำให้การออกแบบในแต่ละชั้นเป็นไปได้ง่าย สะดวกและเป็นอิสระ ยกตัวอย่างเช่น ในชั้นเดต้าลิงก์เลเยอร์ (Data Link Layer) และชั้นฟิสิคัลเลเยอร์ (Physical Layer) จะทำหน้าที่แยกกัน ชั้นฟิสิคัลเลเยอร์จะทำงานให้กับชั้นเดต้าลิงก์เลเยอร์ ซึ่งชั้นเดต้าลิงก์เลเยอร์ จะไม่ต้องสนใจว่าชั้นฟิสิคัลเลเยอร์ จะทำงานอย่างไร แต่จะสนใจแต่ว่า สามารถทำได้ตามหน้าที่ก็เพียงพอแล้ว ดังนั้น การพัฒนาในชั้นฟิสิคัลเลเยอร์จึงไม่มีผลกับชั้นเดต้าลิงก์เลเยอร์ และชั้นอื่นๆ ตราบใดที่ชั้นฟิสิคัลเลเยอร์ยังคงทำงานให้กับชั้นเดต้าลิงก์เลเยอร์ได้

จากรูป 2.1 จะเห็นว่าด้านส่งจะทำการส่งข้อมูลลงมาจากชั้นแอพพลิเคชันเลเยอร์ให้กับชั้นพรีเซนเตชันเลเยอร์ซึ่งชั้นพรีเซนเตชันเลเยอร์ จะทำการส่งข้อมูลต่อลงมาเรื่อยๆ จนมาถึงชั้นฟิสิคัล ดังนั้นในการสร้างชั้นแอพพลิเคชันขึ้นมา หากว่ามีชั้นพรีเซนเตชันเลเยอร์อยู่แล้วก็สามารถกระทำได้ง่ายโดยศึกษาเฉพาะการติดต่อกับชั้นพรีเซนเตชันเลเยอร์เท่านั้น ซึ่งจะทำให้การสร้างแอพพลิเคชันต่างๆ สะดวกขึ้นมาก

2.1.1 การทำงานในแต่ละชั้นของของโอเอสไอโมเด็ล

- ชั้นฟิสิคัลเลเยอร์ (Physical Layer) เป็นชั้นที่อยู่ล่างสุดของโอเอสไอโมเด็ลมีหน้าที่ทำงานเกี่ยวกับการส่งสัญญาณผ่านสื่อกลางไปยังเครื่องรับปลายทาง ข้อกำหนดเกี่ยวกับการส่งสัญญาณ เช่น การมอดูเลท การแทนข้อมูลด้วยสัญญาณแสง ไฟฟ้า หรือรูปแบบการส่งสัญญาณทางกายภาพใดๆ จะถูกกำหนดโดยชั้น ฟิสิคัลเลเยอร์ สายซีเรียล (Serial) และไฟเบอร์ออปติค เป็นตัวอย่างหนึ่งของชั้นฟิสิคัลเลเยอร์

- ชั้นเดต้าลิงก์เลเยอร์ (Data Link layer) ทำหน้าที่ควบคุมการส่งสัญญาณในชั้นฟิสิคัลเลเยอร์ โดยจะทำการรับส่งข้อมูลในรูปของเฟรม (frame) ในเฟรมจะประกอบด้วยข้อมูล เลขลำดับข้อมูล (sequence number) และ checksum เพื่อให้ด้านรับสามารถตรวจสอบความถูกต้องของข้อมูล การทำงานทางด้านรับ เมื่อได้รับข้อมูลแล้วจะมีการตอบรับ (acknowledge) กลับมายังฝั่งส่ง เพื่อยืนยันการไปถึงของข้อมูล รูปแบบการรับส่งข้อมูลและการตอบรับนั้นมีอยู่หลายรูปแบบ เช่น หยุดรอจนกว่าจะได้รับ packet (Stop and Wait) Go back n หรือ Selective Repeat เป็นต้น ในกรณีที่มีการใช้สื่อร่วมกัน จำเป็นต้องมีการควบคุมการเข้าถึงสื่อ (Medium Access Control) ซึ่งการทำงานนี้จะอยู่ในส่วนของเดต้าลิงก์เลเยอร์ เช่นกัน

- ชั้นเน็ตเวิร์กเลเยอร์ (Network Layer) เป็นชั้นที่ทำงานเกี่ยวกับการส่งข้อมูลผ่านเครือข่าย การส่งข้อมูลให้สามารถไปถึงจุดหมายได้ การหาเส้นทาง (Route) การทิ้งแพกเก็ต ควบคุมการไหลของข้อมูล (Flow control) การจอง buffer เพื่อรอข้อมูล ในชั้นเน็ตเวิร์กเลเยอร์ จะสนับสนุนการทำงานของชั้นทรานสปอร์ตเลเยอร์ ด้วยฟังก์ชันการส่งข้อมูล ซึ่งอาจเป็นทั้งแบบ Connectionless หรือ Connectioin-Oriented

- ชั้นทรานสปอร์ตเลเยอร์ (Transport Layer) มีหน้าที่ดูแลการส่งข้อมูลของชั้นเน็ตเวิร์กเลเยอร์ และชั้นต่ำกว่าลงไป ให้เป็นไปอย่างถูกต้อง รวมทั้งสนับสนุนชั้นเซสชันเลเยอร์ ด้วยการติดต่อสื่อสารข้อมูลแบบโลจิคอลระหว่างผู้ใช้บนเน็ตเวิร์ก และเป็นการสื่อสารที่ Transparent

- ชั้นเซสชันเลเยอร์ (Session Layer) ควบคุมการสื่อสารของโปรแกรมผ่านเครือข่าย ให้ข้อมูลมีลำดับที่ถูกต้อง ควบคุมการสื่อสารแบบสองทาง

- ชั้นพรีเซนเตชันเลเยอร์ (Presentation Layer) กำหนดรูปแบบของการแปลงข้อมูลที่ได้มาจากชั้นแอพพลิเคชันเลเยอร์ ให้เป็นข้อมูลที่มีรูปแบบใดแบบหนึ่ง ซึ่งสามารถเข้าใจได้โดยชั้นพรีเซนเตชันของฝั่งรับ รวมถึงการแปลงข้อมูลที่ได้รับมา ให้เป็นข้อมูลที่สามารถเข้าใจได้โดยชั้นแอพพลิเคชันเลเยอร์

- ชั้นแอพพลิเคชันเลเยอร์ (Application Layer) เป็นชั้นการทำงานของโปรแกรม และเป็นชั้นที่อยู่ติดกับผู้ใช้ ดังนั้นเมื่อข้อมูลผ่านชั้นนี้ขึ้นมา จะอยู่ในรูปแบบที่สามารถเข้าใจได้โดยผู้ใช้ ตัวอย่างเช่นโปรแกรม Browser, Telnet หรือโปรแกรมรับส่งเมล์ เป็นต้น

ระบบเครือข่ายโดยส่วนใหญ่ โปรแกรมที่ทำงานจะมีชั้นแอพพลิเคชันเลเยอร์ พรีเซนเตชันเลเยอร์ และเซสชันเลเยอร์เป็นของตัวเอง ซึ่งถึงแม้ว่าในชั้นเหล่านั้นจะเป็นมาตรฐาน แต่ก็ไม่สามารถจะเข้าถึงการทำงานในระดับต่ำกว่าแอพพลิเคชันเลเยอร์ของโปรแกรมได้ ส่วนชั้นต่ำกว่าทรานสปอร์ตลงไปนั้นมักจะเป็นของระบบ ดังนั้นการสร้างแอพพลิเคชันเพื่อสื่อสารข้อมูล จึงต้องมีการศึกษาการทำงานในชั้นที่มีอยู่ในระบบ เพื่อให้สามารถทำงานร่วมกันได้

2.2 IPX (Internetwork Packet Exchange)

IPX เป็นโปรโตคอลที่อยู่ในชั้นเน็ตเวิร์กเลเยอร์ และเป็นโปรโตคอลแบบคอนเน็กต์ชันเลส (Connectionless) ไม่มีการต่อเชื่อมเพื่อบอกเริ่มการติดต่อระหว่างเครื่องคอมพิวเตอร์แต่ละเครื่อง เครื่องคอมพิวเตอร์จะส่งข้อมูลออกไปยังฝั่งรับในรูปของ Packet เมื่อฝั่งรับได้รับข้อมูลจะมีการตอบรับกันระหว่างชั้นเดต้าลิงก์เลเยอร์ ดังนั้นฝั่งส่งจึงสามารถตรวจสอบได้ว่า ข้อมูลไปถึงหรือไม่ โดยดูจากการตอบรับ การรับข้อมูลของฝั่งรับ ฝั่งรับไม่สามารถลำดับข้อมูลได้ เนื่องจากโปรโตคอล IPX ไม่สนับสนุนการใช้เลขลำดับข้อมูล

2.2.1 MAC Address

MAC Address เป็นหมายเลขที่ใช้แทนที่อยู่ของเครื่องในเครือข่าย โดย MAC Address จะถูกกำหนดมาจากโรงงานผลิตเน็ตเวอร์กอแดปเตอร์แล้ว และไม่สามารถเปลี่ยนได้ โปรโตคอล IPX ใช้ MAC Address นี้ในการอ้างอิงที่อยู่ของเครื่องแต่ละเครื่องในเครือข่าย MAC Address มีความยาว 48 บิต (6 ไบต์) ซึ่งจะไม่ซ้ำกันอย่างแน่นอน เนื่องจากมีการตกลงกันระหว่างผู้ผลิตเน็ตเวอร์กอแดปเตอร์แล้ว

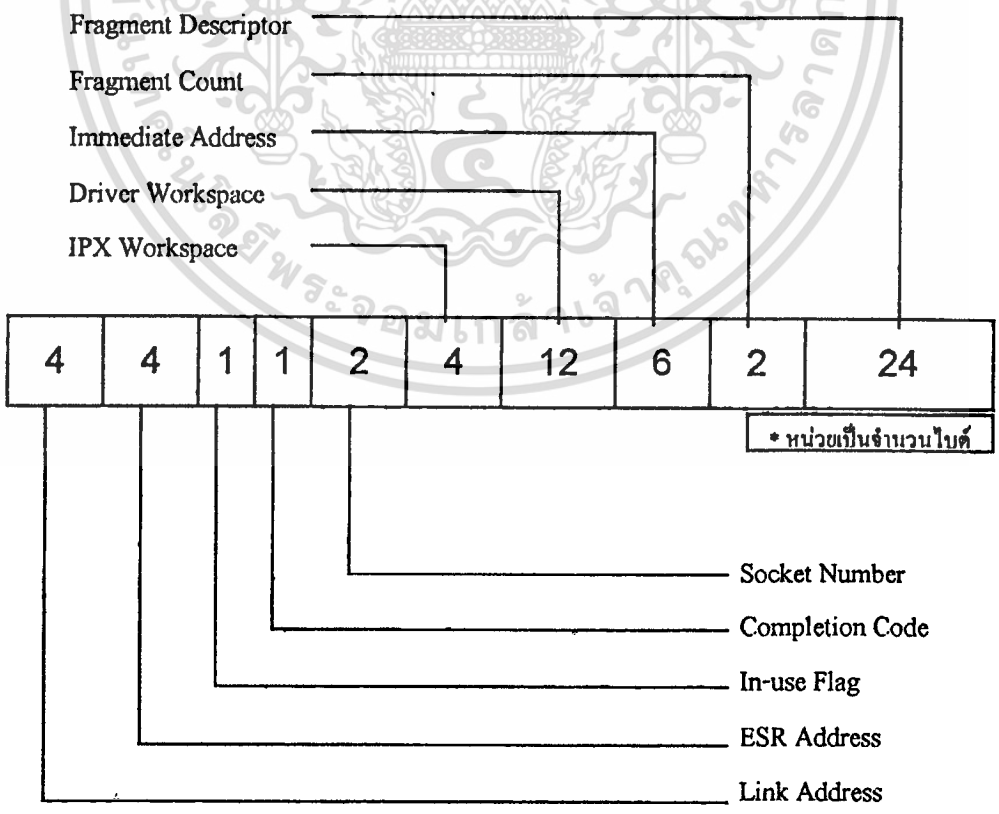
2.2.2 ซอกเก็ต (Socket)

เมื่อโปรแกรมต้องการจะรับส่งข้อมูลนั้น โปรแกรมจะต้องทำการเปิดซอกเก็ตขึ้นมาก่อน ซอกเก็ต จะเปรียบเสมือนช่องทางส่งและรับข้อมูล คล้ายๆ กับการแบ่งเป็นช่องสัญญาณข้อมูล เราสามารถใช้ซอกเก็ต เพื่อการแบ่งแยกข้อมูลข่าวสารแต่ละแบบออกจากกัน นอกจากนั้น ซอกเก็ต ยังมีประโยชน์ในกรณีที่มีแอพพลิเคชันหลายตัวต้องการสื่อสารพร้อมๆ กัน ข่าวสารที่ได้มาจะถูกแบ่งแยกตามหมายเลข ซอกเก็ต และส่งให้กับแอพพลิเคชันที่เปิดซอกเก็ต นั้นๆ อยู่ โดยแต่ละแอพพลิเคชันจะใช้ซอกเก็ตซ้ำกันไม่ได้

หมายเลขซอกเก็ต ของ IPX จะมีความยาว 16 bit ซึ่งหมายความว่า เรามีจำนวนซอกเก็ต ที่จะใช้กว่า 60,000 ซอกเก็ต ในการสื่อสารโดยโปรโตคอล IPX แต่เราจะใช้เฉพาะซอกเก็ตที่สูงกว่า 8000h เท่านั้น เนื่องจากซอกเก็ตที่ต่ำกว่านั้นบางซอกเก็ตถูกใช้โดยซอฟท์แวร์บางตัว

2.2.3 ECB (Event Control Block)

ECB (Event Control Block) เป็นบล็อกของข้อมูลชุดหนึ่ง ที่ใช้ในการรับส่งข้อมูลของ IPX ไดรเวอร์ โดยใน ECB จะประกอบด้วยข้อมูลที่จำเป็นและเป็นประโยชน์ในการสื่อสาร เช่น หมายเลข MAC Address ต้นทางและปลายทาง, หมายเลข ซอกเก็ต เป็นต้น IPX จะใช้ ECB 1 บล็อกในการรับส่งข้อมูล 1 แพกเก็ต และจะใช้จนกว่าการรับส่งข้อมูลแพกเก็ตนั้นสิ้นสุดลง ดังนั้นก่อนการรับส่งข้อมูล จึงจำเป็นที่จะต้องเตรียม ECB ไว้สำหรับ IPX ไดรเวอร์ เพื่อใช้ในการรับส่งข้อมูล โดย ECB จะมีรูปแบบดังรูป 2.2



รูป 2.2 - โครงสร้างของ ECB

จากโครงสร้างของ ECB ดังรูป ค่าต่างๆ ในบล็อกจะต้องถูกกำหนดโดยโปรแกรม ยกเว้นค่า Completion Code และ In-use Flag ซึ่งจะเป็นค่าที่ถูกเซ็ทโดยโปรโตคอล IPX เพื่อบอกถึงสถานะการทำงานในขณะนั้น โดยความหมายของแต่ละค่าของ Completion Code และ In-use Flag สามารถดูได้ในภาคผนวก

ส่วน Driver Workspace และ IPX Workspace นั้นเป็นส่วนที่ IPX ไดรเวอร์ใช้ในการทำงานบางอย่าง ซึ่งในขณะที่ ECB ถูกใช้เพื่อรอรับแพกเก็ตหรือส่งแพกเก็ต ไม่ควรมีการแก้ไขเปลี่ยนแปลงค่าใน Workspace ทั้งสองนี้

2.2.4 ESR (Event Service Routine)

ESR เป็นชุดคำสั่งชุดหนึ่ง ที่จะถูกเรียกโดย IPX ไดรเวอร์ เมื่อได้รับ packet โดยตำแหน่งของชุดคำสั่งนี้จะถูกระบุอยู่ใน ECB ในส่วนของ ESRAddress ซึ่งเป็นตำแหน่งของชุดคำสั่ง ESR ที่จะถูกเรียก ในกรณีที่ไม่จำเป็นหรือไม่ต้องการที่จะมีชุดคำสั่ง ESR จะต้องทำการกำหนดค่า ESRAddress ใน ECB ให้เป็น NULL เพื่อป้องกันการกระโดดไปทำงานในตำแหน่งที่ไม่ต้องการ

2.2.5 Fragment Count และ Fragment Descriptor

ในการส่งข้อมูลแต่ละครั้ง เราอาจจะให้โปรโตคอล IPX ทำการรวบรวมข้อมูลหลายๆ ส่วนจากหลายๆ ตำแหน่งในหน่วยความจำมาเพื่อทำการส่งรวมกันในแพกเก็ตเดียว ซึ่ง Fragment count จะเป็นค่าที่บอกว่า ข้อมูลนั้นมีกี่ส่วน ยกตัวอย่างเช่น ข้อมูลที่ต้องการส่งมักจะแบ่งเป็น 2 ส่วน คือ หัวของแพกเก็ต (Header) และข้อมูลที่เราต้องการส่ง ซึ่งไม่ได้อยู่ติดกัน ดังนั้นจะใช้ Fragment count เป็นตัวบอกว่าข้อมูลนั้นแบ่งเป็น 2 ส่วน และใช้ Fragment Descriptor เป็นตัวบอกตำแหน่งและขนาดของข้อมูลทั้งสองส่วน ซึ่ง Fragment Descriptor จะมีโครงสร้างดังนี้

	ตำแหน่งของข้อมูล	ขนาดข้อมูล
Fragment Descriptor 1	4	2
Fragment Descriptor 2	4	2
Fragment Descriptor 3	4	2
Fragment Descriptor 4	4	2

* หน่วยเป็นจำนวนไบต์

รูป 2.3 - โครงสร้างของ Fragment Descriptor ใน ECB

Fragment Descriptor ใน ECB จะประกอบด้วย Fragment Descriptor 4 ชุดด้วยกัน ซึ่งแต่ละชุดจะมีขนาด 6 ไบต์ ประกอบไปด้วยตำแหน่งของข้อมูล 4 ไบต์ และขนาดของข้อมูลอีก 2 ไบต์

2.2.6 IPX ไดรเวอร์

IPX ไดรเวอร์ เป็นโปรแกรมฝังตัว (Resident Program) ที่ทำให้โปรแกรมสามารถติดต่อรับส่งข้อมูลด้วยโปรโตคอล IPX ได้ โดยผ่านอินเตอร์รัพท์หมายเลข 7Ah ผู้ผลิดการ์ดเน็ตเวอร์กอแดปเตอร์ (Network adapter) ส่วนใหญ่จะให้ IPX ไดรเวอร์ มากับการ์ดด้วย ซึ่ง IPX ไดรเวอร์ จะทำให้โปรแกรมสามารถติดต่อโดยโปรโตคอล IPX ได้ โดยไม่ต้องอ้างอิงถึงชนิดของเน็ตเวอร์กอแดปเตอร์ ซึ่งทำให้โปรแกรมที่เขียนขึ้นมาสามารถทำงานได้กับเน็ตเวอร์กอแดปเตอร์ทุกรุ่นที่มี IPX ไดรเวอร์

IPX ไดรเวอร์ จะจัดหาฟังก์ชันการทำงานต่างๆ เพื่อสนับสนุนการติดต่อโดยโปรโตคอล IPX ให้กับแอพพลิเคชัน เช่นการเปิดซอกเก็ต การส่งแพกเก็ต เป็นต้น

2.2.7 ฟังก์ชันการทำงานของ IPX

เนื่องจากเป็นการติดต่อแบบคอนเน็กต์ชันเลส ฟังก์ชันการทำงานของ IPX จึงมีไม่มาก ในที่นี้จะยกมาเฉพาะเท่าที่จำเป็น โดยฟังก์ชันการทำงานของ IPX ทั้งหมดจะเรียกผ่านอินเตอร์รัพท์ 7Ah ยกเว้นฟังก์ชันในข้อ 2.2.7.1 เท่านั้น รีจิสเตอร์ BX จะเป็นตัวที่กำหนดหมายเลขฟังก์ชันที่ต้องการเรียก

2.2.7.1 <u>ฟังก์ชันบอกเริ่มการใช้งานโปรโตคอล IPX</u> (IPX Initialization) ฟังก์ชันนี้จะบอกให้ไดรเวอร์พร้อมในการทำงาน และตรวจสอบว่า IPX ไดรเวอร์ อยู่ในหน่วยความจำหรือไม่ ฟังก์ชันนี้สมควรที่จะถูกเรียกก่อนการเริ่มทำงานของโปรแกรมทุกครั้ง เพื่อตรวจสอบว่า IPX ไดรเวอร์ อยู่ในหน่วยความจำหรือไม่ เนื่องจากเป็นการทำงานโดยผ่านอินเตอร์รัพท์ของ IPX ไดรเวอร์ ถ้าหาก IPX ไดรเวอร์ ไม่อยู่ในหน่วยความจำ และเกิดการเรียกอินเตอร์รัพท์ขึ้น จะทำให้การทำงานผิดพลาดและอาจทำให้เครื่องหยุดการทำงานได้ การใช้งานของฟังก์ชันมีรายละเอียดดังนี้

-- อินเตอร์รัพท์หมายเลข 2Fh --

ค่าที่ต้องกำหนด	ค่าที่ได้กลับมาหลังจากการเรียกอินเตอร์รัพท์
AX = 7a00h	AL ถ้าเป็น 00h แสดงว่าไม่สามารถเริ่มการติดต่อได้

2.2.7.2 <u>ฟังก์ชันเปิดซอกเก็ต</u> (Open socket) เป็นการเปิด socket ขึ้นมาเพื่อการรับส่งข้อมูล เมื่อได้ทำการเปิดแล้วจะเปิดซ้ำอีกไม่ได้ และในกรณีที่มีแอพพลิเคชันตัวอื่นๆ ทำงานอยู่ แอพพลิเคชันนั้นก็จะไม่สามารถเปิด socket ที่ถูกเปิดไว้แล้วได้ด้วย ในส่วนของ AL ซึ่งเป็นการกำหนดระยะการใช้งาน ถ้าเป็นโปรแกรมธรรมดาจะใช้ 00h แต่ถ้าเป็นโปรแกรมประเภท resident ควรใช้เป็น 01h

-- อินเตอร์รัพท์หมายเลข 7Ah --

ค่าที่ต้องกำหนด	ค่าที่ได้กลับมาหลังจากการเรียกอินเตอร์รัพท์
BX = 0000h AL = ระยะการใช้งาน Socket ถ้าเป็น 00h จะเปิดจนกว่าจะถูกปิดหรือเทอร์มิเนต แต่ถ้าเป็น 01h จะเปิดจนกว่าจะถูกปิดเท่านั้น DX = หมายเลขซอกเก็ตที่ต้องการเปิด (ใช้เลขแบบ Big-endian) แต่ถ้าใช้หมายเลข 0000h หมายถึงการเปิดซอกเก็ตแบบไดนามิค (Dynamic Allocation)	AL = ค่าสถานะที่ได้กลับมา ถ้าเป็น - 00h หมายถึงการเปิดซอกเก็ตสำเร็จ ในกรณีที่ใช้การเปิดซอกเก็ตแบบไดนามิก ค่าหมายเลขซอกเก็ตจะถูกเก็บอยู่ใน DX - FEh ตารางซอกเก็ต (Socket table) เต็ม - FFh ซอกเก็ตถูกใช้อยู่แล้ว

2.2.7.3 ฟังก์ชันปิดซอกเก็ต (Close socket) เป็นการปิด socket เพื่อปล่อยให้ socket นั้นว่าง ในการจบการทำงานของโปรแกรมแต่ละครั้งควรจะมีการปิด socket ที่เปิดไว้ทั้งหมด ถึงแม้ว่าในความจริง โปรโตคอล IPX จะทำการปิดให้เองก็ตาม แต่ก็ควรจะปิดด้วยคำสั่งนี้ก่อนการจบโปรแกรมด้วย เพื่อป้องกันการเปิดซอกเก็ตค้างโดยไม่ได้ใช้ ซึ่งจะทำให้แอพพลิเคชันตัวอื่นๆ ไม่สามารถใช้งาน socket นั้นๆ ได้

-- อินเตอร์รัพท์หมายเลข 7Ah --

ค่าที่ต้องกำหนด	ค่าที่ได้กลับมาหลังจากการเรียกอินเตอร์รัพท์
BX = 0001h DX = หมายเลขซอกเก็ตที่ต้องการปิด (ใช้เลขแบบ big-endian)	ไม่มี

2.2.7.4 ฟังก์ชันส่งแพกเก็ต (Send packet) ใช้ส่งแพกเก็ตไปให้ยังปลายทาง โดยจะต้องกำหนดค่าต่างๆ ใน ECB ให้ครบถ้วนเสียก่อน มิฉะนั้นอาจทำให้เกิดความผิดพลาดและระบบหยุดทำงานได้ หมายเลข socket ที่ใช้ต้องเป็นหมายเลข socket ที่ได้เปิดไว้แล้ว

-- อินเตอร์รัพท์หมายเลข 7Ah --

ค่าที่ต้องกำหนด	ค่าที่ได้กลับมาหลังจากการเรียกอินเตอร์รัพท์
BX = 0003h ES:SI = ตำแหน่งของ ECB ที่จะใช้ในการส่ง	ไม่มี

2.2.7.5 ฟังก์ชันรอแพกเก็ต (Listen for packet) ใช้ในการรับแพกเก็ตโดยจะต้องส่งตำแหน่งของ ECB ที่ได้กำหนดค่าทุกอย่างครบถ้วนเสียก่อนเช่นเดียวกับฟังก์ชันส่งแพ็กเก็ต

-- อินเตอร์รัพท์หมายเลข 7Ah --

ค่าที่ต้องกำหนด	ค่าที่ได้กลับมาหลังจากการเรียกอินเตอร์รัพท์
BX = 0004h ES:SI = ตำแหน่งของ ECB ที่จะใช้ในการส่ง	AL เป็นค่าสถานะ ถ้าเป็น 00h หมายถึงสำเร็จ แต่ถ้าเป็น FFh หมายถึงไม่มี socket ดังที่กำหนดใน ECB เปิดอยู่

2.2.7.6 ฟังก์ชันยกเลิกเหตุการณ์ (Cancel event) ใช้ในกรณีที่ต้องการยกเลิกการส่งแพกเก็ตหรือการรอแพกเก็ต ซึ่งใช้ตำแหน่งของ ECB เป็นตัวบ่งบอกเหตุการณ์ที่ต้องการยกเลิก ฟังก์ชันนี้ไม่สามารถยกเลิกการส่งแพกเก็ตที่ได้ทำการส่งออกไปแล้วได้

-- อินเตอร์รัพท์หมายเลข 7Ah --

ค่าที่ต้องกำหนด	ค่าที่ได้กลับมาหลังจากการเรียกอินเตอร์รัพท์
BX = 0006h ES:SI = ตำแหน่งของ ECB	AL เป็นค่าสถานะ - 00h สำเร็จ - F9h ECB นั้นกำลังถูกดำเนินการอยู่ - FCh เหตุการณ์ถูกยกเลิกไปก่อนหน้านี้แล้ว - FFh ไม่สำเร็จ, ไม่มีเหตุการณ์นั้นจริง หรือ ECB ไม่เป็นที่เข้าใจ

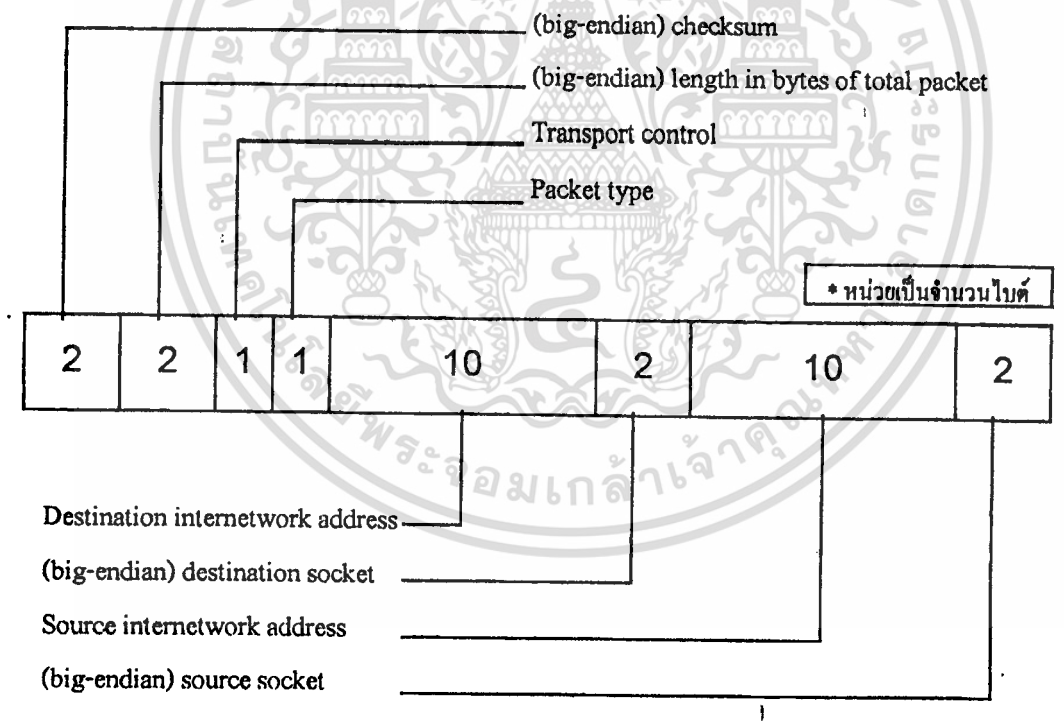
2.2.7.7 ฟังก์ชันขออินเตอร์เน็ตเวอร์กแอดเดรส (Get Internetwork Address) ใช้ในกรณีที่ต้องการทราบว่า อินเตอร์เน็ตเวอร์กแอดเดรสของเครื่องที่กำลังทำงานอยู่เป็นเท่าไร โดยอินเตอร์เน็ตเวอร์กแอดเดรสจะใช้ในการติดต่อกันระหว่างเครือข่าย มีขนาด 10 ไบต์ โดยแบ่งเป็นแอดเดรสของเครื่อง (Node Address) 6 ไบต์ กับแอดเดรสของเครือข่าย (Network Address) อีก 4 ไบต์

-- อินเตอร์รัพท์หมายเลข 7Ah --

ค่าที่ต้องกำหนด	ค่าที่ได้กลับมาหลังจากการเรียกอินเตอร์รัพท์
BX = 0009h ES:SI = ตำแหน่งของบัฟเฟอร์ที่ต้องการเก็บอินเตอร์เน็ตเวอร์กแอดเดรสไว้	ค่าอินเตอร์เน็ตเวอร์กแอดเดรสที่ตำแหน่งที่กำหนดใน ES:SI ก่อนเรียกอินเตอร์รัพท์

2.2.8 IPX เฮดเดอร์

IPX เฮดเดอร์ เป็นส่วนที่สำคัญในการส่งแพกเก็ต แพกเก็ตทุกๆ แพกเก็ตที่ส่งออกไปจะต้องมีเฮดเดอร์เป็นส่วนประกอบอยู่ในส่วนหัวของข้อมูล เฮดเดอร์จะประกอบด้วยข้อมูลที่จำเป็นในการสื่อสาร เช่น จุดหมายปลายทาง หรือชนิดของแพกเก็ต โดย IPX เฮดเดอร์จะมีโครงสร้างดังนี้



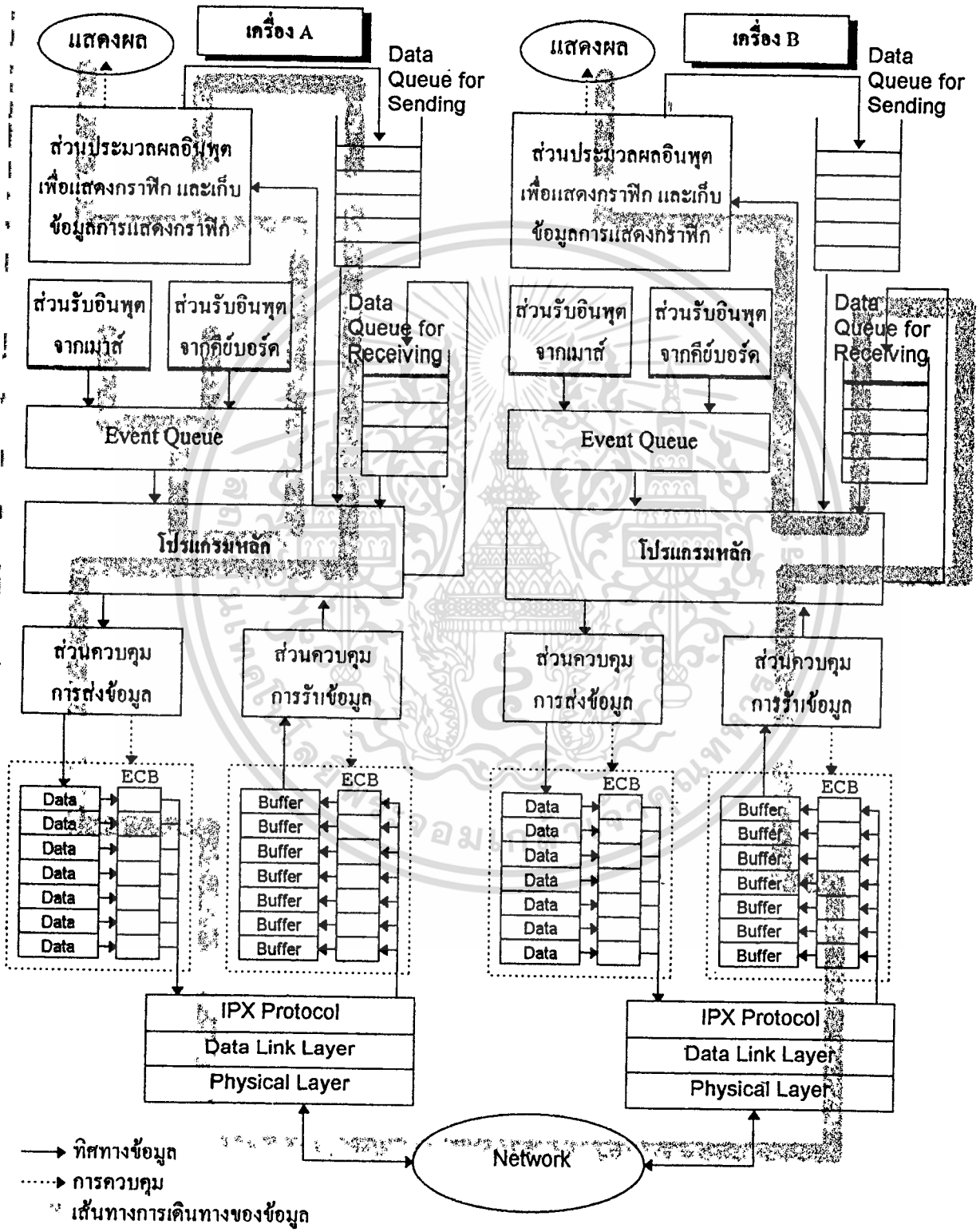
รูปที่ 2.4 - โครงสร้างของ IPX เฮดเดอร์

ในการส่งข้อมูลแต่ละแพกเก็ต เฮดเดอร์นี้จะต้องได้รับการกำหนดค่าที่ถูกต้อง ซึ่งส่วนที่จำเป็นต้องกำหนดก็คือส่วน Transport control, Packet type, Destination internetwork address และ Destination socket โดยรายละเอียดเกี่ยวกับ Transport control และ Packet type สามารถดูได้ในภาคผนวก

บทที่ 3

การออกแบบและการประยุกต์ใช้

3.1 โครงสร้างระบบโดยรวม



รูปที่ 3.1 - โครงสร้างระบบโดยรวม

จากโครงสร้างระบบโดยรวม จะเห็นว่า โปรแกรมหลักจะมีหน้าที่คอยประสานงานระหว่างส่วนต่างๆ ทั้งการรับอินพุต การแสดงผล และการรับส่งข้อมูลจากส่วนควบคุมการรับส่งข้อมูล การทำงานทั้งหมดจะถูกดำเนินงานและจัดการโดยโปรแกรมหลัก โดยมีส่วนต่างๆ คอยช่วยเหลือสนับสนุน ในการอธิบายการทำงานในหัวข้อ 3.1.1 และ 3.1.2 นั้น จะสมมติให้เครื่อง A เป็นเครื่องที่ส่งข้อมูล และเครื่อง B เป็นเครื่องที่รับข้อมูล แต่ในทางกลับกัน เครื่อง B ก็สามารถส่งข้อมูลไปยังเครื่อง A ได้ ด้วยการทำงานที่เหมือนกัน

3.1.1 การทำงานของระบบทางด้านส่ง

ในการทำงานของระบบทางด้านส่ง (ในที่นี้คือเครื่อง A) ส่วนรับอินพุตจากเมาส์และคีย์บอร์ด จะทำหน้าที่สร้างข่าวสารที่เรียกว่าอีเวนท์ (Event) ขึ้น โดยอีเวนท์จะบอกถึงลักษณะของอินพุตของผู้ใช้ ข้อมูลที่ได้รับและตำแหน่งที่เกิดอีเวนท์ หลังจากสร้างแล้ว ก็จะนำอีเวนท์ดังกล่าวไปเก็บไว้ในอีเวนท์คิว (Event queue) เพื่อให้โปรแกรมหลักทำการดึงเอาข้อมูลไปดำเนินการ

เมื่อโปรแกรมหลักได้ดำเนินการกับอีเวนท์นั้น และเห็นว่าควรมีการแสดงผล โปรแกรมหลักจะส่งอีเวนท์นั้นไปยังส่วนประมวลผลเพื่อแสดงกราฟิก และเก็บข้อมูลกราฟิก ในส่วนนี้จะทำการพิจารณาอีเวนท์ที่ได้และทำการแสดงผลออกทางหน้าจอ ข้อมูลกราฟิก คือข้อมูลที่ใช้ในการส่ง เพื่อบอกอีกฝั่งถึงการแสดงกราฟิก ในส่วนนี้จะทำการเก็บข้อมูลดังกล่าวไว้ในคิวข้อมูล (Data queue) ที่ใช้สำหรับการส่ง เพื่อให้โปรแกรมหลักดำเนินการ

เมื่อโปรแกรมหลักพิจารณาแล้วเห็นว่า มีข้อมูลอยู่ในคิวข้อมูลที่ใช้ในการส่ง โปรแกรมหลักจะทำการตรวจสอบความเป็นไปได้ในการส่งข้อมูล หากส่วนควบคุมการส่งข้อมูลตอบรับว่า สามารถส่งข้อมูลได้ โปรแกรมหลักจะทำการดึงข้อมูลออกมาจากคิวข้อมูล เพื่อทำการส่งไปยังอีกฝั่งหนึ่ง โดยผ่านส่วนควบคุมการส่งข้อมูล ส่วนควบคุมการส่งข้อมูลจะทำงานโดยผ่าน ECB และฟังก์ชันการทำงานระดับต่ำของ IPX เพื่อส่งข้อมูลผ่านไปยังอีกฝั่งหนึ่ง ซึ่งตำแหน่งอินเตอร์เน็ตเวิร์กแอดเดรสที่เป็นที่อยู่ของเครื่องฝั่งรับจะถูกเก็บไว้ตั้งแต่การเริ่มการเชื่อมต่อ ในการส่งข้อมูลแต่ละชุด ส่วนควบคุมจะเพิ่มเลขลำดับข้อมูลเข้าไปหน้าข้อมูลด้วย เพื่อให้ฝั่งรับเรียงลำดับข้อมูลได้ถูกต้อง โปรแกรมหลักจะทำการส่งข้อมูลจนกว่าข้อมูลในคิวจะหมด หรือเมื่อส่วนควบคุมไม่สามารถส่งข้อมูลได้อีกต่อไป เมื่อข้อมูลไปถึงฝั่งรับ ซึ่งในที่นี้คือเครื่อง B ข้อมูลจะถูกเก็บอยู่ใน Buffer ของเครื่องทางฝั่งรับ

3.1.2 การทำงานของระบบทางด้านรับ

การทำงานของระบบทางด้านรับ (ในที่นี้คือเครื่อง B) โปรแกรมหลักจะทำการดึงเอาข้อมูลจากบัฟเฟอร์โดยผ่านส่วนควบคุมการรับข้อมูล ส่วนควบคุมการรับข้อมูลจะตรวจเช็คลำดับข้อมูล และนำเอาข้อมูลในลำดับที่ถูกต้องให้กับโปรแกรมหลัก โดยถอดเอาเลขลำดับข้อมูลออกก่อน โปรแกรมหลักจะดึงเอาข้อมูลจากบัฟเฟอร์ไปเก็บไว้ในคิวข้อมูลที่ใช้ในการรับข้อมูล จนกว่าข้อมูลในบัฟเฟอร์จะหมด หลังจากนั้น โปรแกรมก็จะดึงเอาข้อมูลจากคิวข้อมูล ส่งให้กับส่วนควบคุมการแสดงผลทีละชุด จนกว่าข้อมูลในคิวข้อมูลจะหมด เป็นอันจบขั้นตอนการทำงานหนึ่งรอบ

3.2 ฟังก์ชันเพื่อการติดต่อสื่อสารในระดับพื้นฐาน

การติดต่อโดยโปรโตคอล IPX จะต้องมีการเรียกใช้อินเตอร์รัพท์ ดังที่กล่าวไปแล้วในบทที่ 2 เนื่องจาก การเรียกใช้งานอินเตอร์รัพท์โดยตรงนั้นมีความซับซ้อน หากไม่มีการสร้างฟังก์ชันขึ้นมารองรับจะทำให้การพัฒนาและตรวจสอบโปรแกรมเป็นไปอย่างยากลำบาก จึงได้พัฒนาฟังก์ชันสำหรับการติดต่อในระดับพื้นฐานขึ้น เพื่อนำไปใช้ในการสร้างส่วนควบคุมการรับและส่งข้อมูล รายละเอียดของโปรแกรมสามารถดูได้ในภาคผนวกในส่วนของไฟล์ IPX.H

3.2.1 <u>การเริ่มการใช้งานโปรโตคอล IPX</u> จะต้องกระทำก่อนการเริ่มทำงานของโปรแกรมทุกครั้ง เพื่อตรวจเช็คว่ามี IPX ไดรเวอร์อยู่ในหน่วยความจำหรือไม่ และเป็นการบอกการเริ่มการใช้งานโปรโตคอล การเริ่มการใช้งานจะกระทำโดยผ่านอินเตอร์รัพท์หมายเลข 2Fh (อ้างอิงจากข้อ 2.2.7.1) โดยมีรูปแบบของฟังก์ชันดังนี้

```
int IPXInitial(void);
```

ค่าที่ฟังก์ชันส่งกลับมา ถ้าเป็น 0 หมายถึงไม่สามารถเริ่มการใช้งานได้ ถ้าเป็น 1 สามารถเริ่มการใช้งานได้ โดย ipxspx ที่กำหนดเป็นตัวแปรแบบโกลบอล จะถูกกำหนดให้ชี้ไปยังรูทีนของอินเตอร์รัพท์ 0x7a ดังนั้น ถ้าในกรณีที่ไม่สามารถใช้การอินเตอร์รัพท์ ก็อาจใช้การเรียกฟังก์ชันนี้แทนการเรียกการเรียกอินเตอร์รัพท์ได้

3.2.2 <u>การเปิดซอกเก็ต</u> จะกระทำผ่านอินเตอร์รัพท์หมายเลข 7Ah ฟังก์ชันหมายเลข 0000h โดยมีรูปแบบของเป็นฟังก์ชันดังนี้

```
unsigned int IPXOpensocket(unsigned int sock);
```

การใช้งานฟังก์ชัน จะต้องส่งผ่านค่า sock ซึ่งเป็นข้อมูลขนาด 2 ไบต์ ให้กับฟังก์ชัน เพื่อทำการเปิดซอกเก็ตนั้นๆ หากการเปิดซอกเก็ตสำเร็จ ค่าที่ได้กลับมาจะเป็นหมายเลขของซอกเก็ต ซึ่งอาจได้ใช้หากเปิดซอกเก็ตแบบไดนามิก (ใช้หมายเลขซอกเก็ตเป็น 0x0000) แต่ถ้าการเปิดไม่สำเร็จค่าที่ถูกส่งกลับมาจะเป็น 0

3.2.3 <u>การส่งแพกเก็ต</u> ในการส่งแพกเก็ต จำเป็นที่จะต้องมีการตระเตรียมบล็อก ECB ไว้ก่อนที่จะมีการส่ง ซึ่งบล็อก ECB จะสามารถเขียนเป็นโครงสร้างในรูปแบบของภาษา C ดังนี้

```
struct ECB
{
      void far *linkadd; /* link address */
      void (far *ESRaddress)();/* default should set to null */
      char inuseflag;
      char completioncode;
      unsigned int socket;
      char IPXworkspace[4];
      char driverworkspace[12];
      char immediateaddress[6];
      unsigned int fragmentcount;
      ECBFragment fragdesc[4];
};
```

โครงสร้างของ ECBFragment จะต้องถูกกำหนดอยู่ก่อนหน้าโครงสร้างของ ECB ซึ่งเขียนได้ดังนี้

```
struct ECBFragment
{
      void far *addr;
      unsigned int size;
};
```

การส่งแพกเก็ต จะต้องมีเฮดเดอร์ในส่วนแรกของข้อมูลเสมอ ซึ่งเขียนเป็นโครงสร้างดังนี้

```
struct IPXHeader
{
      int checksum; /* Data checksum */
      int length; /* Data length */
      char txcontrol; /* transport control */
      char packettype;
      NetInfo dest; /* Destination Infcrmation */
      NetInfo src; /* Source Information */
};
```

โครงสร้างในส่วนของ NetInfo จะต้องถูกกำหนดอยู่ก่อนหน้า IPXHeader โดยภายในจะประกอบด้วย โหนดแอดเดรส เน็ตเวอร์กแอดเดรส และหมายเลขซอกเก็ต มีรายละเอียดดังนี้

```
struct NetInfo
{
      NetAddr addr;
      unsigned int socket;
};
```

สำหรับฟังก์ชันในการส่งแพ็กเก็ตนั้น จะใช้การเรียกผ่านฟังก์ชันหมายเลข 0003h ซึ่งมีรูปแบบดังนี้

```
void IPXSendpacket(ECB far *ecbs);
```

ฟังก์ชันนี้จะทำงานได้ดังที่ต้องการ ก็ต่อเมื่อค่า ECB ได้ถูกกำหนดอย่างถูกต้องเหมาะสม

3.2.4 ฟังก์ชันการรอรับข้อมูล ใช้การเรียกผ่านฟังก์ชันหมายเลข 0004 h มีรายละเอียดดังนี้

```
char IPXListenforpacket(ECB far *ecbr);
```

โครงสร้างของ ECB จะใช้แบบเดียวกับการส่งข้อมูล ซึ่งค่าที่จำเป็นต้องกำหนดใน ECB ก่อนการเรียกฟังก์ชันก็คือ ESR address, หมายเลขซอกเก็ต, fragment count และ fragment descriptor

ถ้าหากฟังก์ชันทำงานสำเร็จจะส่งค่ากลับเป็น 1 แต่หากไม่สำเร็จจะส่งค่ากลับเป็น 0

3.2.5 ฟังก์ชันการยกเลิกเหตุการณ์ ใช้การเรียกฟังก์ชันหมายเลข 0006h มีรายละเอียดดังนี้

```
char IPXCancelevent(ECB far *ecbr);
```

ค่าที่ส่งกลับมานั้นคือค่า AL สามารถดูความหมายได้จากตารางในข้อ 2.2.7.6 ในกรณีที่การยกเลิกสำเร็จ ค่า AL จะเป็น 00h การยกเลิกอาจจะกระทำเมื่อต้องการยกเลิกการส่งข้อมูลหรือการรอรับข้อมูล

3.2.6 ฟังก์ชันการขอค่าอินเตอร์เน็ตเวอร์กแอดเดรส จะใช้ฟังก์ชันหมายเลข 0009h ซึ่งโครงสร้างของอินเตอร์เน็ตเวอร์กแอดเดรสเป็นดังนี้

```
struct NetAddr
{
      char network[4];
      char node[6];
};
```

และมีรูปแบบของฟังก์ชันดังนี้

```
struct NetAddr IPXGetinternetworkaddress(void);
```

ค่าที่ถูกส่งกลับมาจะเป็นค่าของอินเตอร์เน็ตเวอร์กแอดเดรส ซึ่งประกอบด้วยโหนดแอดเดรส และเน็ตเวิร์กแอดเดรส

3.2.7 ฟังก์ชันปิดซอกเก็ต จะใช้เมื่อเลิกการใช้งานซอกเก็ตนั้นๆ แล้ว โดยใช้การทำงานผ่านฟังก์ชันหมายเลข 0001h อินเตอร์รัพท์หมายเลข 7Ah มีรายละเอียดดังนี้

```
void IPXClosesocket(unsigned int socket)
```

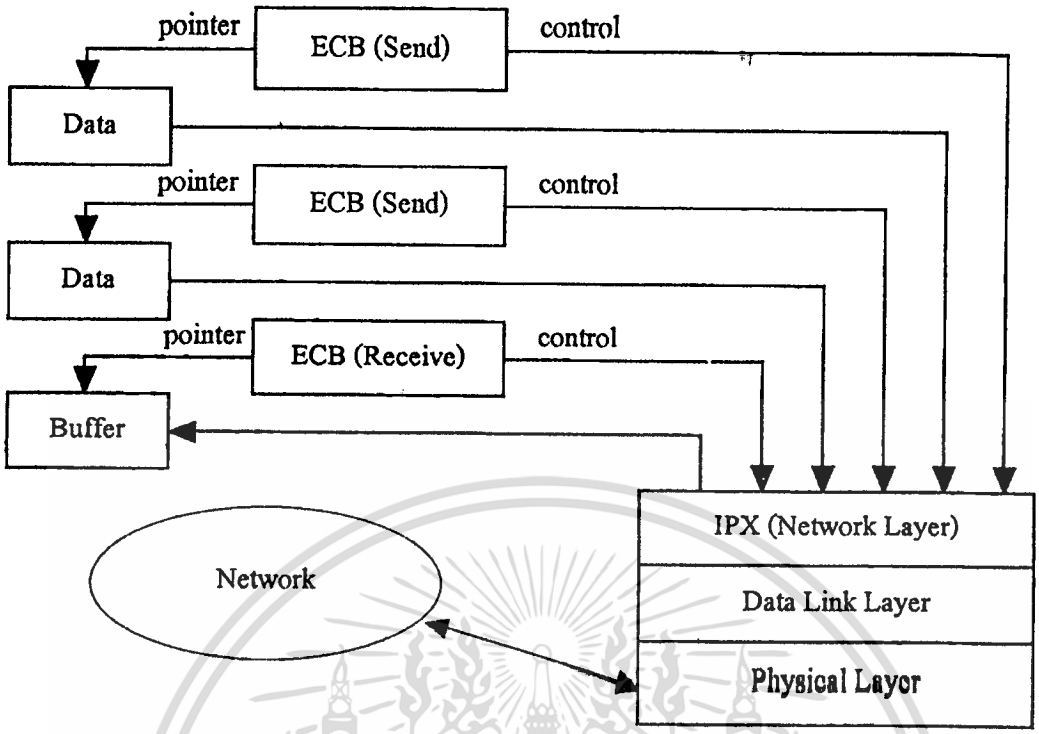
ค่า socket ที่ส่งให้กับฟังก์ชัน คือซอกเก็ตที่ต้องการปิด

3.2.8 ฟังก์ชันการรอช่วงหยุดการทำงานของ IPX ไดรเวอร์ การเรียกฟังก์ชันนี้เป็นการบอกกับ IPX ไดรเวอร์ เพื่อให้ IPX ไดร์เวอร์ได้มีจังหวะทำงาน ควรใช้เรียกก่อนที่จะมีการอ่านข้อมูลที่ได้รับเข้ามา การเรียกฟังก์ชันนี้ก่อนจะทำให้แน่ใจได้ว่า IPX ไดรเวอร์ได้ทำการรับข้อมูลจนครบถ้วนแล้ว มิเช่นนั้นอาจทำให้ข้อมูลที่ได้รับมาไม่ครบถ้วน หรือการรับข้อมูลผิดพลาดได้ การทำงานจะอาศัยฟังก์ชันหมายเลข 000Ah ของอินเตอร์รัพท์หมายเลข 7Ah ฟังก์ชันนี้เรียกได้โดยไม่มีการส่งค่ากลับ และไม่ต้องใส่พารามิเตอร์ใดๆ ดังนี้

```
void IPXRelinquish(void);
```

3.3 การรับส่งข้อมูลโดยโปรโตคอล IPX

การรับส่งข้อมูลโดยโปรโตคอล IPX นั้น จะถูกควบคุมโดย ECB (Event Control Block) ที่ผู้ใช้กำหนดขึ้น ใน ECB จะประกอบด้วยฟิลด์ต่างๆ ซึ่งใช้เป็นข่าวสารเพื่อควบคุมการส่งข้อมูล และข่าวสารเกี่ยวกับสถานะการทำงาน ในการส่งข้อมูล 1 แพกเก็ต จะต้องกำหนด ECB ไว้ 1 บล็อก และจะถูกใช้จนกว่าการรับส่งข้อมูลในแพกเก็ตนั้นจะเสร็จสิ้น



รูปที่ 3.2 - ตัวอย่างการใช้ ECB ควบคุมการรับส่งข้อมูล

จากรูปเป็นตัวอย่างการใช้ ECB ในการรับส่งข้อมูล ในการรับส่งข้อมูล สามารถที่จะกำหนด ECB ในการรับหรือส่งได้มากกว่า 1 บล็อกพร้อมๆ กัน การใช้ ECB หลายๆ บล็อกพร้อมๆ กันในการรับหรือส่ง ข้อมูลจะทำให้การส่งข้อมูลมีประสิทธิภาพมากยิ่งขึ้น ในด้านรับจะทำให้การรับข้อมูลเป็นไปได้อย่างรวดเร็วมากยิ่งขึ้น เนื่องจากสามารถที่จะรับข้อมูลได้ครั้งละหลายๆ แพกเก็ต นอกจากนั้นข้อมูลจะสูญหายน้อยลง เพราะฝั่งรับจะสามารถรับข้อมูลได้ทัน ส่วนในด้านฝั่งส่งนั้น การใช้ ECB หลายบล็อกพร้อมกัน จะทำให้การส่งข้อมูลเป็นไปได้รวดเร็วมากยิ่งขึ้น เนื่องจากไม่ต้องรอให้ส่งเสร็จทีละ packet แต่ข้อเสียของการส่งข้อมูลทีละหลายๆ แพกเก็ตโดยการใช้ ECB มากกว่า 1 ก็คือ ข้อมูลที่มาถึงพร้อมๆ กัน จะไม่สามารถเรียงลำดับก่อนหลังได้ จึงจำเป็นต้องมีการควบคุมโดยการใช้เลขลำดับข้อมูล ซึ่งจะกล่าวถึงต่อไป

3.3.1 ขั้นตอนการส่งข้อมูล

การส่งข้อมูล โปรโตคอล IPX จะทำการส่งข้อมูลผ่านลงไปยังชั้นเดต้าลิงก์ เพื่อให้ชั้นเดต้าลิงก์ทำการส่งข้อมูลไปยังปลายทาง การตอบรับจะเกิดในชั้นเดต้าลิงก์เอง ซึ่งหากการส่งข้อมูลล้มเหลว เช่น ไม่มีการตอบรับ หรือเกิดปัญหาทางด้านสายส่ง ชั้นเดต้าลิงก์จะบอกกลับขึ้นมายังโปรโตคอล IPX รับทราบ การตั้งค่าให้กับ ECB เพื่อทำการส่งข้อมูล มีดังนี้

- ให้ค่ากับ ECB ในส่วนของ Socket เพื่อระบุซอกเก็ตที่ใช้ในการส่งข้อมูล

-ให้ค่ากับ ECB ในส่วนของ Fragment Descriptor และ Fragment Count เพื่อระบุตำแหน่งและขนาดของข้อมูลที่ต้องการส่ง โดยจะต้องกำหนดให้ส่วนแรกของข้อมูลเป็นเฮดเดอร์ของแพกเก็ตที่ต้องการส่งเสมอ

- กำหนดค่าอินเตอร์เน็ตเวิร์กแอดเดรสและซอกเก็ตของเครื่องรับ ในส่วนเฮดเดอร์ของข้อมูล

เมื่อได้ทำการกำหนดค่าทั้งหมดเสร็จแล้ว จึงส่งให้กับฟังก์ชันส่งข้อมูลของโปรโตคอล IPX โปรโตคอล IPX จะทำการรับทราบและพยายามส่ง โดยจะไม่มีการรอให้ส่งเสร็จก่อน โปรโตคอล IPX จะพยายามทำการส่งอยู่ใน background โดยการตรวจสอบสถานะของการส่ง ว่าอยู่ในขั้นตอนใด สามารถตรวจสอบได้จากค่า inuseflag หากค่า inuseflag เป็น 0 หมายถึงโปรโตคอล IPX ได้พยายามทำการส่งเสร็จสิ้นแล้ว ซึ่งอาจจะสำเร็จหรือล้มเหลวก็ได้ โดยค่า completion code ใน ECB จะเป็นตัวบ่งชี้ถึงผลลัพธ์

3.3.2 ขั้นตอนการรอรับข้อมูล

การรอรับข้อมูล (Listen for packet) คือการบอกกับโปรโตคอล IPX ว่า แอพพลิเคชันกำลังรอข้อมูลอยู่ในขณะนั้น หากมีข้อมูลเข้ามาโปรโตคอล IPX ก็จะทำการรับข้อมูลไว้แล้วนำไปเก็บไว้ในบัฟเฟอร์ที่กำหนดไว้ทันที หากไม่มีการรอรับข้อมูล ข้อมูลที่โปรโตคอล IPX ได้นั้นจะไม่ถูกส่งต่อขึ้นมาสู่ชั้นที่อยู่เหนือขึ้นไป และชั้นที่อยู่เหนือขึ้นไปก็ไม่สามารถรู้ได้ว่าโปรโตคอล IPX ได้รับข้อมูลอะไรเข้ามาบ้าง ซึ่งจะทำให้ข้อมูลสูญหาย ในการรอรับข้อมูล ก่อนอื่นจำเป็นที่จะต้องมีการเปิดซอกเก็ตขึ้นมาก่อน และเมื่อเปิดขึ้นมาแล้ว สามารถรอรับข้อมูลเป็นจำนวนกี่ครั้งก็ได้ ไม่จำเป็นต้องมีการเปิดซอกเก็ตขึ้นมาใหม่อีก วิธีการตั้งค่าให้กับ ECB เพื่อใช้ในการรอรับข้อมูล มีดังต่อไปนี้

-ให้ค่ากับ ECB ในส่วนของ Socket เพื่อระบุซอกเก็ตในการรอรับข้อมูล

-ให้ค่ากับ ECB ในส่วนของ Fragment Descriptor และ Fragment Count เพื่อระบุตำแหน่งและขนาดของ buffer ที่จะใช้เก็บข้อมูลที่ได้รับมา โดยหากขนาดของ Packet นั้นใหญ่เกินที่จะใส่ได้ใน buffer โปรโตคอล IPX จะตัดส่วนที่เหลือทิ้งไป

- ตั้งค่าให้กับ ESRAddress ซึ่งควรตั้งเป็น NULL เมื่อไม่มีหรือไม่ต้องการใช้ ESR

เมื่อตั้งค่าให้กับ ECB จนเสร็จสิ้น จึงส่งให้กับฟังก์ชันการรอข้อมูลของโปรโตคอล IPX เมื่อมีแพกเก็ตเข้ามา โปรโตคอล IPX จะเก็บข้อมูลไว้ใน buffer ที่ตำแหน่งที่ได้กำหนดไว้ใน Fragment Descriptor โดยรายละเอียดทุกอย่างของแพกเก็ตจะอยู่ในส่วนของเฮดเดอร์ของแพกเก็ต (หัวข้อ 2.2.8) จากเฮดเดอร์ ซึ่งทำให้สามารถระบุที่มาของแพกเก็ตได้ และการตรวจสอบว่ามีแพกเก็ตเข้ามาใน buffer ที่กำหนดไว้หรือยัง สามารถตรวจสอบได้จากค่า completion code ใน ECB ซึ่งหากเป็น 0 เมื่อใด หมายถึงการรับข้อมูลถูกต้องและเสร็จสิ้นแล้ว และจากค่า inuse flag ของ ECB จะเป็นค่าที่บอกว่า ECB ที่กำหนดไว้ให้กับโปรโตคอล IPX กำลังถูกใช้อยู่หรือไม่ หากค่า inuse flag เป็น 0 หมายถึงการทำงานในการรอรับข้อมูลเสร็จสิ้นแล้ว แต่ได้รับข้อมูลหรือไม่จะดูจาก completion code ดังที่กล่าวไปแล้ว ส่วนความยาวของข้อมูลนั้น สามารถดูได้ในเฮดเดอร์ของแพกเก็ตที่ได้รับมา

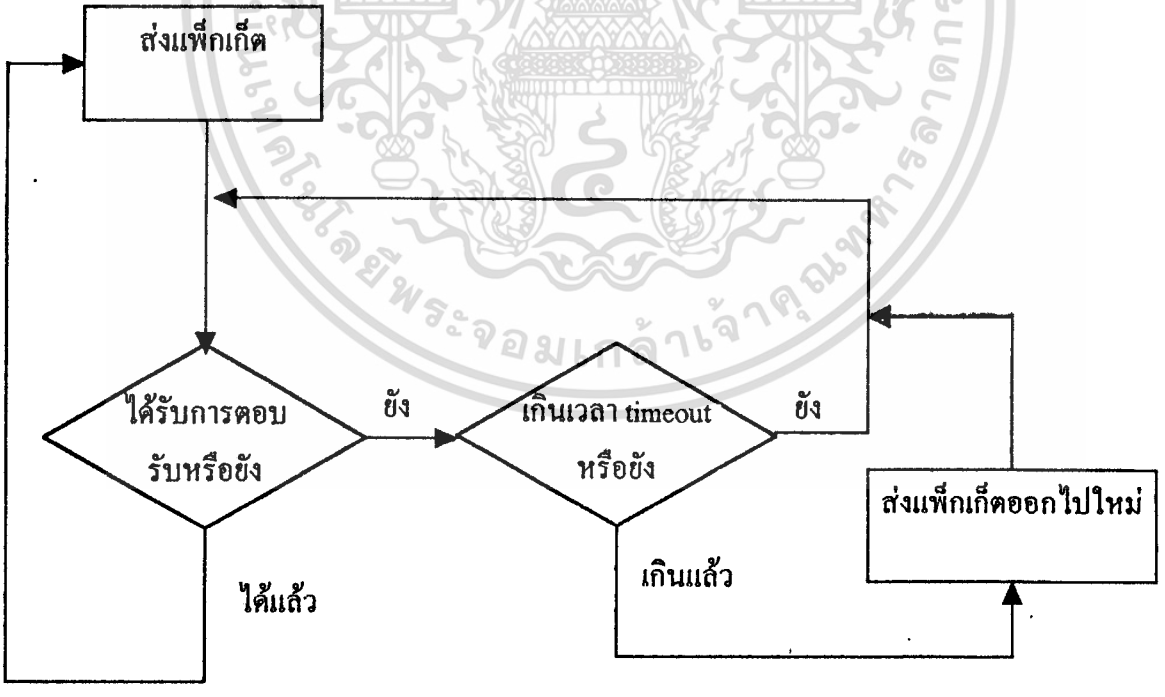
และต้องระวังว่า ค่าความยาวของข้อมูลที่ได้รับมานั้นอยู่ในรูปแบบ big-endian ซึ่งต้องมีการสลับไบต์สูงกับไบต์ต่ำก่อน จึงจะเป็นความยาวที่ถูกต้อง อนึ่ง ในกรณีที่ใช้ ECB มากกว่า 1 บล็อกในการรอข้อมูลในซอกเก็ตเดียวกัน หากมีข้อมูลเข้ามา จะไม่สามารถรู้ได้ว่า ข้อมูลนั้นจะไปอยู่ใน ECB บล็อกใด ดังนั้นเมื่อข้อมูลถูกส่งมามากกว่า 1 แพกเก็ต พร้อมๆ กัน จึงไม่สามารถจำแนกได้ว่า แพกเก็ตไหนที่มาก่อน นอกจากจะมีการกำหนดเลขลำดับข้อมูล

3.4 ระบบรับส่งข้อมูล

ระบบรับส่งข้อมูลที่ต้องการใช้ในโครงงานนี้ ต้องเป็นระบบที่สามารถรับส่งข้อมูลได้รวดเร็ว และสามารถรับส่งข้อมูลได้พร้อมๆ กัน เนื่องจากเป็นการติดต่อแบบสองทาง และต้องมีลำดับของข้อมูลที่ถูกต้อง การส่งข้อมูลชุดใหม่เมื่อข้อมูลสูญหายยังถือว่าสามารถใช้ได้

3.4.1 รูปแบบการรับส่งข้อมูล

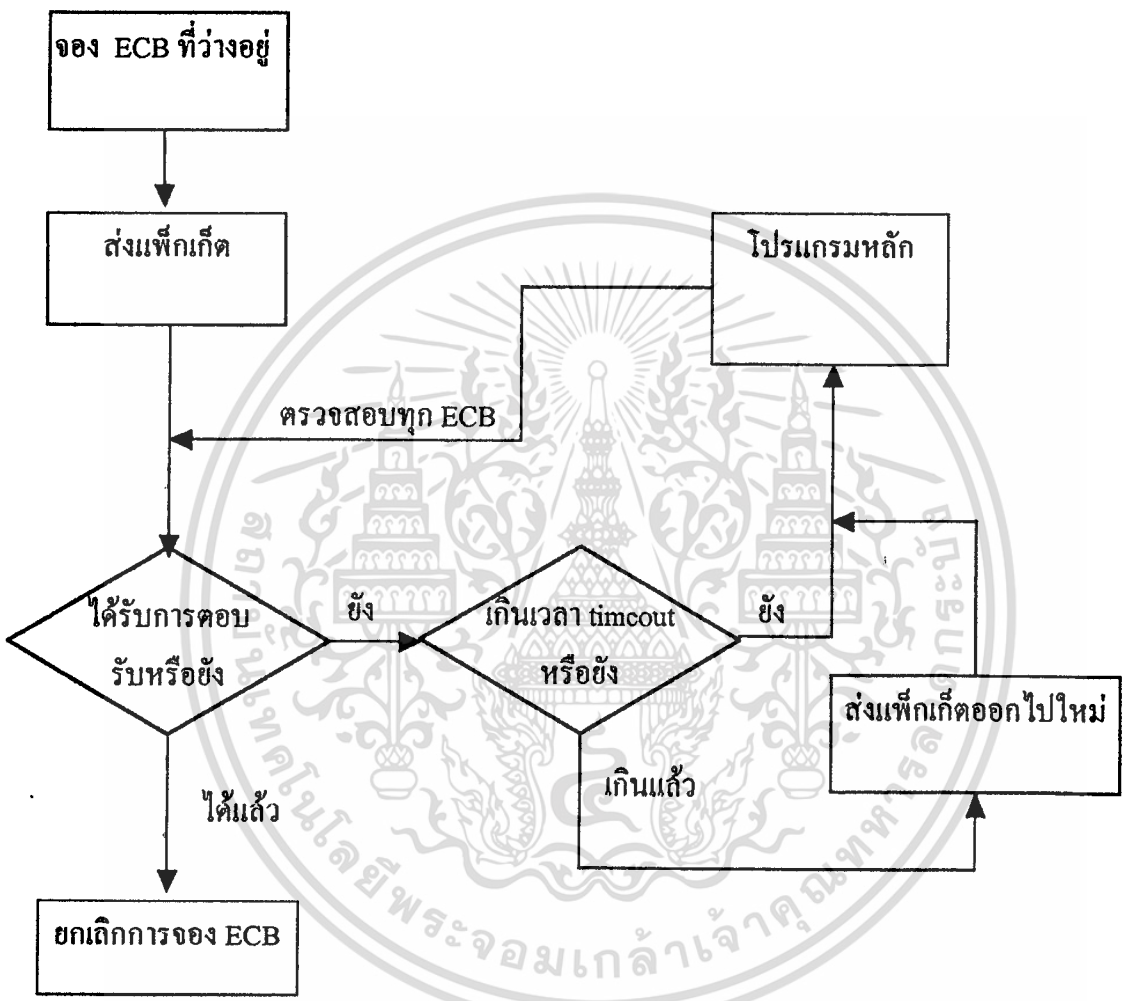
เนื่องจากการส่งข้อมูลโดยโปรโตคอล IPX ไม่มีเลขลำดับข้อมูล ซึ่งจำเป็นต้องใช้หากต้องการข้อมูลที่เรียงลำดับถูกต้อง ซึ่งความจริงการรับส่งข้อมูลแบบ Stop and wait ด้วยการใช้ ECB เพียง 1 บล็อก และใช้การตอบรับนั้น นับว่าใช้ได้และสามารถทำให้ข้อมูลเรียงลำดับถูกต้อง แต่จะทำให้การรับส่งข้อมูลเป็นไปได้ช้า เนื่องจากฝ่ายส่งจำเป็นต้องหยุดรอการตอบรับจากฝั่งตรงข้ามทุกครั้ง ก่อนที่จะมีการส่งแพกเก็ตต่อไป



รูปที่ 3.3 - อัลกอริทึมการส่งข้อมูลแบบ Stop-and-Wait

สำหรับทางด้านรับนั้น การทำงานจะมีเพียงการเตรียมรับ และการตรวจสอบการซ้ำกันของแพ็กเก็ตเท่านั้น ฝ่ายรับจะต้องทำการตอบรับทุกๆ แพ็กเก็ต ไม่ว่าจะซ้ำกันหรือไม่ก็ตาม

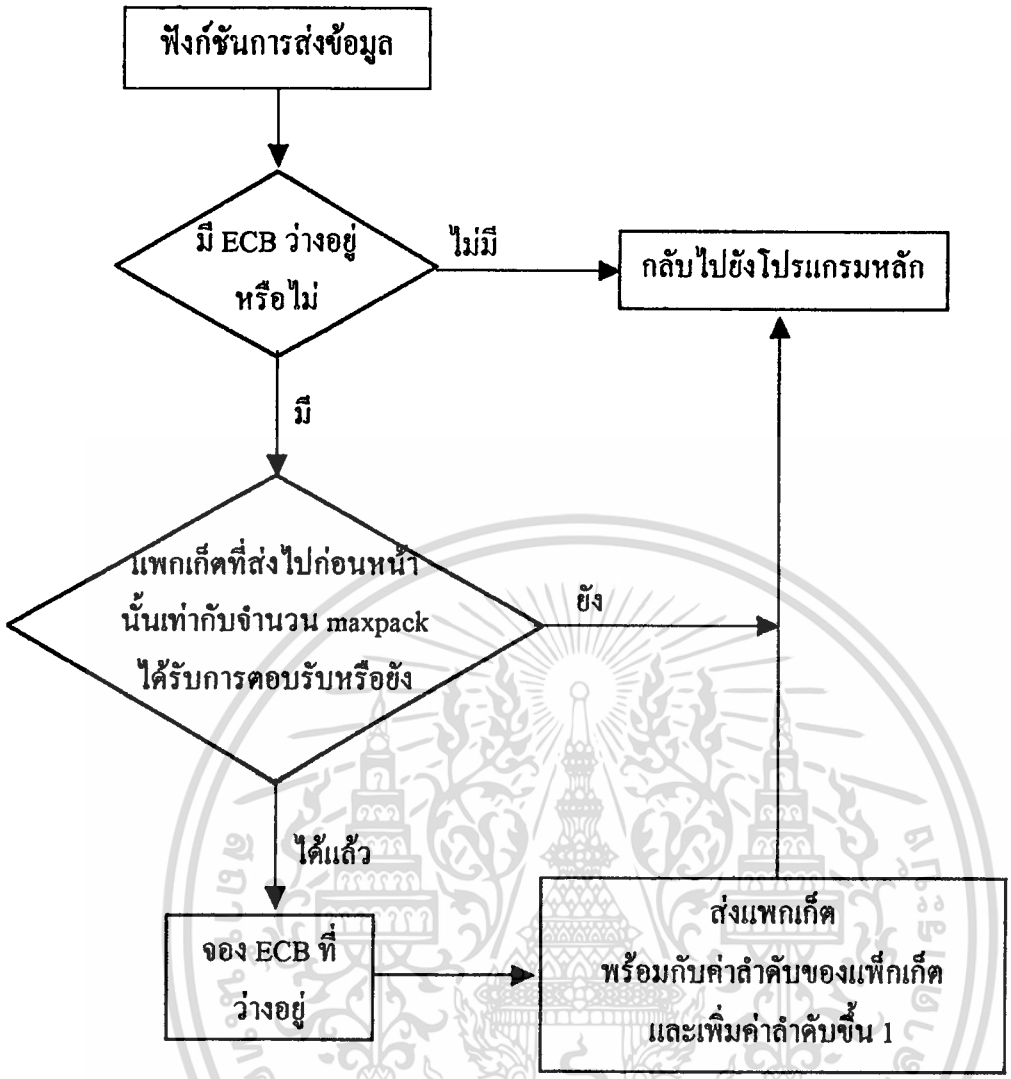
จากข้อเสียดังกล่าว จะนำเอารูปแบบ Stop-and-Wait มาทำการพัฒนา โดยการใช้รูปแบบที่คล้ายกัน แต่จะสามารถทำให้ส่งข้อมูลครั้งละหลายๆ แพ็กเก็ตได้พร้อมๆ กัน โดยการใช้ ECB ครั้งละหลายๆ บล็อก



รูป 3.4 - การใช้ ECB มากกว่า 1 บล็อกในการส่งข้อมูล

การทำงานรูปแบบนี้ ECB จะถูกตรวจเช็คโดยการเรียกจากโปรแกรมหลักอยู่เสมอ ซึ่งหาก ECB บล็อกใดได้รับการตอบรับแล้ว ก็จะยกเลิกการจอง ECB นั้นๆ ไป หากไม่ได้รับการตอบรับภายในเวลาที่กำหนด ก็จะทำการส่งแพ็กเก็ตออกไปใหม่ ซึ่งการส่งข้อมูลแบบนี้ฝ่ายรับก็จำเป็นที่จะต้องเตรียม ECB และบัฟเฟอร์ในการเก็บข้อมูลมากกว่า 1 บล็อกเช่นเดียวกัน เพื่อป้องกันการรับข้อมูลจากฝั่งส่งไม่ทัน เนื่องจากฝั่งส่งมีโอกาสส่งข้อมูลได้เป็นจำนวนมากกว่าหนึ่งแพกเก็ตในเวลาไล่เลี่ยกัน

อย่างไรก็ดี การส่งแบบนี้ยังมีข้อเสียอยู่ที่ไม่มีการควบคุมการไหลของข้อมูล ไม่มีเลขลำดับข้อมูล ซึ่งทำให้ฝั่งรับไม่สามารถรับข้อมูลได้อย่างมีลำดับ และอาจเกิดเหตุการณ์รับข้อมูลไม่ทัน ซึ่งอาจทำให้ข้อมูลสูญหายได้ ดังนั้นจึงต้องมีการส่งเลขลำดับข้อมูลไปพร้อมๆ กับข้อมูลด้วย

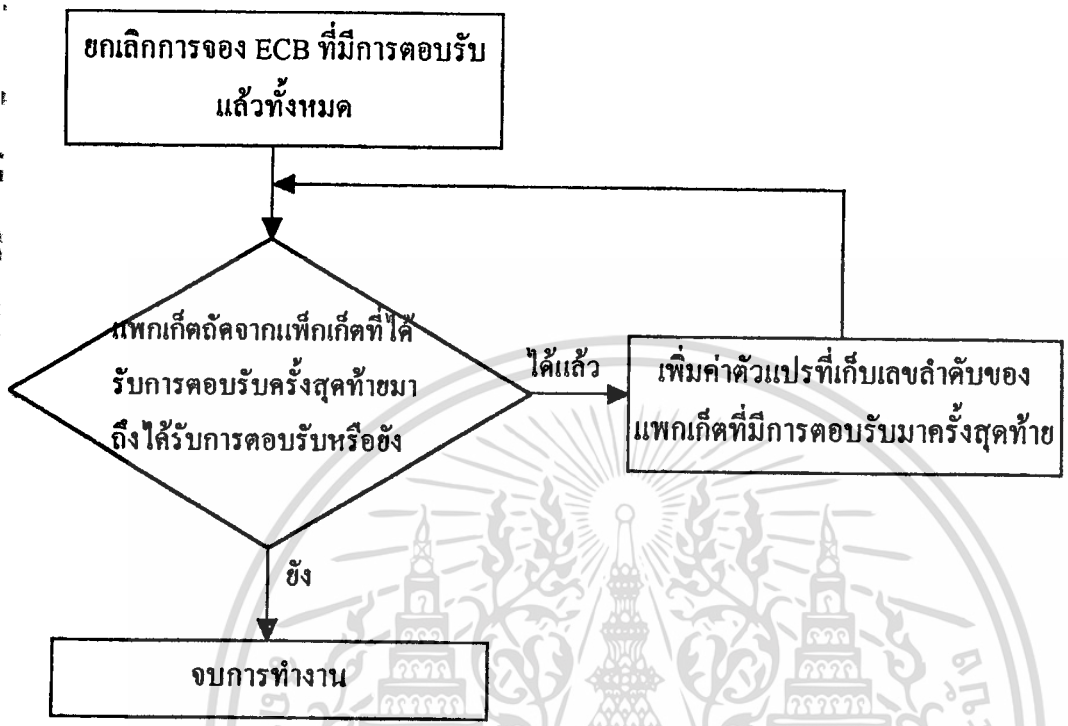


รูป 3.5 - อัลกอริทึมการส่งข้อมูลโดยการใช้ค่าลำดับ

จากขั้นตอนการทำงานโดยการใช้เลขลำดับข้อมูล จะเห็นว่า มีการกำหนดค่า maxpack เพื่อเป็นจำนวนแพกเก็ตที่มากที่สุดในการส่งแพกเก็ตพร้อมกันในขณะใดขณะหนึ่ง โดยการตรวจสอบกับการตอบรับ ซึ่งการตอบรับจะทำให้เราทราบได้ว่า ฝั่งตรงข้ามได้รับข้อมูลถึงเลขลำดับใดแล้ว หากฝั่งรับได้รับแพกเก็ตที่ขาดหายไปบางแพกเก็ต ทางฝั่งรับจะต้องเก็บข้อมูลที่มาหลังจากแพกเก็ตที่ขาดหายไปไว้ทั้งหมด ดังนั้น หากเลขลำดับใดลำดับหนึ่งไม่ถูกตอบรับโดยฝั่งรับ หรือไปไม่ถึงฝั่งรับ ทางฝั่งส่งจะต้องหยุดส่งข้อมูลหลังจากนั้นเป็นจำนวนใดจำนวนหนึ่ง (ในที่นี้กำหนดให้เป็น maxpack) เพื่อไม่ให้ทางฝั่งรับต้องรับข้อมูลมากกว่าที่จะรับได้ โดย maxpack ควรจะมีค่าประมาณเท่ากับจำนวน buffer ทั้งหมดของฝั่งส่ง และหากแพกเก็ตใดไม่มีการตอบรับมานานเกินกว่าเวลา timeout ก็จะส่งแพกเก็ตไปใหม่

การทำงานในส่วนการส่งข้อมูลแบบมีลำดับ จะต้องมีส่วนสนับสนุน ก็คือส่วนที่คอยยกเลิกการจอง ECB สำหรับแพกเก็ตที่ได้รับการตอบรับเรียบร้อยแล้ว เพื่อให้ส่วนการส่งข้อมูล สามารถใช้ ECB บล็อกนั้นใน

การส่งข้อมูลในแพกเก็ตอื่นต่อไปได้ รวมถึงการตรวจสอบค่าหมายเลขตอบรับ เพื่อนำไปใช้ในการทำงานในการส่งข้อมูล



รูป 3.6 - การทำงานในส่วนยกเลิกการจอง ECB

ในส่วนของการยกเลิกการจอง จะมีส่วนที่ทำการเพิ่มค่าของตัวแปรที่เก็บเลขลำดับของ แพกเก็ตที่มีการตอบรับมาครั้งสุดท้าย การเพิ่มค่าจะไม่เกิดขึ้น หากการตอบรับแพกเก็ตมีการขาดหายไป

ในด้านการรับข้อมูล จำเป็นที่จะต้องตระเตรียม buffer ไว้ให้เพียงพอในการรับข้อมูล และทำการตอบรับเพื่อให้ฝั่งส่งรับทราบในทุกแพกเก็ต การทำงานในส่วนทางด้านรับจะมีการตรวจสอบลำดับของข้อมูลเพื่อตรวจหาการซ้ำกันของแพกเก็ต และการขาดหายไปของแพกเก็ต เพื่อที่จะสามารถเรียงลำดับแพกเก็ตได้อย่างถูกต้อง การรับแพกเก็ตมีอัลกอริทึมการทำงานดังนี้

1. ให้ n เป็นค่าที่เก็บหมายเลขแพ็กเก็ตสุดท้ายที่ถูกต้องไว้ โดยมีค่าเริ่มต้นที่ 0
2. ตรวจสอบหา ECB ที่มี packet ใน buffer ที่มีค่าเลขลำดับต่ำสุด โดย ECB บล็อกใดถูกเลือกแล้วจะไม่มีการเลือกซ้ำอีก ถ้าถูกเลือกจนหมด ให้ไปที่ข้อ 6.
3. ถ้าเลขลำดับที่ได้ ต่ำกว่าหรือเท่ากับ n หมายถึงเกิดการซ้ำกันของแพกเก็ต ให้ทำลายข้อมูลใน buffer และให้ ECB นั้นรอรับแพกเก็ตใหม่ และกลับไปทำงานที่ข้อ 2 ใหม่
4. ถ้าเลขลำดับที่ได้ เป็นค่าที่มีลำดับต่อจาก n ให้เพิ่มค่า n อีก 1 และให้เก็บข้อมูลใน buffer นั้นไว้ และกลับไปทำงานในข้อ 2.

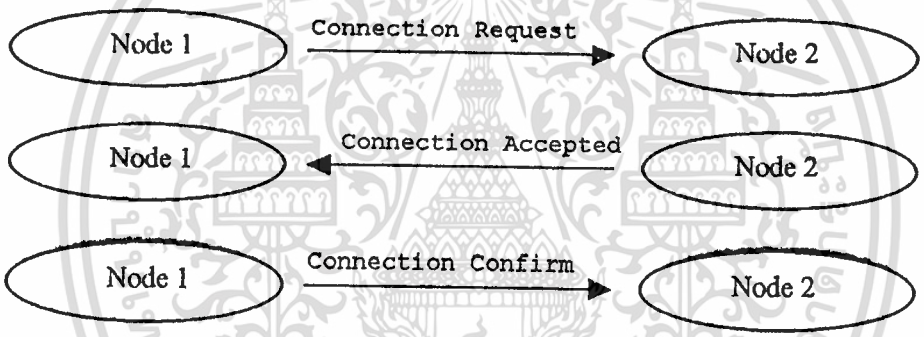
5. ถ้าเลขลำดับที่ได้ มากกว่าเลขลำดับที่มีค่าต่อจาก h ให้เก็บข้อมูลนั้นไว้เช่นกัน แต่ไม่ต้องมีการเพิ่มค่า n และกลับไปทำงานที่ข้อ 2.

6. จบการทำงาน

อัลกอริทึมในการรอรับแพกเก็ต จะตรวจสอบการซ้ำกันของแพ็กเก็ต และการขาดหายของแพกเก็ต แพกเก็ตใดที่ซ้ำกันหรือได้รับแล้วจะถูกโยนทิ้งไป แพกเก็ตใดที่มาหลังแพกเก็ตที่ขาดหายจะถูกเก็บไว้ เพื่อรอจนกว่าแพกเก็ตที่ขาดหายไปจะมาใหม่ หากแพกเก็ตไม่มีการขาดหายเลย ค่า n จะถูกเพิ่มจนเท่ากับเลขลำดับของแพกเก็ตสุดท้ายที่ได้รับมาอย่างมีลำดับถูกต้อง

3.4.2 การส่งสัญญาณ

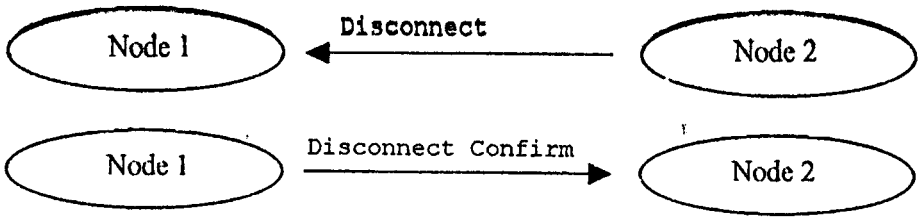
การส่งสัญญาณ จะใช้เพื่อการส่งข้อมูลข่าวสารบางอย่างที่ไม่ใช่ตัวข้อมูล เช่น สัญญาณขอติดต่อ สัญญาณเลิกการติดต่อ เป็นต้น โดยการเริ่มการติดต่อ ควรจะมีขั้นตอนการส่งสัญญาณ 3 ขั้นตอน ดังนี้



รูปที่ 3.7 - ขั้นตอนการส่งสัญญาณบอกเริ่มการติดต่อ

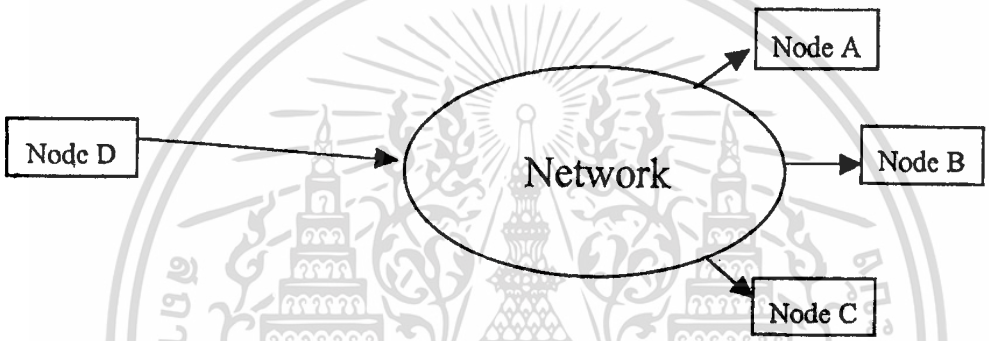
การติดต่อ จะเริ่มจากการที่ Node 1 ต้องการติดต่อกับ Node 2 จึงส่งสัญญาณ Connection Request ให้กับ Node 2 เมื่อ Node 2 รับสัญญาณได้ว่า Node 1 ต้องการติดต่อด้วย ก็จะพิจารณาว่า ต้องการยอมรับการติดต่อหรือไม่ ถ้าต้องการก็จะส่งสัญญาณ Connection Accepted กลับไปยัง Node 1 เมื่อ Node 1 ได้รับสัญญาณดังกล่าวก็จะส่งสัญญาณ Connection Confirm กลับไปยังโหนด 2 เพื่อยืนยันการติดต่อ อนึ่ง ในกรณีที่ Node 2 ไม่ต้องการติดต่อกับ Node 1 อาจจะมีการส่งสัญญาณ Connection Refuse ไปยัง Node 1 เพื่อบอกให้ Node 1 ทราบว่า ไม่ต้องการติดต่อด้วย แต่หากว่า Node 2 นั้น ไม่มีอยู่จริง สัญญาณที่ Node 1 ส่งออกไปจะไม่ได้รับการตอบรับใดๆ และ Node 1 ก็จะรอเป็นเวลาช่วงหนึ่ง และยกเลิกการรอไปในที่สุด

หลังจากการติดต่อเสร็จสิ้น ก็จะเป็นการส่งข้อมูลกันระหว่างโหนดทั้งสอง ซึ่งอาจจะมีการส่งสัญญาณข้อมูลข่าวสารบางอย่างก่อน และเมื่อเลิกการติดต่อ ก็จะมีวิธีการบอกเลิกการติดต่อดังนี้



รูป 3.8 - การส่งสัญญาณบอกยกเลิกการติดต่อ

นอกจากการส่งสัญญาณบอกเริ่มและยกเลิกการติดต่อ อาจจะมีการส่งสัญญาณแบบอื่นๆ ด้วย ในโครงงานนี้จะมีการสร้างการส่งสัญญาณเพื่อค้นหาผู้ใช้ใน Network โดยใช้การบรอดคาสติง (Broadcasting)

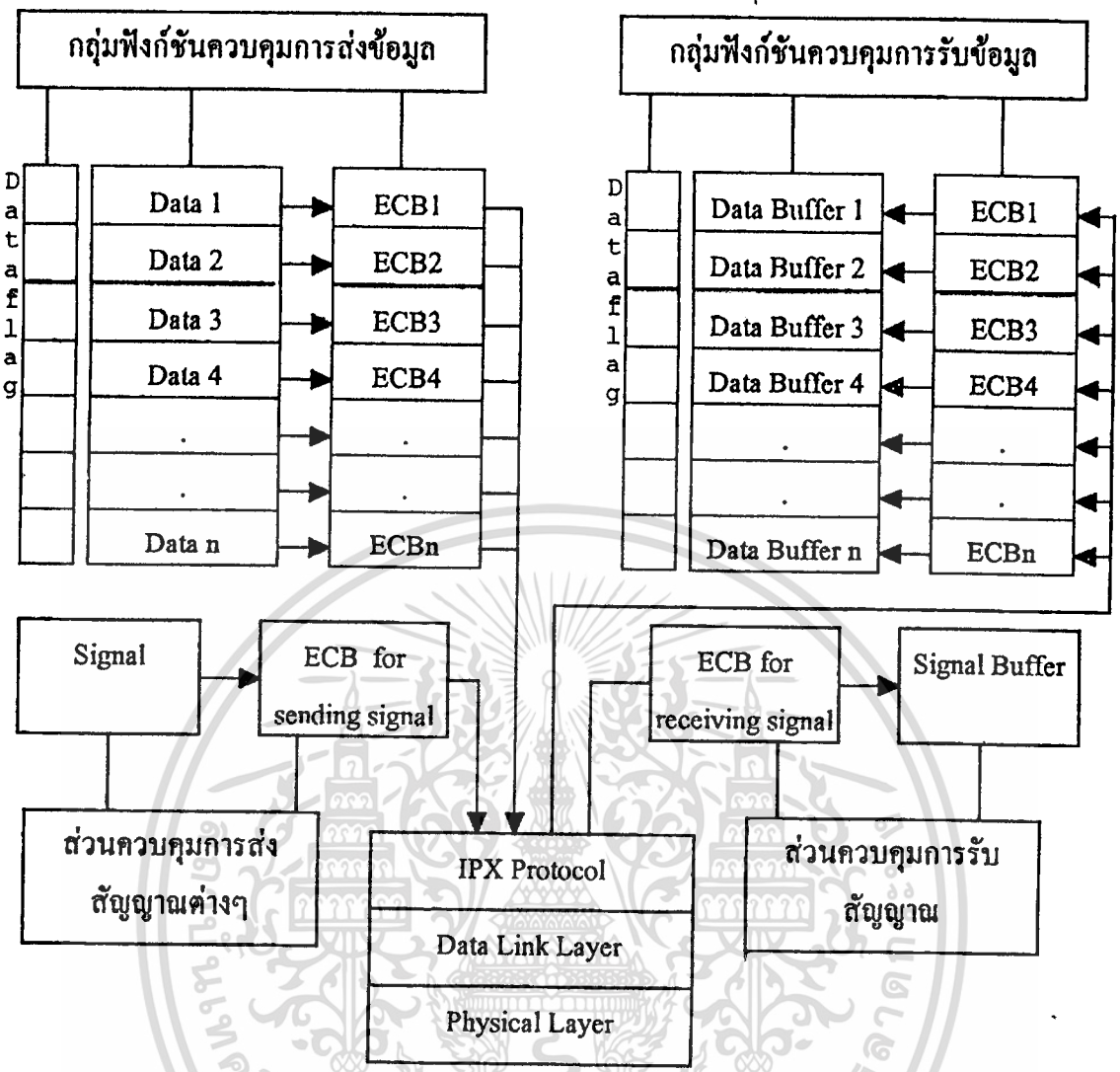


รูปที่ 3.9 - การบรอดแคสติง

จากรูป จะเป็นการส่งแพกเก็ตแบบบรอดคาสติง คือ เมื่อ Node D ทำการบรอดคาสติงออกมา ทุกๆ โหนดที่อยู่บนเครือข่ายจะได้รับแพกเก็ตที่ออกมาจาก Node D เหมือนกันหมด การนำมาใช้ในการส่งสัญญาณเพื่อค้นหาผู้ใช้ สมมติว่า โหนด Dโหนด ต้องการค้นหา โหนด D จะบรอดคาสติงแพ็กเก็ตที่มีชื่อผู้ใช้ที่ต้องการค้นหาออกมา เมื่อทุกโหนดได้รับข้อมูล แต่ละโหนดจะตรวจสอบว่า ชื่อดังกล่าวตรงกับชื่อโหนดของตนหรือไม่ ถ้าหากตรงกัน ก็จะส่งสัญญาณกลับไปยังโหนด D เพื่อตอบรับว่า โหนดของตนมีชื่อตรงกัน โหนด D ก็จะรับทราบได้ว่า โหนดที่ต้องการค้นหามีอยู่บนเครือข่าย และทราบอินเตอร์เน็ตเวิร์กแอดเดรสจากเฮดเดอร์ของแพกเก็ตที่ตอบรับมา

3.4.3 รูปแบบของระบบรับส่งข้อมูล

จากหัวข้อ 3.4.1 และ 3.4.2 จะนำการทำงานในส่วนต่างๆ มาแบ่งและออกแบบเป็นระบบ โดยมีฟังก์ชันควบคุมการทำงานในแต่ละส่วน ในส่วนการรับข้อมูลและการส่งข้อมูลจะเป็นอิสระต่อกัน ในส่วนควบคุมการส่งข้อมูลจะทำการควบคุม ECB ซึ่งใช้ในการส่ง ทำการใส่ข้อมูลลงไป และจัดการกับค่า Dataflag ซึ่งเป็นค่าที่บอกว่า Data ในส่วนนั้นๆ ได้ถูกใช้งานอยู่หรือไม่ ส่วนทางด้านรับก็จะจัดการกับ ECB การจัดเก็บข้อมูลและ Dataflag เช่นเดียวกัน



รูปที่ 3.10 - ระบบการส่งข้อมูล

การที่ใช้ Signal buffer เพียงช่องเดียว เป็นการประหยัดทรัพยากรของระบบ เพราะว่าการส่งสัญญาณนั้นมักจะไม่มีการส่งแบบต่อเนื่อง และมีการใช้งานน้อย จึงไม่มีความจำเป็นที่จะเผื่อบัฟเฟอร์ไว้เป็นจำนวนมาก โดยการรับสัญญาณจะใช้ซอกเก็ตแยกต่างหากจากซอกเก็ตที่ใช้รับข้อมูล ระบบนี้มีการจองหน่วยความจำเพื่อเป็น buffer ในการรับส่งไว้ตายตัว ซึ่งจะต้องมี buffer ในการรับมากกว่าการส่งประมาณ 2 เท่า โดยคิดจากกรณีที่เลวร้ายที่สุด (worse case) ในการสูญหายของข้อมูล

3.5 คลาสเพื่อการรับส่งข้อมูล

จากหัวข้อ 3.2 จะเห็นว่า ฟังก์ชันพื้นฐานนั้นสามารถใช้ส่งข้อมูลได้ก็จริง แต่ก็ยังมีความลำบากในการใช้งานอยู่มาก เช่น การจะส่งข้อมูลแต่ละครั้งจะต้องมีการกำหนดค่าให้กับในหลายๆ ส่วนของ ECB และยังต้องกำหนดค่าในเซตเตอร์ต่างๆ อีก ดังนั้นจึงสร้างคลาสขึ้นมาเพื่อรองรับการทำงานในการรับส่งข้อมูล ชื่อว่าคลาส IPX โดยใช้ระบบการส่งข้อมูลในหัวข้อ 3.4.3 เป็นต้นแบบ

3.5.1 การออกแบบฟังก์ชันในการใช้งาน

ในการสร้างคลาส จะออกแบบในส่วนของฟังก์ชันที่ต้องการใช้งาน ว่ามีความต้องการอะไรในการสร้างคลาสนี้ขึ้นมา คลาสนี้มีหน้าที่ทำอะไรบ้าง แล้วจึงค่อยออกแบบการทำงานของฟังก์ชัน และค่า property ต่างๆ ทีหลัง การออกแบบฟังก์ชันที่ต้องการใช้ ต้องออกแบบให้ครบถ้วน เพื่อให้สามารถใช้งานได้อย่างมีประสิทธิภาพ ซึ่งได้ออกแบบดังนี้

- ฟังก์ชันเพื่อติดต่อ รอรับการติดต่อ และยอมรับการติดต่อ
- ฟังก์ชันเพื่อการส่งข้อมูล โดยอ้างถึงหมายเลขการติดต่อ
- ฟังก์ชันเพื่อการรับข้อมูลจากบัฟเฟอร์ โดยอ้างถึงหมายเลขการติดต่อ
- ฟังก์ชันยกเลิกการติดต่อ
- ฟังก์ชันในการตรวจสอบความเป็นไปได้ในการส่งข้อมูลขณะนั้น
- ฟังก์ชันตรวจสอบข้อมูลในบัฟเฟอร์
- ฟังก์ชันเพื่อค้นหาผู้ใช้บนเครือข่าย
- ฟังก์ชันตรวจสอบสถานะการติดต่อขณะนั้น

นอกจากฟังก์ชันที่ได้กล่าวมา ความจริงยังมีอีกหนึ่งฟังก์ชันที่ชื่อว่า alwayscall ซึ่งต้องถูกเรียกอยู่ตลอดเวลา แต่มิได้มีประโยชน์ในการใช้งานโดยตรง แต่ใช้เพื่อควบคุมการรับส่งข้อมูล การส่งใหม่ การเก็บข้อมูลในบัฟเฟอร์ไว้หรือเคลียร์บัฟเฟอร์ และการตรวจสอบการ timeout เป็นต้น ให้เป็นไปอย่างอัตโนมัติ โดยในการใช้งานจะต้องเรียกฟังก์ชันนี้ทุกๆ รอบการทำงาน

3.5.2 ตัวแปรในส่วน private

การทำงานของคลาส จะซ่อนตัวแปรในส่วนของ private ไม่ให้เข้าถึงได้จากภายนอก ซึ่งในส่วนนี้เราจะเก็บตัวแปรและฟังก์ชันที่ไม่มีความจำเป็นต้องใช้จากภายนอก รวมไปถึงตัวแปรที่ไม่ควรถูกเปลี่ยนแปลงค่าหรือฟังก์ชันที่ไม่ควรถูกเรียกโดยโปรแกรมภายนอก ซึ่งรายละเอียดของตัวแปรต่างๆ มีดังนี้

- ecbrecv, ecbsend, ecbrecvsig และ ecbsendsig เป็นตัวแปรโครงสร้าง ECB ใช้สำหรับเป็น ECB ในการรับส่งข้อมูลและสัญญาณ
- ecbrecvstat และ ecbsendstat ใช้สำหรับเก็บค่าสถานะของข้อมูลในบัฟเฟอร์ของ ecbrecv และ ecbsend
- ecbrecvbuffer, ecbsendbuffer เป็นบัฟเฟอร์ของ ecbrecv และ ecbsend
- sendtime ใช้เก็บเวลาที่ packet ถูกส่งออกไป เพื่อใช้ตรวจสอบการ timeout
- sendsignal เป็นฟังก์ชันที่ใช้สำหรับส่งสัญญาณไปยังเครื่องปลายทาง
- sendblock เป็นฟังก์ชันที่ใช้สำหรับส่งข้อมูลไปพร้อมๆ กับเลขลำดับ
- countecbs ใช้สำหรับนับ ECB ที่ใช้ในการส่งที่กำลังถูกใช้งานอยู่

นอกจากนี้ ยังมีฟังก์ชันและการทำงานอื่นๆ อีก ซึ่งจะไม่กล่าวในที่นี้ โดยรายละเอียดของคลาสและการทำงานสามารถดูได้ในภาคผนวกในส่วนของไฟล์ IPXCLAS4.H ซึ่งจะเก็บคลาส IPX ไว้ คลาสที่ได้สร้างขึ้นทำให้การรับส่งข้อมูลเป็นรูปแบบมากยิ่งขึ้น และจะนำไปใช้ประยุกต์รวมกับการสร้างอินเตอร์เฟสต่อไป

3.5.3 การทำงานของแต่ละฟังก์ชัน

ในส่วนนี้จะอธิบายการทำงานของแต่ละฟังก์ชันในคลาส IPX

3.5.3.1 ฟังก์ชันเพื่อเริ่มการติดต่อ (Connect) ใช้เพื่อเริ่มการติดต่อ โดยการเรียกใช้งานจะต้องส่งค่าพารามิเตอร์เป็นอินเตอร์เน็ตเวิร์กแอดเดรสของเครื่องปลายทางให้กับฟังก์ชัน ฟังก์ชันนี้ทำงานโดยการส่งสัญญาณเพื่อขอเริ่มการติดต่อไปยังเครื่องปลายทาง และเปลี่ยนค่าสถานะของการติดต่อเป็น Connecting

3.5.3.2 ฟังก์ชันยกเลิกการติดต่อ (Disconnect) ทำงานโดยการส่งสัญญาณเพื่อขอเลิกการติดต่อไปยังปลายทาง และเปลี่ยนค่าสถานะของการติดต่อเป็น Disconnecting

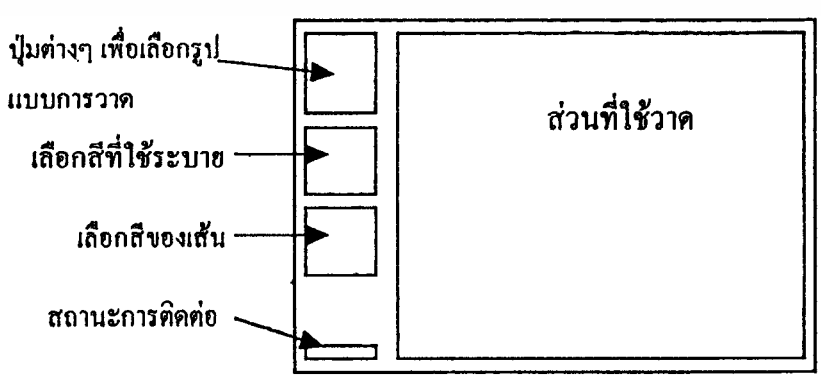
3.5.3.3 ฟังก์ชันส่งข้อมูล (Senddata) ฟังก์ชันนี้จะทำการส่งข้อมูลที่ตำแหน่งที่กำหนดไปยังเครื่องปลายทาง โดยในขั้นแรก ฟังก์ชันจะทำการค้นหา ECB ที่ใช้ในการส่งที่ว่างอยู่เสียก่อน เมื่อพบแล้ว ฟังก์ชันก็จะทำการจอง ECB นั้น ต่อจากนั้นฟังก์ชันก็จะทำการกำหนดค่าต่างๆ ใน ECB และเฮดเดอร์ของแพ็กเก็ตให้เหมาะสม ก่อนที่จะใช้ฟังก์ชัน IPXSendpacket เพื่อทำการส่งข้อมูลออกไป

3.5.3.4 ฟังก์ชันรับข้อมูล (Getdata) อันที่จริง ข้อมูลนั้นหากมีการส่งมา ทางฝั่งรับก็จะรับไว้ในบัฟเฟอร์อยู่แล้ว ฟังก์ชันนี้จะทำการดึงข้อมูลที่อยู่ในบัฟเฟอร์ออกมาใช้ โดยจะมีการตรวจสอบลำดับของข้อมูลด้วย ถ้าหากไม่ถูกต้องก็จะ ไม่ทำการดึงข้อมูลให้ จนกว่าจะได้รับข้อมูลที่ขาดหายไป

3.5.3.5 ฟังก์ชันตรวจสอบสถานะ (Getstat) ใช้เพื่อตรวจสอบสถานะของการติดต่อขณะนั้น

3.5.3.6 ฟังก์ชัน alwayscall เป็นฟังก์ชันที่จะถูกเรียกจากโปรแกรมหลักทุกรอบการทำงาน ใช้สำหรับการตรวจสอบการ timeout ของแพกเก็ต การตรวจสอบและประมวลผลสัญญาณที่เข้ามา การเคลียร์ ECB ที่เลิกใช้แล้ว เป็นต้น

3.6 รูปแบบอินเตอร์เฟส



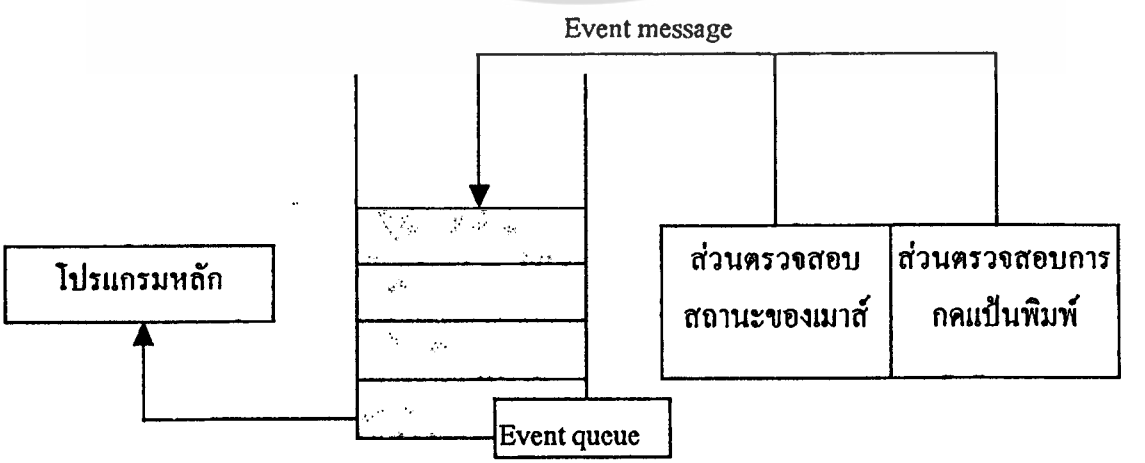
รูป 3.11 - รูปแบบกราฟิกอินเตอร์เฟส

รูปแบบของการอินเตอร์เฟส จะใช้เมาส์ และคีย์บอร์ดเป็นอุปกรณ์ในการรับอินพุต และใช้การแสดงผลแบบกราฟิก การวาดรูปจะใช้เมาส์เป็นหลัก สามารถเลือกสีที่ใช้ระบาย สีเส้น เลือกความหนาของเส้นได้ และสามารถพิมพ์ข้อความได้ การเลือกอุปกรณ์ในการวาด จะใช้ปุ่มรูปอุปกรณ์ต่างๆ เป็นไอคอนให้เลือกโดยการใช้เมาส์ รวมทั้งการเลือกสีต่างๆ ก็ใช้เมาส์เช่นกัน การวาดรูปจะกระทำได้เฉพาะในส่วนที่ได้กำหนดไว้ให้เป็นที่สำหรับวาดทางขวามือเท่านั้น ไม่สามารถวาดเกินออกมาได้ มุมล่างขวาจะแสดงสถานะการติดต่อว่าเป็นอย่างไร อยู่ในขั้นตอนไหน

3.6.1 การรับอินพุต

การรับอินพุตจากเมาส์ จะสร้างคลาสขึ้นมาอีกคลาสหนึ่งเพื่อรองรับการทำงาน ฟังก์ชันพื้นฐานที่ติดต่อกับเมาส์โดยผ่านอินเตอร์รัพท์หมายเลข 33h จะสามารถตรวจสอบการกดและปล่อยเมาส์ และตำแหน่งของเมาส์บนหน้าจอได้ (รายละเอียดของอินเตอร์รัพท์และฟังก์ชันเกี่ยวกับเมาส์อยู่ในภาคผนวก) รวมไปถึงการแสดงและซ่อนตัวชี้ ในการสร้างคลาสของเมาส์ขึ้น เพื่อให้การตรวจสอบการกดและปล่อยเป็นไปอย่างอัตโนมัติ หากมีการกดหรือปล่อยเกิดขึ้น ข้อมูลนั้นจะถูกเก็บไว้ในคิวของอีเวนท์ เพื่อรอให้โปรแกรมหลักมาดึงเอาไปเพื่อใช้เป็นอินพุต แต่การที่จะให้คลาสนี้ทำงานได้ โปรแกรมหลักจะต้องเรียกฟังก์ชัน alwayscall เพื่อให้คลาสตรวจสอบสถานะของเมาส์ และสร้างอีเวนท์ขึ้น ส่วนการตรวจสอบอินพุตทางคีย์บอร์ด จะรวมอยู่ในส่วนของโปรแกรมหลัก เมื่อเกิดการกดแป้นพิมพ์ขึ้น โปรแกรมหลักจะทำการสร้างอีเวนท์ขึ้นเอง การใช้อีเวนท์คิวในการประมวลผล ทำให้อินพุตสามารถรวมเป็นรูปแบบเดียวกัน เพื่อให้การออกแบบและประมวลผลอินพุตง่ายขึ้น

การทำงานทั้งหมดเกี่ยวกับเมาส์ จะใช้ปุ่มซ้ายในการทำงาน อินพุตจากคีย์บอร์ดจะใช้ในเรื่องของการติดต่อ การรับการติดต่อและยกเลิกการติดต่อ การค้นหาผู้ใช้ และการออกจากโปรแกรม รวมไปถึงการพิมพ์ข้อความบนหน้าจอและส่งไปยังเครื่องรับ



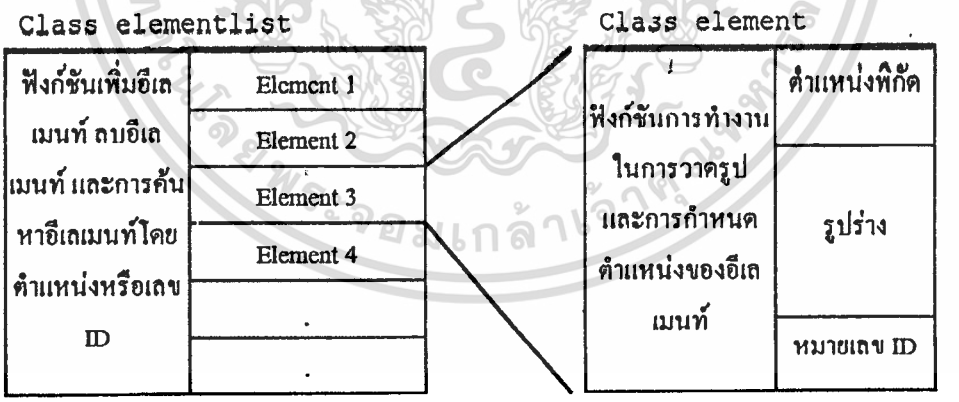
รูปที่ 3.12 - การทำงานของอีเวนท์คิว

รูปแบบของ Event message ที่ส่งให้กับโปรแกรมหลัก จะมีรายละเอียดเกี่ยวกับประเภทของอีเวนท์ และตำแหน่งที่เกิดอีเวนท์ เช่น เกิดการกดเมาส์ปุ่มขวาที่ตำแหน่ง (200,157) เป็นต้น ถ้าเป็นอีเวนท์ประเภทกดแป้นพิมพ์ จะไม่มีการบอกตำแหน่ง แต่จะเก็บค่าตัวอักขระไว้ที่ตัวแปรที่บอกตำแหน่งในแกน x ของอีเวนท์

การวาดรูป ผู้ใช้ต้องเลือกอุปกรณ์ที่ใช้ในการวาด ซึ่งมีอยู่ 5 แบบ คือ ดินสอ อุปกรณ์ลากเส้นตรง วาดวงกลม วาดสี่เหลี่ยม และเขียนข้อความ สามารถเลือกสีได้โดยการใช้เมาส์เลือก หรือกดปุ่ม Home และ End ส่วนการเลือกความหนาของเส้น ใช้การกดปุ่ม PageUp และ PageDown

การตรวจสอบว่า ในตำแหน่งที่มีการกดเมาส์นั้นควรจะทำอะไร จะใช้การตรวจสอบจากอีเลเมนท์ที่มีอยู่บนจอแต่ละตัว ว่ามีตำแหน่งตรงกับตำแหน่งเมาส์หรือไม่ โดยการสร้างอีเลเมนท์ขึ้นมาจะใช้คลาสที่สร้างขึ้นชื่อว่าคลาส element ซึ่งแต่ละอีลีเมนท์จะประกอบด้วยรายละเอียดเกี่ยวกับตำแหน่งที่อยู่ของอีเลเมนท์ รูปแบบของอีเลเมนท์ และฟังก์ชันการทำงานเมื่ออีเลเมนท์นั้นได้รับอีเวนท์ การใช้อีเลเมนท์จะใช้ในส่วนที่เป็นปุ่มอุปกรณ์เกี่ยวกับการวาด เมื่อได้สร้างอีเลเมนท์ขึ้นมาแล้ว จะนำเอาอีเลเมนท์ที่สร้างขึ้นไปรวมกันโดยใช้คลาส elementlist และใช้ฟังก์ชันของคลาสนี้ในการตรวจสอบตำแหน่งของทุกอีเลเมนท์ การนำเอาคลาส elementlist มารวมอีเลเมนท์ก็เพื่อให้สามารถตรวจสอบทุกอีเลเมนท์ได้ และสามารถเพิ่ม ลบหรือค้นหาอีเลเมนท์ที่ต้องการได้ง่าย โดยใช้เมธอด (method) ของคลาส elementlist

การค้นหาผู้ใช้บนเครือข่าย ใช้ปุ่ม F10 การติดต่อโดยใช้หมายเลขอินเตอร์เน็ตเวิร์กแอดเดรส ใช้ปุ่ม F9 การเคลียร์หน้าจอ ใช้ปุ่ม Ctrl-A การยกเลิกการติดต่อ ใช้ปุ่ม Ctrl-D การเลือกว่า จะวาดรูปโดยระบายสีในรูปหรือไม่ ใช้ปุ่ม F1 และออกจากโปรแกรม ใช้ปุ่ม ESC



รูปที่ 3.13 - คลาสอีเลเมนท์ และอีเลเมนท์ลิสต์

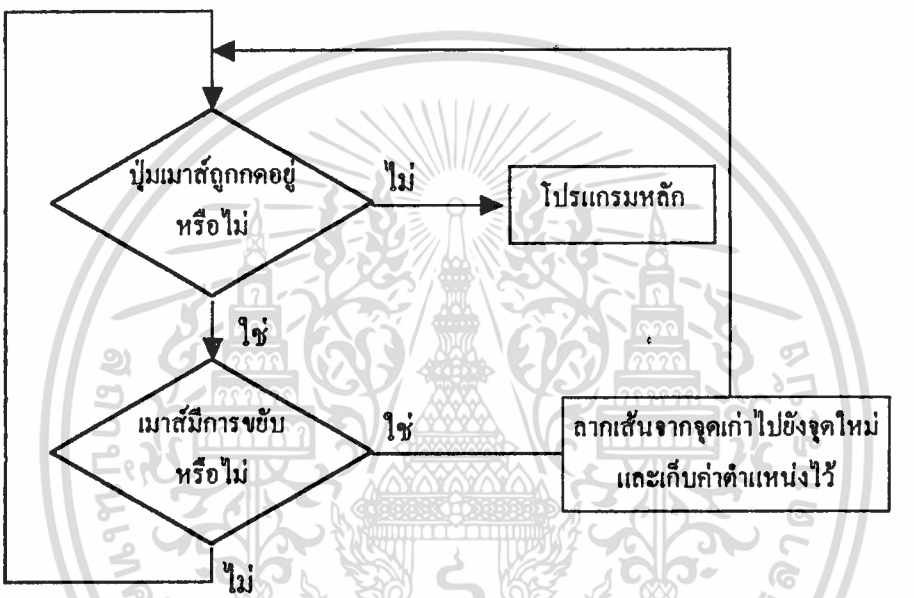
3.6.2 การแสดงผล

การแสดงผล กระทำที่ความละเอียด 640x480 ใช้สี 16 สี โดยใช้ไลบรารีพื้นฐานของภาษาซี ในการเขียนพิกเซล วาดรูปสี่เหลี่ยมและสี่เหลี่ยมทึบมาประยุกต์ใช้ในการสร้างฟังก์ชันเพื่อลากเส้น สี่เหลี่ยมและวงกลมที่สามารถกำหนดความหนาของเส้นได้ การเขียนตัวอักษรก็ใช้ไลบรารีพื้นฐานของภาษาซีเช่นเดียวกัน ถึงแม้ว่าฟังก์ชันพื้นฐานของภาษาซีจะมีความเร็วไม่สูงเท่ากับการเขียนขึ้นเอง แต่ก็เพียงพอสำหรับการทำงานที่ไม่

ต้องการความเร็วสูงมากนัก และเมื่อมีการใช้เมาส์ ก่อนการแสดงผลทุกครั้งต้องมีการซ่อนตัวชี้ของเมาส์ มิเช่นนั้นจะทำให้รูปตัวชี้ของเมาส์เลือนไป

3.6.2.1 การแสดงผลการลากเส้นแบบฟรีแฮนด์

การลากเส้นฟรีแฮนด์ เมื่อผู้ใช้กดเมาส์ไว้ โปรแกรมจะตรวจสอบตำแหน่งของเมาส์อยู่เสมอ หากเมาส์มีการขยับ ก็จะลากเส้นด้วยความหนาและสีที่ผู้ใช้ได้เลือกไว้ จากจุดเก่าไปยังจุดใหม่ที่เมาส์อยู่ และทำอย่างนี้ต่อไปเรื่อย จนกว่าผู้ใช้จะปล่อยปุ่มเมาส์



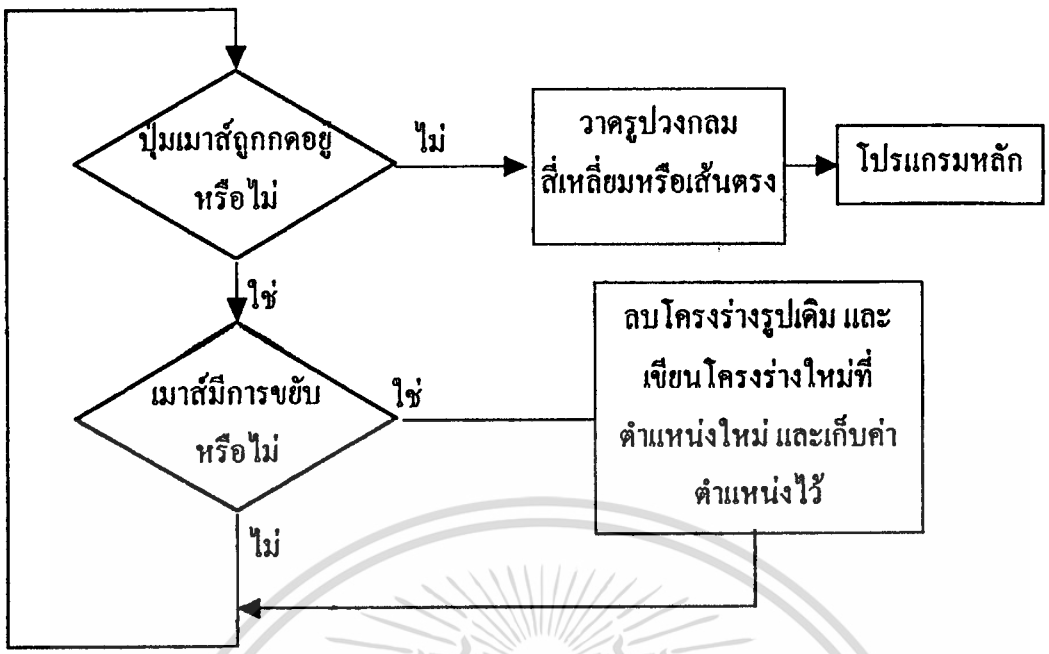
รูปที่ 3.14 - อัลกอริทึมการแสดงผลลากเส้นแบบฟรีแฮนด์

3.6.2.2 การแสดงผลการลากเส้นตรง สี่เหลี่ยม และวงกลม

รูปแบบในการแสดงผลในการลากเส้นตรง สี่เหลี่ยม และวงกลม เมื่อผู้ใช้กดเมาส์นั้นมีรูปแบบที่คล้ายกัน โดยมีอัลกอริทึมดังรูป 3.15

3.6.2.3 การแสดงผลข้อความ

การแสดงผลข้อความนั้น จะรับข้อมูลที่เป็นตัวอักขระเข้ามา เมื่อตรวจสอบว่าเป็นตัวอักษรก็จะแสดงผลออกทางหน้าจอต่อจากข้อความเดิม แต่หากเป็นการกด Enter ก็จะทำการขึ้นบรรทัดใหม่ ถ้าหากเป็นการกดแป้น Backspace ก็จะทำการลบตัวอักษรที่แล้วทิ้งไป หากเกิดการกดเมาส์ขึ้น ก็จะไปเริ่มพิมพ์ข้อความใหม่ ณ ตำแหน่งที่เมาส์ถูกกด ซึ่งก่อนที่จะมีการเริ่มพิมพ์ข้อความโดยผู้ใช้ จะมีการเก็บรูปด้านหลังในแถวที่ข้อความนั้นอยู่ไว้ก่อน เพื่อใช้แสดงผลในกรณีที่มีการกด Backspace เพื่อลบตัวอักขระ



รูปที่ 3.15 - อัลกอริทึมการแสดงผลเมื่อมีการวาดเส้นตรง สี่เหลี่ยมและวงกลม

3.6.2.4 ปุ่มอุปกรณ์ต่างๆ

ปุ่มอุปกรณ์ต่างๆ ที่ใช้สำหรับเลือกสิ่งที่ต้องการวาด จะมีลักษณะเป็นสี่เหลี่ยม ซึ่งรูปที่แสดงนั้น ได้มาจากการดึงเอารูปจากไฟล์บิตแมปขึ้นมาแสดงบนหน้าจอ เมื่อมีการเลือกปุ่มใดๆ จะมีการเขียนรูปสีเหลี่ยมไว้รอบๆ ปุ่ม เพื่อบ่งบอกว่ากำลังใช้อุปกรณ์นี้อยู่ ในส่วนของปุ่มเลือกสี จะตรวจสอบตำแหน่งการกดเมาส์ของผู้ใช้ และตรวจสอบว่าอยู่ตรงกับสีใด แล้วจึงวาดสี่เหลี่ยมรอบสีนั้น

3.7 โปรแกรมหลัก

3.7.1 หน้าที่ของโปรแกรมหลัก

ส่วนของโปรแกรมหลัก มีหน้าที่ดำเนินงานและประสานงานระหว่างส่วนต่างๆ ให้สามารถเชื่อมโยงกันและทำงานได้ การทำงานในส่วนนี้จะใช้การวนลูปเพื่อตรวจสอบค่าในคิวข้อมูลส่ง คิวข้อมูลรับ อีเวนท์คิว และข้อมูลในบัฟเฟอร์ เพื่อทำการนำข้อมูลเหล่านั้นมาประมวลผลและส่งให้กับส่วนอื่นต่อไปอย่างเหมาะสม โปรแกรมหลักมีหน้าที่สร้างอีเลเมนท์ทั้งหมดขึ้นมา แล้วนำไปรวมกันในคลาส Elementlist การทำงานเกี่ยวกับการแสดงผลบางอย่าง นอกจากนี้ โปรแกรมหลักยังมีหน้าที่เรียกฟังก์ชัน alwayscall ของคลาส mouse และ IPX เพื่อให้ฟังก์ชันทั้งสองได้มีโอกาสทำงานในบางส่วนที่กำหนดไว้

หลังจากส่วนอินเตอร์เฟสและส่วนการรับส่งข้อมูลสามารถทำงานได้ เราจะทำการนำเอาทั้งสองส่วนมาเชื่อมต่อกัน โดยใช้โปรแกรมหลักเป็นตัวเชื่อมต่อ โปรแกรมหลักมีหน้าที่ประสานงานทั้งสองฝ่าย กล่าวคือ เมื่อมีอินพุตจากผู้ใช้เข้ามา โปรแกรมหลักก็จะส่งต่อให้กับส่วนประมวลผลเพื่อแสดงผล และทำการส่งข้อมูลกราฟิกออกมาเก็บไว้ในคิวข้อมูลที่ใช้สำหรับการส่ง เพื่อให้โปรแกรมหลักนำไปส่งให้กับส่วนรับส่งข้อมูล ในด้านรับ โปรแกรมหลักก็จะนำข้อมูลส่งให้กับส่วนประมวลผลเพื่อแสดงผลออกมา

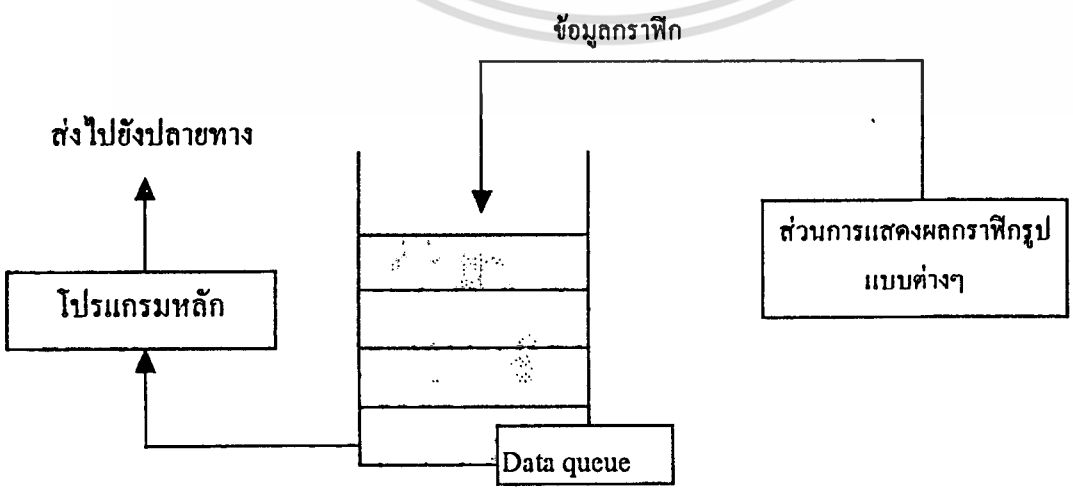
รูปแบบของข้อมูลกราฟิกที่จะใช้ส่งเพื่อติดต่อสื่อสารกัน ได้กำหนดไว้ดังนี้

```
struct drawinfo
{
    int ID; /* use to identify shape */
    int x1,y1,x2,y2; /* y2 = ratio1, x2 = ratio2 */
    int radius;
    int fillcol; /* FillColor, -1 if not fill */
    int col; /* draw color */
    int thickness;
    char text[MAXTEXTLONG];
};
```

รูปที่ 3.16 - โครงสร้างของ drawinfo

ตัวแปร ID จะเป็นตัวที่บอกว่า ข้อมูลชุดนี้เป็นข้อมูลที่ใช้ในการวาดรูปแบบใด ซึ่งได้มีการกำหนดไว้ในส่วนต้นของโปรแกรม ค่าตัวแปร fillcol, col และ thickness คือค่าสีที่ใช้ระบาย, สีของเส้น และความหนาของเส้นตามลำดับ ค่า (x1,y1) และ (x2,y2) เป็นค่าพิกัดของรูปสี่เหลี่ยมหรือเส้นตรง ส่วนในการส่งข้อมูลในการวาดวงกลม จะใช้ตัวแปร (x1,y1) ในการกำหนดจุดศูนย์กลาง radius ในการกำหนดรัศมี และ y2/x2 เป็นอัตราส่วนระหว่างแกน y และแกน x และหากเป็นการเขียนข้อความ ข้อความจะอยู่ในสตริง text ที่มีความยาวเท่ากับ MAXTEXTLONG

ในการวาดรูปแต่ละครั้ง ฟังก์ชันการวาดรูปต้องส่งข้อมูลให้กับโปรแกรมหลัก เพื่อให้โปรแกรมหลักทำการส่งข้อมูลนั้นไปยังเครื่องอีกฝั่งหนึ่ง การส่งจะใช้ queue เข้ามาช่วยในการส่ง โดยทุกๆ ฟังก์ชันในการวาดภาพ จะทำการนำข้อมูลเกี่ยวกับการวาด มาต่อท้าย queue และโปรแกรมหลักก็จะนำเอาข้อมูลจาก queue ส่งออกไปอีกทีหนึ่ง การใช้ queue ในการส่งข้อมูลก็เพื่อป้องกันการส่งข้อมูลไม่ทัน โดยหากส่งข้อมูลได้ไม่ทัน ข้อมูลก็จะยังคงเก็บอยู่ใน queue โดยไม่สูญหายไป จนกว่าจะสามารถส่งข้อมูลออกไปได้



รูปที่ 3.17 - การใช้ Data queue

คำสั่งในการใช้งาน queue มีอยู่ 3 รูปแบบ คือ การดึงข้อมูลออกจากหัว queue การเพิ่มข้อมูลต่อท้าย และการเคลียร์ข้อมูลทั้งหมด

3.7.2 ขั้นตอนการทำงานของโปรแกรมหลัก

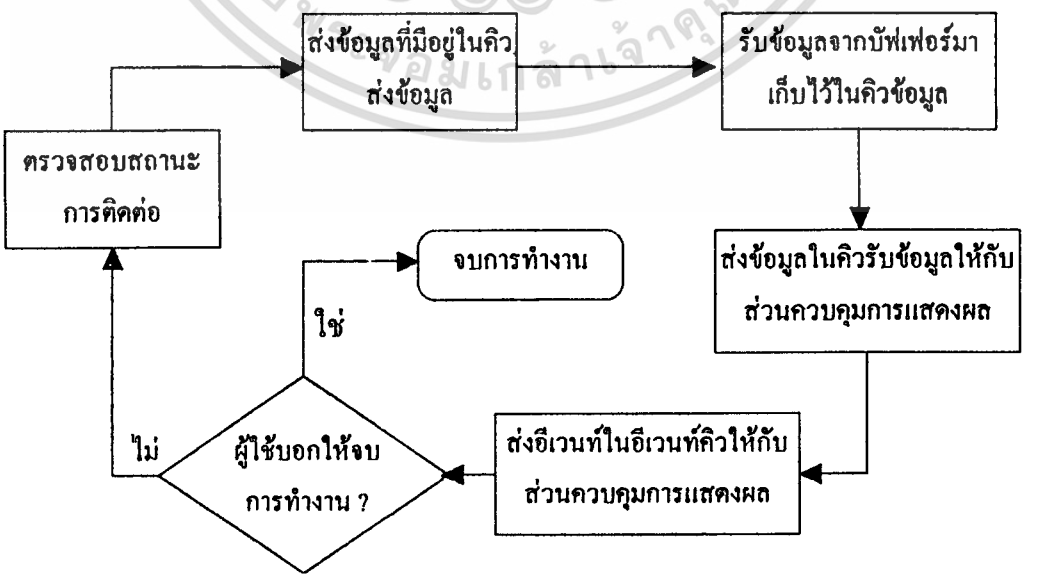
3.7.2.1 ขั้นตอนเริ่มต้น (Initialize)

ก่อนที่โปรแกรมจะเริ่มการทำงาน จะต้องมีการตั้งค่าเริ่มต้นในส่วนต่างๆ การตรวจสอบว่ามีโปรโตคอล IPX รองรับอยู่ในระบบหรือไม่ ตรวจสอบการเริ่มการทำงานในโหมดกราฟิก ซึ่งหากไม่มีโปรโตคอล IPX อยู่ในระบบ หรือกราฟิกไม่สามารถเริ่มทำงานได้ โปรแกรมหลักจะฟ้องข้อผิดพลาด และออกจากโปรแกรมทันที แต่ถ้าทุกอย่างปกติ โปรแกรมหลักก็จะทำการสร้างและวาดอีเลเมนท์ต่างๆ และนำไปรวมกันในคลาส elementlist ต่อไป

3.7.2.2 ขั้นตอนการทำซ้ำ (Loop)

การทำงานของโปรแกรมหลัก จะทำการวนรอบซ้ำๆ เพื่อตรวจรับและส่งข้อมูล และการประสานงานในส่วนต่างๆ โดยมีการทำงานดังนี้

1. ตรวจสอบสถานะการติดต่อ และทำงานตามที่ได้กำหนดไว้ในสถานะนั้นๆ เช่น เมื่อสถานะเป็น WAITFORUSERCONFIRM หมายถึง มีการติดต่อเข้ามาจากเครื่องอื่น ก็จะถามกับผู้ใช้ว่า มีผู้ต้องการจะติดต่อด้วย จะรับการติดต่อหรือไม่ เป็นต้น
2. ส่งข้อมูลที่มีอยู่ในคิวส่งข้อมูลออกไป เท่าที่จะเป็นไปได้
3. รับข้อมูลที่มีอยู่ในบัฟเฟอร์ มาเก็บไว้ในคิวรับข้อมูล
4. ส่งข้อมูลในคิวรับข้อมูลและอีเวนท์จากอีเวนท์คิวให้กับส่วนควบคุมการแสดงผล
5. หากไม่มีการบอกเลิกการใช้งานจากผู้ใช้ ให้กลับไปทำข้อ 1 อีก



รูป 3.18 - โฟลวชาร์ตแสดงการทำงานในส่วนการทำซ้ำของโปรแกรมหลัก

3.7.2.3 ขั้นตอนการจบการทำงาน

ขั้นตอนนี้คือการทำงานในส่วนก่อนที่จะออกจากโปรแกรม จะเป็นการลบ (delete) ออบเจ็กต์ที่ใช้อยู่ออกจนหมด เพื่อเคลียร์หน่วยความจำ ทำการปิดซอกเก็ตของ IPX ที่เปิดอยู่ทั้งหมด และออกจากกราฟิกโหมด

3.8 รูปแบบของสัญญาณและสถานะการติดต่อ

รูปแบบของสัญญาณและสถานะการติดต่อ จะถูกกำหนดไว้เป็นค่าคงที่ในส่วนของไฟล์ IPXCLAS4.H เพื่อใช้แทนรูปแบบต่างๆ ของสัญญาณและสถานะการติดต่อ โดยสัญญาณจะถูกประมวลผลในส่วนของส่วนควบคุมการรับส่งข้อมูล และสถานะการติดต่อจะถูกใช้โดยโปรแกรมหลัก

3.8.1 รูปแบบของสัญญาณ มี 7 รูปแบบ ดังนี้

- REQCONN คือสัญญาณที่ใช้ส่งเพื่อร้องขอการติดต่อกับอีกฝั่งหนึ่ง
- ACCEPTCONN คือสัญญาณที่ใช้ส่งเมื่อมีการร้องขอการติดต่อเข้ามา เพื่อยอมรับการติดต่อ
- CONFIRMCONN คือสัญญาณที่ใช้ส่งเป็นการยืนยันเมื่อได้รับสัญญาณ ACCEPTCONN
- DISCONN ใช้ส่งเมื่อต้องการบอกเลิกการติดต่อ
- DISCONNACK ใช้ตอบรับการบอกเลิกการติดต่อ
- FINDUSER ใช้สัญญาณนี้ในการบรอดคาสติงเพื่อค้นหาผู้ใช้ในเครือข่าย โดยจะมีการส่งชื่อที่ต้องการทำการค้นหาไปกับสัญญาณนี้ด้วย
- FINDUSERACK ในกรณีที่ชื่อในสัญญาณ FINDUSER ที่ถูกบรอดคาสติงออกมา ตรงกับชื่อของเครื่องใดเครื่องหนึ่ง เครื่องนั้นจะตอบรับโดยใช้สัญญาณ FINDUSERACK

3.8.2 สถานะการติดต่อ มีอยู่ 7 สถานะ ดังนี้

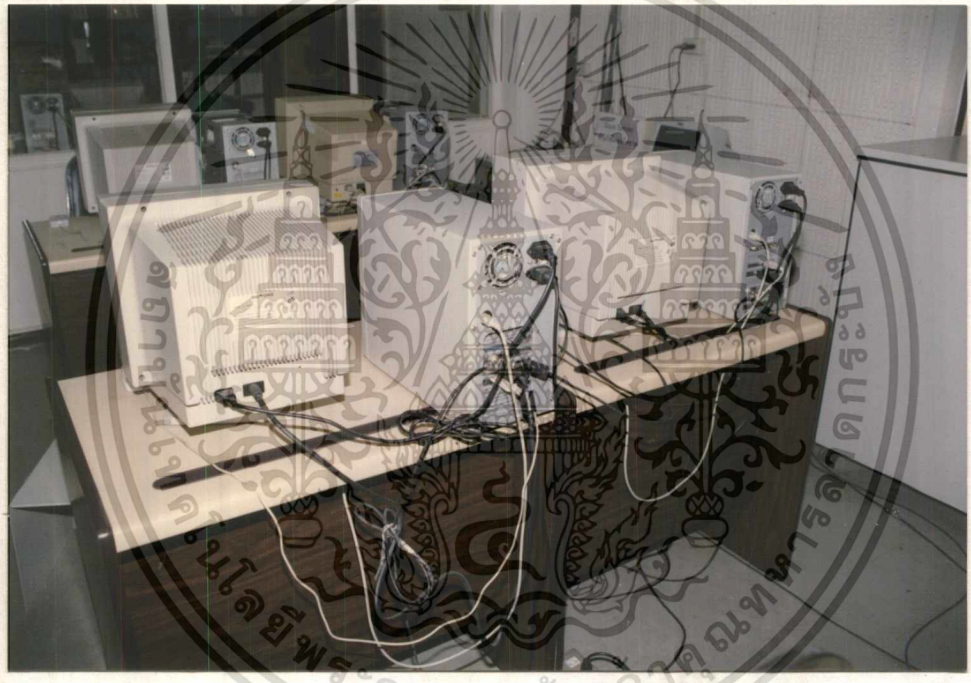
- AVAIL หมายถึงช่องส่งข้อมูลว่าง และยังไม่มีการรอรับการติดต่อ
- WAITING หมายถึงกำลังรอการติดต่อ
- CONNECTING หมายถึง ได้ส่งสัญญาณร้องขอการติดต่อออกไปแล้ว และกำลังรอการตอบรับ
- WAITFORCONFIRM หมายถึง กำลังรอสัญญาณ CONFIRMCONN จากอีกฝั่งหนึ่ง
- CONNECTED หมายถึงการติดต่อสำเร็จ
- DISCONNECTING หมายถึง ได้ส่งสัญญาณบอกเลิกการติดต่อออกไปแล้ว และกำลังรอสัญญาณ DISCONNACK
- DISCONNECTED หมายถึง การยกเลิกการติดต่อเสร็จสิ้น

บทที่ 4

การทดลองและผลการทดลอง

4.1 ขั้นตอนการทดลอง

ในการทดลอง จะทดลองในการติดต่อระหว่างเครื่องทั้งสอง ว่าสามารถใช้งานได้จริง โดยจะทดลองการติดต่อทั้งแบบธรรมดา แบบบรอดคาสติง และแบบค้นหาผู้ใช้ การยกเลิกการติดต่อ หลังจากนั้นจะทำการทดลองส่งข้อมูลและการแสดงผลในการวาดรูปด้วยอุปกรณ์ต่างๆ โดยการใช้งานเครื่องทั้งสองพร้อมกัน ว่าเป็นไปอย่างถูกต้อง โดยเครื่องคอมพิวเตอร์ที่ใช้ทดลองจะต่อเชื่อมกับเครือข่ายวงเดียวกัน โดยใช้สายโคแอกเซียล ความเร็ว 10 Mbit/s และเน็ตเวิร์กอแดปเตอร์แบบอีเทอร์เน็ต (Ethernet)

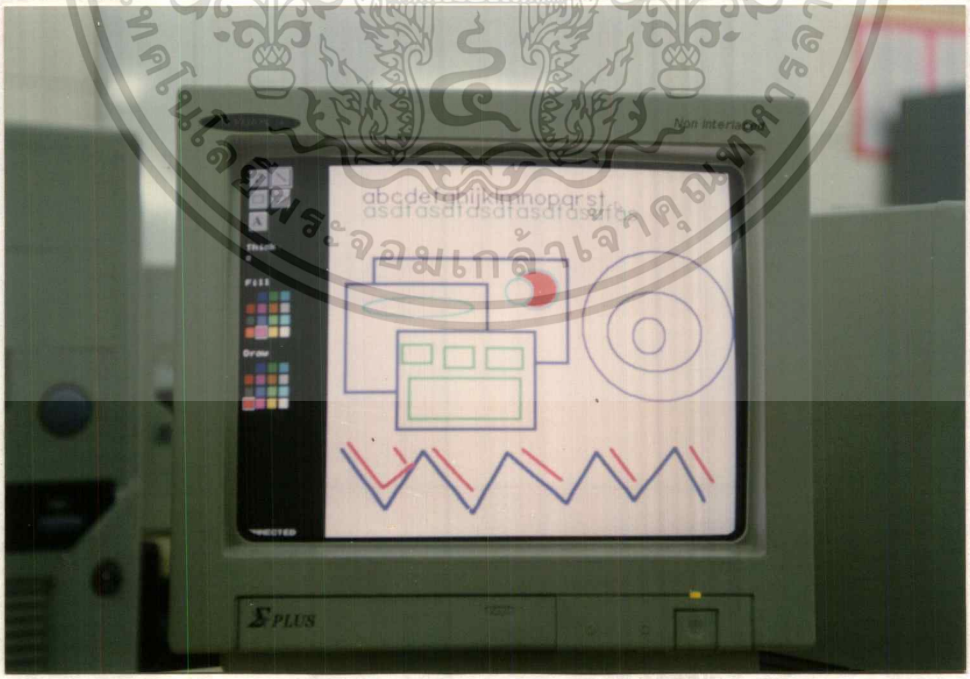


รูปที่ 4.1 - การเชื่อมต่อกันระหว่างเครื่องทั้งสอง

4.2 ผลการทดลอง

การติดต่อทั้งแบบบรอดคาสติงและแบบโดยตรงเป็นไปอย่างถูกต้อง การส่งสัญญาณค้นหาผู้ใช้บนเครือข่ายถูกต้องและใช้งานได้ การทดลองการใช้งานในการส่งข้อมูลแบบกราฟิกในการสี่เหลี่ยม ลากเส้นตรง ลากแบบฟรีแฮนด์ และการเขียนข้อความเป็นไปอย่างถูกต้อง และข้อมูลไปถึงยังอีกเครื่องหนึ่งครบถ้วน แต่ในส่วนของวงกลมจะเกิดปัญหาในส่วนการแสดงผล ถ้าหากเขียนเป็นรูปวงรีที่รัศมีในแกนนอนต่ำมากๆ การยกเลิกการติดต่อใช้งานได้ และการติดต่อใหม่หลังยกเลิกการติดต่อสามารถกระทำได้อย่างถูกต้อง ส่วนอินเตอร์เฟสและการรับอินพุตไม่พบปัญหาใดๆ

รูปที่ 4.2 - แสดงการรับส่งข้อมูลแบบกราฟิค



รูปที่ 4.3 - รูปแบบของกราฟิกอินเตอร์เฟส

บทที่ 5
บทวิจารณ์และบทสรุป

ในการสร้างโปรแกรมเพื่อใช้ในการติดต่อสื่อสารนั้น สิ่งที่สำคัญคือ การสร้างโปรแกรมให้ติดต่อสื่อสารอย่างมีเสถียรภาพและไม่ผิดพลาด การติดต่อกับผู้ใช้ ความเสถียรภาพของโปรแกรมเอง และความรัดกุมของโปรโตคอลชั้นสูงที่สร้างขึ้นมา ในส่วนอัลกอริทึมนั้นมีส่วนที่สำคัญคือ การเปลี่ยนอินพุตจากผู้ใช้ที่เข้ามาให้กลายเป็นข้อมูลที่สามารถส่งไปได้ การโปรแกรมแบบทำงานหลายอย่างในเวลาเดียวกัน (เช่น สามารถรับข้อความได้ ขณะที่กำลังลากเส้นอยู่) อัลกอริทึมการวาดรูปเรขาคณิตต่าง ๆ ฯลฯ ซึ่งการออกแบบไม่ว่าในส่วนใดก็ตาม ควรจะต้องคำนึงถึงประโยชน์ที่ผู้ใช้จะได้รับ ไม่ว่าในด้านความรวดเร็ว หรือความเชื่อถือได้ของโปรแกรมก็ตาม

โปรแกรมที่ได้สร้างขึ้น นับว่ามีส่วนต่างๆ ดังที่กล่าวมาอยู่เกือบครบถ้วน แต่อาจจะขาดความรัดกุมของโปรโตคอลที่สร้างขึ้นไปบ้าง แต่ก็นับว่าสามารถใช้ติดต่อสื่อสารได้ การส่งข้อมูลมีการใช้เลขลำดับ และการป้องกันการไหลของข้อมูลไม่ให้มีมากเกินกว่าฝั่งรับจะรับทัน ทำให้ข้อมูลไม่สูญหายและทั้งสองฝั่งจะสามารถทำงานได้อย่างถูกต้อง ในส่วนอินเตอร์เฟสนับว่าค่อนข้างเพียงพอต่อการติดต่อสื่อสาร แต่อาจยังขาดในด้านความสวยงามและความสามารถบางอย่างไป ซึ่งสามารถที่จะพัฒนาต่อไปได้

ในการทำโครงงานนี้ นับว่าประสบความสำเร็จค่อนข้างสมบูรณ์ แต่ประสิทธิภาพของงานที่ได้ยังไม่ดีนักในบางส่วน เช่น ส่วนอินเตอร์เฟสที่ไม่สวยงาม และใช้สีได้เพียง 16 สี เป็นต้น การทำงานในด้านการสื่อสารโดยโปรโตคอล IPX เป็นไปอย่างยากลำบาก เนื่องจาก แม้ว่าโปรโตคอล IPX จะใช้กันอย่างแพร่หลาย แต่ไม่เป็นที่นิยมนำมาใช้สร้างแอพพลิเคชันในการติดต่อสื่อสาร ดังนั้นหนังสือและเอกสารอ้างอิงจึงมีน้อย จึงต้องใช้การทดลองเป็นส่วนใหญ่ เพื่อทดสอบผลของการทำงานในแต่ละฟังก์ชันเอง

นับได้ว่า การโปรแกรมแบบออบเจ็กต์ออเรียนเต็ด (Object-Oriented) ที่ใช้ในโครงงานนี้ มีประโยชน์มาก เนื่องจากสามารถทำให้การสร้างระบบในแต่ละส่วนค่อนข้างจะเป็นอิสระต่อกัน ทำให้การเปลี่ยนแปลงแก้ไขตรวจสอบโปรแกรมเป็นไปได้ง่าย ดูเป็นรูปแบบและพัฒนาได้ง่ายกว่าการโปรแกรมแบบเดิม แต่อย่างไรก็ดี การโปรแกรมยังมีในบางจุดที่ไม่ได้ประยุกต์การใช้ออบเจ็กต์เข้าไป เนื่องจากเป็นฟังก์ชันหรือการทำงานพื้นฐาน ซึ่งไม่สามารถระบุได้ว่าควรจะอยู่ในส่วนไหน

แนวทางการพัฒนาในโครงงานนี้ อาจจะใช้คลาสพื้นฐานที่ได้สร้างขึ้น มาทำการพัฒนาให้มีความน่าเชื่อถือมากยิ่งขึ้น หรือพัฒนาทางด้านการอินเตอร์เฟสและการรับข้อมูลให้ดียิ่งขึ้น คลาสพื้นฐานที่สร้างขึ้น หากทำการพัฒนาอาจสามารถนำไปใช้ในงานอื่นๆ เช่นการส่งเสียงพูดหรือภาพเคลื่อนไหวผ่านเครือข่ายได้ และหากต้องการพัฒนาให้สามารถติดต่อผ่านโปรโตคอลอื่นๆ ก็สามารถทำได้โดยการเปลี่ยนแปลงในคลาส IPX ที่ได้สร้างขึ้น เพื่อให้เป็นการติดต่อในโปรโตคอลอื่นๆ แทน

ภาคผนวก ก. -- อินเตอร์รัพท์ของเมาส์

```
FUNCTION: AX = 0h
    Description: Determines whether a mouse is available and if it is
                 initializes the mouse driver.
    Returns    : AX = Non-Zero (If Mouse is installed, 0 if not)
                 BX = Number of Mouse Buttons

FUNCTION: AX = 1h
    Description: Increments the mouse cursor display counter.
    Returns    : Nothing

FUNCTION: AX = 2h
    Description: Decrements the mouse cursor display counter.
    Returns    : Nothing

FUNCTION: AX = 3h
    Description: Returns the current mouse position and button status.
    Returns    : BX = Buttons status:
                  Bit 0: Left Button
                  Bit 1: Right Button
                  Bit 2: Center Button
                 CX = Horizontal Coordinate
                 DX = Vertical Coordinate

FUNCTION: AX = 4h
    Description: Moves the mouse cursor to a certain position on the
screen.
    Call  With : CX = Horizontal Coordinate
                 DX = Vertical Coordinate
    Returns    : Nothing

FUNCTION: AX = 5h
    Description: Reports on the status and numbers of presses for a
                 button.
    Call with  : BX = Button to Check
                 0 = Left Button
                 1 = Right Button
                 2 = Center Button
    Returns  : AX = Button Status
                 Bit 0 = Left Button
                 Bit 1 = Right Button
                 Bit 2 = Center Button
                 BX = Button Press Counter
                 CX = Horizontal Coordinate of Last Button Press
                 DX = Vertical Coordinate of Last Button Press

FUNCTION: AX = 6h
    Description: Gets the button release information.
    Call With  : BX = Button to Query
                  0 = Left Button
                  1 = Right Button
                  2 = Center Button
    Returns  : AX = Button Status(1 if pressed)
                 Bit 0 = Left Button
                 Bit 1 = Right Button
                 Bit 2 = Center Button
               BX = Button Release counter
               CX = Horizontal Coordinate of last button release
               DX = Vertical Coordinate of last button release
```

```
FUNCTION: AX = 7h
        Description: Sets the horizontal limits for the mouse cursor.
    Calls With : CX = Minimum horizontal mouse coordinate.
                 DX = Maximum horizontal mouse coordinate.

FUNCTION: AX = 8h
    Description: Sets the vertical limits for the mouse cursor.
        Call With  : CX = Minimum vertical mouse coordinate.
             DX = Maximum vertical mouse coordinate.
    Returns    : Nothing

FUNCTION: AX = 9h
    Description: Defines the shape of the graphics mode cursor.
    Call With  : BX = Hoorizontal hot spot offset
             CX = Vertical hot spot offset
             ES = Segment of buffer containing cursor mask
             DX = Offset of buffer containing cusror mask

    Returns    : Nothing

FUNCTION: AX = 0Ah
        Description: Definesthe shape of the text mode cursor
    Call With  : BX = Cursor Type
                 0 = Software Cursor
            1 = Hardware cursor
            if BX = 0 then
              CX = Screen Mask value
              DX = Cursor Mask Value
            else
           CX = Starting Scan Line For Cursor
           DX = Ending Scan Line For Cursor

FUNCTION: AX = 0Bh
    Description: Returns the net mouse movement since the last call
             to this function(or since the mouse was initialized).
    Returns    : CX = Horizontal mouse movement(in Mickeys)
             DX = Vertical mouse movement(in Mickeys)

FUNCTION: AX = 0Ch
    Description: Sets the user defined mouse handler.
    Call With  : CX = Event Mask

        Bit   If set
         0    Mouse Cursor Movement
         1    Left Button Pressed
         2    Left Button Released
         3    Right Button Pressed
         4    Right Button Released
         5    Center Button Pressed
         6    Center Button Released

             ES = Segment of your mouse handler code
             DX = Offset of your mouse handler code
    Returns    : AX = Event Mask
             BX = Button Status(1 if pressed)
               Bit 0 = Left Button
               Bit 1 = Right Button
               Bit 2 = Center Button
                     CX = Horizontal Coordinate
             DX = Vertical Coordinate
```

ภาคผนวก ข. – อินเตอร์รัพท์ของ IPX ไดรเวอร์

```
--------N-7A----------------------------------
INT 7A - Novell NetWare - LOW-LEVEL API - Notes
Note:   this interrupt is used for IPX/SPX access in NetWare versions through
          2.0a; in later versions, you should use INT 2F/AX=7A00h to get an
          entry point even though INT 7A still exists.  For both INT 7A and
          the FAR entry point, BX contains the function number; IPX is
          sometimes called internally with BX bit 15 set, which causes the
          handler to bypass some initial checks and an optional call to the
          IPX Windows support handler set with INT 2F/AX=7AFFh/BX=0000h
          (see #1514)
SeeAlso: INT 2F/AX=7A00h,INT 64"Novell",INT 7A/BX=0000h

--------N-7A----BX0000----------------------
INT 7A - Novell NetWare - IPX Driver - OPEN SOCKET
        BX = 0000h
        AL = socket longevity
             00h open until close or terminate
             FFh open until close
        DX = socket number (high byte in DL)
             0000h dynamic allocation
             else  socket to open (see #2212)
Return: AL = return code
             00h success
                 DX = socket number
             FEh socket table full
             FFh socket already open
Notes:  TSRs which need to use sockets should set AL to FFh, non-resident
          programs should normally use AL=00h
        IPX can be configured to support up to 150 open sockets on a
          workstation, and defaults to 20
        this function is supported by Advanced NetWare 1.02+
SeeAlso: INT 7A/BX=0001h,INT 7A/BX=0004h,INT 7A/BX=0023h

(Table 2212)
Values for IPX socket number:
 0451h  File Service (NetWare Core Protocol)
 0452h  Service Advertising Protocol
 0453h  Routing Information Packet
 0455h  NetBIOS Packet
 0456h  diagnostics
 0457h  server serial numbers (labeled "Copy Protection" by Lanalyzer)
 4000h-7FFFh used for dynamic allocation
 4444h  Brightwork Development's SiteLock server
 5555h  Brightwork Development's SiteLock client (workstation)
 8000h-FFFFh assigned by Novell
Note:   SiteLock is an application metering product using IPX to communicate
          between the application and the license server

--------N-7A----BX0001----------------------
INT 7A - Novell NetWare - IPX Driver - CLOSE SOCKET
        BX = 0001h
        DX = socket number (high byte in DL)
Notes:  also cancels events set by any Event Control Blocks for the socket
        the program must close all open sockets before terminating
        this function is supported by Advanced NetWare 1.02+
SeeAlso: BX=0000h

--------N-7A----BX0002----------------------
INT 7A - Novell NetWare - IPX Driver - GET LOCAL TARGET
        BX = 0002h
        ES:SI -> target internetwork address (see INT 7A/BX=000Bh)
        ES:DI -> 6-byte buffer for local target
Return: AL = return code
            00h success
               CX = expected one-way transfer time (clock ticks) for a 576-byte
                    packet
               ES:DI -> local target
            FAh unsuccessful (no path to destination)
```

```
Notes:  the internetwork address consists of a 4-byte network address followed
          by a 6-byte node address.  The local target is only a 6-byte node
          address.  If the target is in the same network, the local target is
          just the node address of target; otherwise, the local target is the
          node address of the bridge that leads to the target.
        this function may be called from inside IPX and AES Event Service
          Routines, but not from other interrupt handlers
        this function is supported by Advanced NetWare 1.02+
SeeAlso: BX=0009h

--------N-7A----BX0003----------------------
INT 7A - Novell NetWare - IPX Driver - SEND PACKET
        BX = 0003h
        ES:SI -> Event Control Block (see #2213,#2214)
Notes:  returns immediately; IPX attempts to send the packet in the background
        this function is supported by Advanced NetWare 1.02+
        this function is nearly identical to BX=000Fh, except that it always
          copies the source address into the IPX header assumed to be at the
          beginning of the first fragment
SeeAlso: BX=0004h,BX=000Fh,INT 21/AH=EEh"Novell"

Format of IPX Event Control Block:
Offset  Size    Description     (Table 2213)
 00h    DWORD   Link
 04h    DWORD   -> Event Service Routine (00000000h if none)
 08h    BYTE    in use flag (see #2215)
 09h    BYTE    completion code (see #2216)
 0Ah    WORD    (big-endian) socket number (see INT 7A/BX=0000h)
 0Ch  4 BYTEs   IPX workspace
 10h 12 BYTEs   driver workspace
 1Ch  6 BYTEs   immediate local node address
 22h    WORD    fragment count
 24h    var     fragment descriptors
                Offset  Size    Description
                 00h    DWORD   -> fragment data
                 04h    WORD    size of fragment in bytes.
Notes:  ESR is a far procedure that is called when the ECB has been handled.
          On call, the in use flag is zero if the ECB has been handled,
          non-zero otherwise. If the flag is zero, the completion code holds
          the result of the event.
        the first fragment should start with an IPX header
        all fragments are concatenated and sent in one piece
        node address FFh FFh FFh FFh FFh FFh broadcasts to all nodes

Format of AES-ECB:
Offset  Size    Description     (Table 2214)
 00h    DWORD   Link
 04h    DWORD   ESR address
 08h    BYTE    in use flag (see #2215)
 09h  5 BYTEs   AES workspace

(Table 2215)
Values for ECB in use flag:
 00h    available
 E0h    AES temporary
 F6h    \ special IPX/SPX processing for v3.02+
 F7h    /
 F8h    IPX in critical section
 F9h    SPX listening
 FAh    processing
 FBh    holding
 FCh    AES waiting
 FDh    AES couting down delay time
 FEh    awaiting packet reception
 FFh    sending packet

(Table 2216)
Values for ECB completion code:
 00h    success
 ECh    remote terminated connection without acknowledging packet
 EDh    abnormal connection termination
 EEh    invalid connection ID
```

```
  EFh    SPX connection table full
  F9h    event should not be cancelled
  FAh    cannot establish connection with specified destination
  FCh    cancelled
  FDh    malformed packet
  FEh    packet undeliverable
  FFh    physical error

(Table 2217)
Values event Service Routine is called with:
        AL = caller's identity (00h = AES, FFh = IPX)
        ES:SI -> event control block
        interrupts disabled

Format of IPX header:
Offset  Size    Description     (Table 2218)
 00h    WORD    (big-endian) checksum
 02h    WORD    (big-endian) length in bytes of total packet
 04h    BYTE    transport control
 05h    BYTE    packet type (see #2219)
 06h 10 BYTEs   destination internetwork address
 10h    WORD    (big-endian) destination socket
 12h 10 BYTEs   source internetwork address
 1Ch    WORD    (big-endian) source socket

(Table 2219)
Values for IPX packet type:
 00h    unknown packet type
 01h    routing information packet
 02h    echo packet
 03h    error packet
 04h    packet exchange packet (always use this one)
 05h    SPX packet
 11h    NetWare Core Protocol
 14h    Propagated Packet (for NetWare), NetBIOS name packet
 15h-1Eh experimental protocols
Note:   undocumented packet type 14h will cross up to 16 networks deep in
          all directions; as Aaron Martin of Origin Systems discovered, the
          first 64 bytes of the IPX data in such packets should be considered
          reserved, as IPX places the traversed server nodes there.

Format of Service Advertising Protocol Service Query Packet:
Offset  Size    Description     (Table 2220)
 00h 30 BYTEs   IPX header
 1Eh    WORD    (big-endian) query type
                0001h general find service
                0003h find nearest server
 20h    WORD    (big-endian) server type (see INT 21/AH=E3h"NetWare")

Format of Service Advertising Protocol Server Identification Packet:
Offset  Size    Description     (Table 2221)
 00h 30 BYTEs   IPX header
 1Eh    WORD    (big-endian) response type
                0002h general service
                0004h nearest service
 20h 64N BYTEs  server entries (1-7) (see #2222)

Format of SAP server entry:
Offset  Size    Description     (Table 2222)
 00h    WORD    (big-endian) server type (see INT 21/AH=E3h"NetWare")
 02h 48 BYTEs   ASCIZ server name
 32h  2 WORDs   (big-endian) network number
 34h  3 WORDs   (big-endian) node number
 3Ch    WORD    (big-endian) socket number
 3Eh    WORD    (big-endian) number of hops between caller and server

Format of IPX Routing Information packet:
Offset  Size    Description     (Table 2223)
 00h 30 BYTEs   IPX header
 1Eh    WORD    operation (0001h request, 0002h response)
 20h 8N BYTEs   network entries (1-50) (see #2224)
```

```
Format of RIP network entry:
Offset  Size    Description     (Table 2224)
 00h    DWORD   network number (FFFFFFFFh = general request)
 04h    WORD    (response) number of hops
 06h    WORD    (response) number of clock ticks to reach destination
--------N-7A----BX0004-----------------------
INT 7A - Novell NetWare - IPX Driver - LISTEN FOR PACKET
        BX = 0004h
        ES:SI -> Event Control Block (see BX=0003h)
Return: AL = status
            00h successful
            FFh no listening socket for packet
Desc:   this function provides IPX with an ECB for receiving an IPX packet, but
          does not wait for a packet to arrive
Notes:  the application must open a socket and initialize the ECB's ESR
          address, socket number, fragment count, and fragment descriptor
          fields before invoking this function
        there is no limit on the number of ECBs which may simultaneously be
          listening on a socket
        this function is supported by Advanced NetWare 1.02+
SeeAlso: BX=0000h,BX=0003h

--------N-7A----BX0005-----------------------
INT 7A - Novell NetWare - IPX Driver - SCHEDULE IPX EVENT
        BX = 0005h
        AX = delay time in clock ticks
        ES:SI -> Event Control Block (see BX=0003h)
Note:   this function is supported by Advanced NetWare 1.02+
SeeAlso: BX=0006h,BX=0007h,BX=0008h

--------N-7A----BX0006-----------------------
INT 7A - Novell NetWare - IPX Driver - CANCEL EVENT
        BX = 0006h
        ES:SI -> Event Control Block (see BX=0003h)
Return: AL = return code (see #2225)
Notes:  cannot cancel packets which the node's driver has already sent
        this function is supported by Advanced NetWare 1.02+
SeeAlso: BX=0005h

(Table 2225)
Values for IPX return code:
 00h    success
 F9h    event in use
 FCh    event cancelled
 FFh    unsuccessful, event not in use, or unrecognized ECB flag

--------N-7A----BX0007-----------------------
INT 7A - Novell NetWare - IPX Driver - SCHEDULE SPECIAL EVENT
        BX = 0007h
        AX = delay time
        ES:SI -> Event Control Block (see BX=0003h)
Note:   this function is supported by Advanced NetWare 1.02+
SeeAlso: BX=0006h

--------N-7A----BX0008-----------------------
INT 7A - Novell NetWare - IPX Driver - GET INTERVAL MARKER
        BX = 0008h
Return: AX = interval marker in clock ticks
Notes:  may be used to measure the time elapsed between two events, up to one
          hour
        this function is supported by Advanced NetWare 1.02+
SeeAlso: BX=0005h

--------N-7A----BX0009-----------------------
INT 7A - Novell NetWare - IPX Driver - GET INTERNETWORK ADDRESS
        BX = 0009h
        ES:SI -> buffer for own internetwork address (see #2226)
Return: ES:SI buffer filled
        SI destroyed
Note:   this function is supported by Advanced NetWare 1.02+
SeeAlso: BX=0002h,BX=000Bh
```

```
Format of IPX internetwork address:
Offset  Size    Description     (Table 2226)
 00h  4 BYTEs   (big-endian) network number
 04h  6 BYTEs   (big-endian) node number within network

--------N-7A----BX000A----------------------
INT 7A - Novell NetWare - IPX Driver - RELINQUISH CONTROL
        BX = 000Ah
Desc:   this call indicates that the application is idle and permits the IPX
          driver to do some work
Note:   this function is supported by Advanced NetWare 1.02+
SeeAlso: INT 15/AX=1000h,INT 21/AH=89h,INT 2F/AX=1680h

--------N-7A----BX000B----------------------
INT 7A - Novell NetWare - IPX Driver - DISCONNECT FROM TARGET
        BX = 000Bh
        ES:SI -> internetwork address (see #2227)
Notes:  this function permits the network software on the remote machine to
          remove any virtual connection with the calling machine
        only use in point-to-point networks
        should never be called from within an Event Service Routine
        this function is supported by Advanced NetWare 1.02+
SeeAlso: BX=0002h,BX=0009h

Format of IPX internetwork address:
Offset  Size    Description     (Table 2227)
 00h  4 BYTEs   (big-endian) destination network
 04h  6 BYTEs   (big-endian) destination node
 0Ah  2 BYTEs   (big-endian) destination socket

--------N-7A----BX000C----------------------
INT 7A U - Novell NetWare - IPX Driver - internal - INITIALIZE NETWORK ADDRESS
        BX = 000Ch
        CX:DX = global network address (see INT 7A/BX=0002h)
        ES:DI -> "OSINCRITICALSECTION" flag
        DS:SI -> current mode for socket
Note:   the address cannot be changed once it has been initialized
SeeAlso: INT 7A/BX=0024h

--------N-7A----BX000D----------------------
INT 7A U - Novell NetWare - IPX Driver - internal - IPX GET PACKET SIZE
        BX = 000Dh
Return: AX = maximum packet size
        CX = retry count
SeeAlso: BX=001Ah

--------N-7A----BX000E----------------------
INT 7A U - Novell NetWare - IPX Driver - internal - TERMINATE SOCKETS
        BX = 000Eh
Return: nothing
Notes:  this function terminates all sockets opened with the current mode; this
          may be intended for future enhancements as the socket mode never
          changes in v2.15
        called by the NetWare shell if a program terminates

--------N-7A----BX000F----------------------
INT 7A - Novell NetWare - IPX Driver - INTERNAL - SEND PACKET
        BX = 000Fh
        ES:SI -> Event Control Block (see BX=0003h)
Note:   nearly identical to function 0003h, but does not copy address into
          the first fragment, and bypasses normal error checking
SeeAlso: BX=0003h
```

ภาคผนวก ค. – IPX.H

```
#include <dos.h>
void (far *ipxspx)(void);
REGS regs;
SREGS sregs;

struct ECBFragment
{
    void far *addr;
    unsigned int size;
};

struct NetAddr
{
    unsigned char network[4];
    unsigned char node[6];
};

struct NetInfo
{
    NetAddr addr;
    unsigned int socket;
};

struct ECB /* 60 bytes long */
{
    void far *linkadd; /* link address */
    void (far *ESRaddress)();/* default should set to null */
    unsigned char inuseflag;
    char completioncode; /* be 0 on success */
    unsigned int socket;
    char IPXworkspace[4];
    char driverworkspace[12];
    char immediateaddress[6];
    unsigned int fragmentcount;
    ECBFragment fragdesc[4];
};

struct IPXHeader /* 30 bytes long */
{
    int checksum;
    unsigned int length;
    char txcontrol; /* transport control */
    char packettype;
    NetInfo dest;
    NetInfo src;
};

int IPXInitial(void)
{
    regs.x.ax = 0x7a00;
    int86x(0x2f, &regs,&regs, &sregs);
    if (regs.h.al != 0xff) return 0;
    ipxspx = (void far(*)()) MK_FP(sregs.es, regs.x.di);
    regs.x.bx=0x0010;
    regs.h.al=0x00;
    int86(0x7a, &regs, &regs);
    if (regs.h.al == 0x00) return 1;
    return 2;
}
```

```
unsigned int IPXOpensocket(unsigned int sock)
{
    unsigned int a;
    char succ;
    regs.x.bx=0x0000;
    regs.h.al=0x00;
    regs.x.dx=sock;
    int86(0x7a, &regs, &regs);
    a=regs.x.dx;        /* Get socket number */
    succ=regs.h.al;
    if (succ==0) return a;
    else return 0;
}

void IPXSendpacket(ECB far *ecbs)
{
    regs.x.bx=0x0003;
    sregs.es=FP_SEG(ecbs);
    regs.x.si=FP_OFF(ecbs);
    int86x(0x7a, &regs, &regs, &sregs);
}

char IPXListenforpacket(ECB far *ecbr)
{
    int a;
    sregs.es=FP_SEG(ecbr);
    regs.x.si=FP_OFF(ecbr);
    regs.x.bx=0x0004;
    int86x(0x7a, &regs, &regs, &sregs);
    a=regs.h.al;
    if (a==0) return 1; else return 0;
}

char IPXCancelevent(ECB far *ecbr)
{
    char a;
    regs.x.bx=0x0006;
    sregs.es=FP_SEG(ecbr);
    regs.x.si=FP_OFF(ecbr);
    int86x(0x7a, &regs, &regs, &sregs);
    a=regs.h.al;
    return a;
}

struct NetAddr IPXGetinternetworkaddress(void)
{
    NetAddr tmp;
    regs.x.bx=0x0009;
    regs.x.si=FP_OFF((void far *) &tmp);
    sregs.es=FP_SEG((void far *) &tmp);
    int86x(0x7a, &regs, &regs, &sregs);
    return tmp;
}

void IPXClosesocket(unsigned int socket)
{
    regs.x.bx=0x0001;
    regs.x.dx=socket;
    int86(0x7a, &regs, &regs);
}
```

```
void IPXRelinquish(void)
{
    regs.x.bx=0x000a;
    int86(0x7a, &regs, &regs);
}
```

ภาคผนวก จ. – IPXCLAS4.H

```
#include <mem.h>
#include <alloc.h>
#include <time.h>
#include <string.h>
#include "misc.h"
#include "ipx.h"
#define MAXRECVECB 20
#define MAXSENDECB 10
#define MAXDATASIZE 1024
#define RECVSOCK 0x4040
#define SIGSOCK 0x4242
#define SENDSOCK 0x4343
#define SIZEOFPACK 256
#define MAXCX 20    /* concurrent connection : unusable */
#define TIMEOUT 2 /* timeout in second */
#define SIGTIMEOUT 5 /* signalling timeout */
#define USERNAMELONG 10

typedef struct Datapack
{
    IPXHeader header;
    unsigned long id;
    char data[SIZEOFPACK];
};

typedef struct Sigpack
{
    IPXHeader header;
    unsigned long sig;
    char localusername[USERNAMELONG];
    char remoteusername[USERNAMELONG];
};

enum signaldefine {
    NONE, REQCONN, ACCEPTCONN, CONFIRMCONN, DISCONN, DISCONNACK,
    FINDUSER, FINDUSERACK
};

enum statusdefine {
    AVAIL, WAITING, CONNECTING, WAITFORCONFIRM,
    CONNECTED, DISCONNECTING, DISCONNECTED, FINDING, FOUNDUSER,
    WAITFORUSERCONFIRM
};

enum ecbstatusdefine {
    FREE, INUSE, DATAREADY
};

NetAddr NULLADDR;

class IPX
{
    private:
        ECB ecbrecv[MAXRECVECB];
        int ecbrecvstat[MAXRECVECB];
        Datapack ecbrecvbuffer[MAXRECVECB];

        ECB ecbsend[MAXSENDECB];
        int ecbsendstat[MAXSENDECB];
        Datapack ecbsendbuffer[MAXSENDECB];
```

```
        time_t sendtime[MAXSENDECB];

        ECB ecbrecvsig; /* signal receiving ECB */
        Sigpack ecbrecvsigbuffer;
        ECB ecbsendsig;
        Sigpack ecbsendsigbuffer;
        NetAddr addrlist[MAXCX+1];
        int netstat[MAXCX]; /* status of connection */
        Sigpack sigbuffer; /* signal buffer */
        NetAddr sigaddr; /* address of signal source */
        int con; /* use for count number of connection */
        int countecbr; /* use for count ECBs for receiving */
        int countecbs; /* use for count ECBs for sending */
        int recvsocket; /* receive socket number */
        int sigsocket; /* receive signal socket number */
        NetAddr localaddr; /* local internetwork address */
        void requestconnect(int cxno);
        void sendblock(int cxno, int ecbno, char far *buffer, unsigned
size);
        void makewait(int i);
        void makewaitsig();
        void rewait(int i);
        int seekdata(int cxno, unsigned long *minimum);
        int seekfreeslot(int ecbstat[]);
        int seekecbbypackno(int cxno, unsigned long int packno);
        void clearecbs(void);
        void resend(int i); /* resend packet */
        void checktimeout(void); /* check n resend timeout packet */
        unsigned long acktable[MAXSENDECB];
        int acktablesize;
        void addtotable(unsigned long ackno);
        int delfromtable(unsigned long ackno);
        void sorttable(void);
        void cleartable(void);
        NetAddr useraddr, remoteaddr;
        unsigned long lastsendno[MAXCX]; /* last seqno that sent */
        unsigned long lastackno[MAXCX]; /* last seqno that acked
                                   correctly (from remote) */
        unsigned long lastgetno[MAXCX];
            /* last packet's seqno that got by getdata */

    public:
        IPX();
        void endIPX();
        void waiting();
        int connect(NetAddr dest);
        void finduser(const char *name);
        void changeusername(const char *name);
        int accept(NetAddr dest);
        NetAddr checkincomingreq();
        int disconnect(int cxno);
        int senddata(int cxno, char far *buffer, unsigned int size);
        int abletosend(int cxno);
        int getdata(int cxno, char far *buffer);
        int checkdata(int cxno);
        void alwayscall();
        int getstat(int cxno);
        NetAddr getuseraddr();
        int sendsignal(int cxno, int sig);
        NetAddr getremoteaddr();
};
```

```
IPX::IPX() /* Constructor */
{
    int i;
    acktablesize=0;
    memset(&acktable,0xFF,MAXSENDECB*sizeof(int));
    memset(&addrlist[MAXCX],0xFF,sizeof(NetAddr));
    for(i=0;i<MAXCX;i++)
    {
        lastsendno[i]=0; lastackno[i]=0;
        lastgetno[i]=0; netstat[i]=AVAIL;
    }
    con=0; countecbr=0; countecbs=0;
    if (IPXInitial()==0) myexit(INITERR);
    recvsocket=IPXOpensocket(RECVSOCK);
    sigsocket=IPXOpensocket(SIGSOCK);
    IPXOpensocket(SENDSOCK);
    if ((recvsocket==0)||(sigsocket==0)) myexit(SOCKERR);
    localaddr=IPXGetinternetworkaddress();

    for(i=0;i<MAXSENDECB;i++)
    {
        ecbsend[i].ESRaddress=NULL;
        ecbsend[i].linkadd=NULL;
        ecbsend[i].socket=SENDSOCK;
        ecbsend[i].fragmentcount=3;
        ecbsend[i].fragdesc[0].addr=(void far *)
&(ecbsendbuffer[i].header);
        ecbsend[i].fragdesc[0].size=sizeof(IPXHeader);
        ecbsend[i].fragdesc[1].addr=(void far *) &(ecbsendbuffer[i].id);
        ecbsend[i].fragdesc[1].size=sizeof(unsigned long);
        ecbsend[i].fragdesc[2].addr=(void far *) ecbsendbuffer[i].data;
        ecbsend[i].fragdesc[2].size=SIZEOFPACK;
        ecbsend[i].completioncode=1;
        ecbsendbuffer[i].header.src.addr=localaddr;
        ecbsendbuffer[i].header.src.socket=SENDSOCK;
        ecbsendbuffer[i].header.dest.socket=RECVSOCK;
        ecbsendbuffer[i].header.packettype=4;
        ecbsendstat[i]=FREE;
    }

    for(i=0;i<MAXRECVECB;i++)
    {
        ecbrecv[i].ESRaddress=NULL;
        ecbrecv[i].linkadd=NULL;
        ecbrecv[i].socket=RECVSOCK;
        ecbrecv[i].fragmentcount=3;
        ecbrecv[i].fragdesc[0].addr=(void far *)
&(ecbrecvbuffer[i].header);
        ecbrecv[i].fragdesc[0].size=sizeof(IPXHeader);
        ecbrecv[i].fragdesc[1].addr=(void far *) &(ecbrecvbuffer[i].id);
        ecbrecv[i].fragdesc[1].size=sizeof(unsigned long);
        ecbrecv[i].fragdesc[2].addr=(void far *) ecbrecvbuffer[i].data;
        ecbrecv[i].fragdesc[2].size=SIZEOFPACK;
        ecbrecv[i].completioncode=1;
        ecbrecvstat[i]=FREE;
    }

    ecbrecvsig.ESRaddress=NULL;
    ecbrecvsig.linkadd=NULL;
    ecbrecvsig.socket=SIGSOCK;
    ecbrecvsig.fragmentcount=4;
    ecbrecvsig.fragdesc[0].addr=(void far *) &(ecbrecvsigbuffer.header);
    ecbrecvsig.fragdesc[0].size=sizeof(IPXHeader);
    ecbrecvsig.fragdesc[1].addr=(void far *) &(ecbrecvsigbuffer.sig);
```

```
    ecbrecvsig.fragdesc[1].size=sizeof(unsigned long);
    ecbrecvsig.fragdesc[2].addr=(void far *)
ecbrecvsigbuffer.localusername;
    ecbrecvsig.fragdesc[2].size=USERNAMELONG;
    ecbrecvsig.fragdesc[3].addr=(void far *)
ecbrecvsigbuffer.remoteusername;
    ecbrecvsig.fragdesc[3].size=USERNAMELONG;
    ecbrecvsig.completioncode=1;

    ecbsendsig.inuseflag=0;
    ecbsendsig.completioncode=0;
    ecbsendsigbuffer.localusername[0]='\0';
    ecbsendsigbuffer.remoteusername[0]='\0';
    ecbrecvsig.inuseflag=0;

    makewaitsig();
    memcpy((char *) &NULLADDR,"0000000000",10);
    sigbuffer.sig=NONE;
}

void IPX::endIPX()
{
    sendsignal(0, DISCONN);
    IPXClosesocket(RECVSOCK);
    IPXClosesocket(SIGSOCK);
    IPXClosesocket(SENDSOCK);
}

NetAddr IPX::getuseraddr()
{
    return useraddr;
}

NetAddr IPX::getremoteaddr()
{
    return remoteaddr;
}

void IPX::finduser(const char *name)
{
    strcpy(ecbsendsigbuffer.remoteusername,name);
    sendsignal(MAXCX, FINDUSER);
    IPXCancelevent(&ecbrecvsig);
    IPXListenforpacket(&ecbrecvsig);
}

void IPX::changeusername(const char *name)
{
    strcpy(ecbsendsigbuffer.localusername,name);
}

int IPX::abletosend(int cxno)
{
    if ((countecbs<MAXSENDECB)&&(netstat[cxno]==CONNECTED)
    &&(lastsendno[cxno]-lastackno[cxno]<MAXRECVECB)) return 1;
    else return 0;
}

void IPX::checktimeout(void)
{
    static time_t now;
    now=time(NULL);
    for(int i=0;i<MAXSENDECB;i++)
    {
```

```
        if ((now-sendtime[i]>TIMEOUT)&&(ecbsendstat[i]==INUSE)
        &&(ecbsend[i].completioncode==0))
        {
            resend(i);
        }
    }
}

NetAddr IPX::checkincomingreq()
{
    if (sigbuffer.sig==REQCONN) return sigaddr;
    else return NULLADDR;
}

int IPX::seekfreeslot(int ecbstat[])
{
    int i=0,ret=-1;
    while(i<MAXSENDECB)
    {
        if (ecbstat[i]==FREE)
        {
            ret=i;
            i=MAXSENDECB+10;
        }
        i++;
    }
    return ret;
}

void IPX::makewait(int i)
{
    IPXCancelevent((ECB far *) &ecbrecv[i]);
    ecbrecvstat[i]=INUSE;
    IPXListenforpacket((ECB far *) &ecbrecv[i]);
    IPXRelinquish();
}

void IPX::rewait(int i)
{
    IPXCancelevent((ECB far *) &ecbrecv[i]);
    ecbrecvstat[i]=INUSE;
    IPXListenforpacket((ECB far *) &ecbrecv[i]);
    IPXRelinquish();
}

void IPX::makewaitsig()
{
    IPXCancelevent((ECB far *) &ecbrecvsig);
    IPXListenforpacket((ECB far *) &ecbrecvsig);
    IPXRelinquish();
}


int IPX::connect(NetAddr dest)
{
    if (con>=MAXCX) return 0;
    else {
        addrlist[con]=dest;
        sendsignal(con, REQCONN);
        netstat[con]=CONNECTING;
        con++;
        return 1;
    }
}
```

```
void IPX::waiting()
{
    netstat[0]=WAITING;
}

int IPX::accept(NetAddr dest)
{
    if (con>=MAXCX) return 0;
    else {
        addrlist[con]=dest;
        sendsignal(con, ACCEPTCONN);
        netstat[con]=WAITFORCONFIRM;
        con++;
        return 1;
    }
}

void IPX::alwayscall(void)
{
    int i=0;
    IPXRelinquish();
    if (sigbuffer.sig==FINDUSER)
    {
        if (strcmp(sigbuffer.remoteusername,
ecbsendsigbuffer.localusername)==0)
        {
            sendsignal(MAXCX, FINDUSERACK);
        }
        sigbuffer.sig=NONE;
    }

    if (sigbuffer.sig==FINDUSERACK)
    {
        if (strcmp(sigbuffer.localusername,
ecbsendsigbuffer.remoteusername)==0)
        {
            netstat[i]=FOUNDUSER;
            useraddr=sigaddr;
        }
        sigbuffer.sig=NONE;
    }

    switch (netstat[i]) {
    case DISCONNECTED :
    case WAITING : if (sigbuffer.sig==REQCONN)
                   {
                       netstat[i]=WAITFORUSERCONFIRM;
                       remoteaddr=sigaddr;
                   }
                   sigbuffer.sig=NONE;
                   break;

    case CONNECTING :
//                 if ((memcmp(&sigaddr,&addrlist[i],sizeof(sigaddr))==0)
                   if ((sigbuffer.sig==ACCEPTCONN)) /* if remote accept
connect */
                   {
                       sendsignal(i,CONFIRMCONN); /* confirm connect to
remote */
                       netstat[i]=CONNECTED;
                       addrlist[i]=sigaddr;
                       for(i=0;i<MAXRECVECB;i++)
                       {
```

```
                        makewait(i); /* make ecbrecv, listen for packet
*/
                    }
                }
                sigbuffer.sig=NONE;
                break;

    case WAITFORCONFIRM :
                if (sigbuffer.sig==CONFIRMCONN)
                {
                    netstat[i]=CONNECTED;
                    for(i=0;i<MAXRECVECB;i++)
                    {
                        makewait(i); /*make ecbrecv listenforpacket */
                        ecbrecvstat[i]=INUSE;
                    }
                }
                sigbuffer.sig=NONE;
                break;

    case CONNECTED :
                if ((memcmp(&sigaddr,&addrlist[i],sizeof(sigaddr))==0)
                if ((sigbuffer.sig==DISCONN))
                {
                    sendsignal(i,DISCONNACK);
                    netstat[i]=DISCONNECTED;
                    sigbuffer.sig=NONE;
                }
                break;

    case DISCONNECTING :
                if ((memcmp(&sigaddr,&addrlist[i],sizeof(sigaddr))==0)
                if ((sigbuffer.sig==DISCONNACK))
                {
                    netstat[i]=DISCONNECTED;
                    sigbuffer.sig=NONE;
                }
                break;
    }

    if (ecbrecvsig.inuseflag==0)
    {
        if (_fmemcmp( &ecbrecvsigbuffer.header.src.addr, &localaddr,
sizeof(localaddr) )==0)
            makewaitsig();
        else
        {
            if (sigbuffer.sig==NONE)
            {
                sigaddr=ecbrecvsigbuffer.header.src.addr;
                sigbuffer=ecbrecvsigbuffer;
                makewaitsig();
            }
        }
    }

    if (netstat[0]==CONNECTED)
    {
        clearecbs(); /*clear ECB use for sending that already have ack */
        checktimeout(); /* check timeout packet and resend */
    }
}
```

```
int IPX::getstat(int cxno)
{
    return netstat[cxno];
}

int IPX::disconnect(int cxno)
{
    sendsignal(cxno, DISCONN);
    for(int i=cxno; i<=con-2; i++)
        addrlist[i]=addrlist[i+1];
    con--;
    netstat[cxno]=DISCONNECTING;
    return 1;
}

int IPX::senddata(int cxno, char far *buffer, unsigned int size)
{
    if (abletosend(cxno))
    {
        int i;
        clearecbs();
        if ((i=seekfreeslot(ecbsendstat))<0) return 0;
        sendblock(cxno, i, buffer, size);
        ecbsendstat[i]=INUSE;
        sendtime[i]=time(NULL); /* keep time */
        countecbs++;
        return 1; /* sending success */
    } else return 0; /* ecb full */
}

void IPX::clearecbs() /* use to clear ecbsend that already receive
acknowledge */
{
    int sendidx;
    for(int cxno=0;cxno<con;cxno++)
    {
        for(int sendidx=0;sendidx<MAXSENDECB;sendidx++)
        {

    if((ecbsendstat[sendidx]==INUSE)&&(ecbsend[sendidx].inuseflag==0))
    /* if finished send */
            {
                if(ecbsend[sendidx].completioncode==0)
                /* if send success */
                {
                    if (ecbsendbuffer[sendidx].id>lastackno[cxno])
                        addtotable(ecbsendbuffer[sendidx].id);
                    ecbsendstat[sendidx]=FREE;
                    countecbs--;
                }
                else
                {
                    printf("resend");
                    resend(sendidx);
                }
            }
        }
        sorttable();
        while(acktablesize>0)
        {
            if(acktable[0]==lastackno[0]+1)
            {
                delfromtable(acktable[0]);
                lastackno[0]++;
```

```
            }
            else break;
        }
    }
}

void IPX::resend(int i)
{
    IPXCancelevent(&ecbsend[i]);
    ecbsend[i].completioncode=1;
    sendtime[i]=time(NULL);
    ecbsend[i].socket=SENDSOCK;
    IPXSendpacket((ECB far *)&(ecbsend[i]));
    IPXRelinquish();
}

int IPX::sendsignal(int cxno, int signo)
{
    time_t lastmoment;
    while ((volatile) ecbsendsig.inuseflag!=0); /*wait for lastsendsig */
    ecbsendsigbuffer.header.dest.addr=addrlist[cxno];
    ecbsendsigbuffer.header.dest.socket=SIGSOCK;
    ecbsendsigbuffer.header.src.addr=localaddr;
    ecbsendsigbuffer.header.packettype=4;
    ecbsendsigbuffer.sig=signo;
    ecbsendsig.socket=SENDSOCK;
    memcpy(ecbsendsig.immediateaddress,addrlist[cxno].node,6);
    ecbsendsig.ESRaddress=NULL;
    ecbsendsig.linkadd=NULL;
    ecbsendsig.fragmentcount=4;
    ecbsendsig.fragdesc[0].addr=(void far *) &(ecbsendsigbuffer.header);
    ecbsendsig.fragdesc[0].size=sizeof(IPXHeader);
    ecbsendsig.fragdesc[1].addr=(void far *) &(ecbsendsigbuffer.sig);
    ecbsendsig.fragdesc[1].size=sizeof(unsigned long);
    ecbsendsig.fragdesc[2].addr=(void far *)
ecbsendsigbuffer.localusername;
    ecbsendsig.fragdesc[2].size=USERNAMELONG;
    ecbsendsig.fragdesc[3].addr=(void far *)
ecbsendsigbuffer.remoteusername;
    ecbsendsig.fragdesc[3].size=USERNAMELONG;
    ecbsendsig.completioncode=1;
    lastmoment=time(NULL);
    while ((volatile) ecbsendsig.completioncode!=0)
    {
        IPXCancelevent(&ecbsendsig);
        IPXSendpacket(&ecbsendsig);
        while ((volatile) ecbsendsig.inuseflag!=0);
        if ((time(NULL)-lastmoment)>SIGTIMEOUT) printf("SigTimeout");
    }

    return 1;
}

int IPX::seekdata(int cxno, unsigned long *minimum)
{
    int min=-1;
    *minimum=0xFFFFFFFFL; /* set minimum to be a large value */
    for(int i=0;i<MAXRECVECB;i++)
    {
        if
((ecbrecv[i].inuseflag==0x00)&&(ecbrecv[i].completioncode==0x00))
//      &&(_fmemcmp((char far *) &(*(IPXHeader far
*)(ecbrecv[i].fragdesc[0].addr)).src.addr,(char far *)&(addrlist[cxno]),
sizeof(NetAddr))==0))
```

```
        {
            if (*minimum>*((unsigned long far
*)(ecbrecv[i].fragdesc[1].addr)))
            {
                *minimum=*((unsigned long far
*)(ecbrecv[i].fragdesc[1].addr));
                min=i;
            }
        }
    }
    return min; /* if no packet, -1 will be returned */
}

int IPX::getdata(int cxno, char far *buffer)
{
    static unsigned long l;
    static int i;
    static unsigned packlen,t;
    static IPXHeader far *hd;
    if (netstat[cxno]!=CONNECTED) return 0;
    /* get idno of packet that have lowest idno */
    /* return 0 if no packet in buffer */
    if ((i=seekdata(cxno,&l))<0) return 0;
    while ((l<=lastgetno[cxno])&&(i>=0))
    {
        rewait(i);
        i=seekdata(cxno,&l);
    }
    if (i>MAXRECVECB) myexit(PACKETSOBIG);
    if(l==lastgetno[cxno]+1)
    {
        hd=(IPXHeader far *)(ecbrecv[i].fragdesc[0].addr);
        t=hd->length;
        swab((char *)&t, (char *)&packlen, 2);
        _fmemcpy(buffer, ecbrecv[i].fragdesc[2].addr, packlen-
sizeof(IPXHeader)-sizeof(long));
        rewait(i);
        lastgetno[cxno]++;
        return i;
    } else return 0; /* in case packet loss and waiting for new one */
}


void IPX::sendblock(int cxno, int ecbno, char far *buffer, unsigned size)
{
    if (size>SIZEOFPACK) myexit(PACKETSOBIG);
    IPXCancelevent(&ecbsend[ecbno]);
    _fmemcpy(ecbsendbuffer[ecbno].data,buffer,size);
    ecbsendbuffer[ecbno].id=lastsendno[cxno]+1; /* set id no */
    lastsendno[cxno]++;
    ecbsendbuffer[ecbno].header.dest.addr=addrlist[cxno]; /*set target*/
    _fmemcpy(ecbsend[ecbno].immediateaddress,addrlist[cxno].node,6);
    ecbsend[ecbno].completioncode=1;
    ecbsend[ecbno].fragdesc[2].size=size;
    IPXSendpacket((ECB far *) &ecbsend[ecbno]);
    if (ecbsendbuffer[ecbno].id>lastsendno[cxno]) printf("Memory
Error!");
}
void IPX::cleartable()
{
    for(int i=0;i<MAXSENDECB;i++) acktable[i]=0;
}

void IPX::addtotable(unsigned long ackno)
```

```
{
    acktable[acktablesize]=ackno;
    acktablesize++;
}

int IPX::delfromtable(unsigned long ackno)
{
    int i;
    for(i=0;i<acktablesize;i++)
    {
        if (acktable[i]==ackno) break;
    }
    if (i>=acktablesize) return 0;
    if (i==acktablesize-1) acktable[i]=0;
    else
    for(;i<acktablesize;i++) acktable[i]=acktable[i+1];
    acktablesize--;
    if (acktablesize<0) printf("table error");
    return 1;
}

void IPX::sorttable()
{
    int i,j;
    unsigned long tmp;
    for(i=0;i<acktablesize;i++)
    {
        for(j=i;j<acktablesize;j++)
        {
            if(acktable[j]<acktable[i])
            {
                tmp=acktable[i];
                acktable[i]=acktable[j];
                acktable[j]=tmp;
            }
        }
    }
}
```

ภาคผนวก จ. – MOUSE4.H

```
#include <dos.h>
#define EV 100 /* Amount of event buffer */
#define BUTTON 2 /* number of button of mouse */
enum axis { X = 0,  Y, };
enum eventtype {
    NoEvent = 0,
    LeftDown, RightDown, MidDown,
    LeftRel, RightRel, MidRel, Keyboard
};
enum intrmask {
    Movement = 1,
    LeftPress= 2, LeftRelease = 4,
    RightPress = 8, RightRelease = 16,
};

struct eventrecord
{
    int type;
    int x,y;
};

union posunion
{
    long int xy;
    int pos[2];
};

class eventclass
{
    private :
        static struct eventrecord list[EV]; /* event list */
        static int currentevent;
        static int latestevent;

    public :
        eventclass();
        static struct eventrecord getevent(void);
        static void delevent(void);
        static void clearallevent(void);
        static void createevent(eventrecord e);
};

/* NEED TO define static variable in eventclass */

int eventclass::currentevent;
int eventclass::latestevent;
struct eventrecord eventclass::list[EV];

eventclass::eventclass()
{
    latestevent=0;
    clearallevent();
}

struct eventrecord eventclass::getevent(void)
{
    return list[currentevent];
}

void eventclass::delevent(void)
```

```
{
/*  static int a;
    a=currentevent;*/
    if (currentevent!=latestevent)
    {
        currentevent++;
        if (currentevent>EV) currentevent=0;
    }
    else list[currentevent].type=NoEvent;
}

void eventclass::clearallevent(void)
{
    currentevent=latestevent+1;
    if (currentevent>EV) currentevent=0;
    list[currentevent].type=NoEvent;
}

void eventclass::createevent(eventrecord e)
{
    int temp;
    temp=latestevent+1;
    if (temp>EV) temp=0;
    list[temp].type=e.type;
    list[temp].x=e.x;
    list[temp].y=e.y;
    latestevent=temp;
}

class mouse
{
    private :
        int reset(void);
        unsigned int x;
        unsigned int y;
        int xpress, ypress, xrel, yrel;

    public :
        mouse();
        ~mouse();
        void setvscreen(unsigned int, unsigned int);
        void show(void);
        void hide(void);
        void sethandler(void interrupt far (*handler)());
        void refreshxy(void);
        int xpos(void);
        int ypos(void);
        unsigned int getpress(int button);
        unsigned int getrelease(int button);
        void alwayscall(void);
        eventclass event;
};

mouse::mouse()
{
    reset();
    alwayscall();
}

mouse::~mouse()
{
    _AX=0;
    geninterrupt(0x33);
    _CX=0x0000;
```

```
    _AX=0x0C;
    geninterrupt(0x33);
}

int mouse::reset()
{
    unsigned int rvalue;
    asm {
        mov ax, 0x00;
        int 0x33;
        mov rvalue, ax;
    }
    return rvalue;
}

void mouse::sethandler(void interrupt far (*handler)())
{
    _CX=0x00FE; /* all event except mouse movement */
    _AX=0x0C;
    _ES=FP_SEG(handler);
    _DX=FP_OFF(handler);
    geninterrupt(0x33);
}


void mouse::setvscreen(unsigned int vx, unsigned int vy)
{
    asm {
        mov ax, 0x07;
        mov cx, 0x00;
        mov dx, vx;
        int 0x33;
    }

    asm {
        mov ax, 0x08;
        mov cx, 0x00;
        mov dx, vy;
        int 0x33
    }
}

void mouse::show(void)
{
    asm {
        mov ax,0x01;
        int 0x33
    }
}

void mouse::hide(void)
{
    asm {
        mov ax,0x02;
        int 0x33;
    }
}

void mouse::refreshxy(void)
{
    _AX=0x03;
    geninterrupt(0x33);
    x=_CX;
    y=_DX;
```

```
}

int mouse::xpos(void)
{
    return x;
}

int mouse::ypos(void)
{
    return y;
}

unsigned int mouse::getpress(int button)
{
    static int ret;
    _BX=button;
    _AX=5;
    geninterrupt(0x33);
    ret=_BX;
    xpress=_CX;
    ypress=_DX;
    return ret;
}

unsigned int mouse::getrelease(int button)
{
    static int ret;
    _BX=button;
    _AX=6;
    geninterrupt(0x33);
    ret=_BX;
    xrel=_CX;
    yrel=_DX;
    return ret;
}


void mouse::alwayscall(void)
{
    static unsigned int i, gp, gr;
    static eventrecord e;
    for(i=0;i<=2;i++)
    {
        gp=getpress(i); gr=getrelease(i);
        if (gp>gr)
        {
            e.type=LeftDown+i;
            e.x=xpress; e.y=ypress;
            event.createevent(e);
        }
        if (gr>gp)
        {
            e.type=LeftRel+i;
            e.x=xrel; e.y=yrel;
            event.createevent(e);
        }

        i++;
    }
}
```

ภาคผนวก ฉ. – โปรแกรมหลัก

```
#include<conio.h>
#include<bios.h>
#include<stdio.h>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>
#include<mem.h>
#include<string.h>
#include<alloc.h>
#include "mouse4.h"
#include "bitmap.h"
#include "ipxclas4.h"
#define LX 100
#define MX 500
#define LY 0
#define MY 400
#define FHcolor 15
#define FREE_X1 0
#define FREE_Y1 0
#define FREE_X2 29
#define FREE_Y2 29
#define LINE_X1 30
#define LINE_Y1 0
#define LINE_X2 59
#define LINE_Y2 29
#define RECT_X1 0
#define RECT_Y1 30
#define RECT_X2 29
#define RECT_Y2 59
#define CIRC_X1 30
#define CIRC_Y1 30
#define CIRC_X2 59
#define CIRC_Y2 59
#define TEXT_X1 0
#define TEXT_Y1 60
#define TEXT_X2 29
#define TEXT_Y2 89
#define STAT_X1 0
#define STAT_Y1 470
#define STAT_X2 100
#define STAT_Y2 479
#define LEFTBOARD 120
#define BUT_X1 0
#define BUT_Y1 150
#define MENUBGCOLOR 0
#define QLONG 100
#define MAXTEXTLONG 40
enum elementID {
    ClearScr=-10, HeadID=-1, NoButton, Freehand, Line, Rect, Circle, Text
};

struct drawinfo
{
    int ID; /* use to identify shape */
    int x1,y1,x2,y2; /* y2 = ratio1, x2 = ratio2 */
    int radius;
    int fillcol; /* FillColor, -1 if not fill */
    int col; /* draw color */
    int thickness;
    char text[MAXTEXTLONG];
};
```

```
static mouse m;
int currentbutton=NoButton;
int firstflag=0;
int thick=2;
int inboard=1;
int Color=0; /* Current Color */
int Fillflag=1; /* Fill or not? */
int FillColor=11;
int BGColor=0;
char inputc=0;
drawinfo tosend;

inline void blackbox(int x1, int y1, int x2, int y2)
{
    setfillstyle(SOLID_FILL,BGColor);
    bar(x1,y1,x2,y2);
}

void drawcircle(int cx,int cy, int r, int ratio1, int ratio2, int wmode)
// 'r' refer to radius IN X AXIS //
// ratio1 and ratio2 will be used to plot in y axis... by y*ratio1/ratio2
//
// if wmode=1 will XOR with old pixel //
{
    int curx,cury,tmp,dx=0,dy=0,c;
    if (ratio2==0) ratio2=1;
    if (ratio1<ratio2)
    {
        for(curx=cx-((long)r*14143L/20000L);curx<=cx;curx++)
        {
            dx=cx-curx;
            while((long)dy*(long)dy+(long)dx*(long)dx < (long)r*(long)r)
dy++;
            dy--; // makes x^2 + y^2 always less than r //
            tmp=dy;
            dy=(long)dy*(long)ratio1/(long)ratio2;
            if (wmode==0) {
                putpixel(cx-dx,cy-dy,Color);
                putpixel(cx+dx,cy-dy,Color);
                putpixel(cx-dx,cy+dy,Color);
                putpixel(cx+dx,cy+dy,Color);
            }
            else {
                putpixel(cx-dx,cy-dy,Color^getpixel(cx-dx,cy-dy));
                putpixel(cx+dx,cy-dy,Color^getpixel(cx+dx,cy-dy));
                putpixel(cx-dx,cy+dy,Color^getpixel(cx-dx,cy+dy));
                putpixel(cx+dx,cy+dy,Color^getpixel(cx+dx,cy+dy));
            }
            dy=tmp; tmp=dx; dx=(long)dx*(long)ratio1/(long)ratio2;

        if (wmode==0) {
                putpixel(cx-dy,cy-dx,Color);
                putpixel(cx+dy,cy-dx,Color);
                putpixel(cx-dy,cy+dx,Color);
                putpixel(cx+dy,cy+dx,Color);
            }
            else {
                putpixel(cx-dy,cy-dx,Color^getpixel(cx-dy,cy-dx));
                putpixel(cx+dy,cy-dx,Color^getpixel(cx+dy,cy-dx));
                putpixel(cx-dy,cy+dx,Color^getpixel(cx-dy,cy+dx));
                putpixel(cx+dy,cy+dx,Color^getpixel(cx+dy,cy+dx));
            }
            dx=tmp;
```

```
            }
        }
        else
        {
            for(cury=cy-((long)r*14143L/20000L);cury<=cy;cury++)
            {
                dy=cy-cury;
                while((long)dx*(long)dx+(long)dy*(long)dy < (long)r*(long)r)
    dx++;
                dx--; // makes x^2 + y^2 always less than r //
                tmp=dy;
                dy=(long)dy*(long)ratio1/(long)ratio2;
                if (wmode==0) {
                    putpixel(cx-dx,cy-dy,Color);
                    putpixel(cx+dx,cy-dy,Color);
                    putpixel(cx-dx,cy+dy,Color);
                    putpixel(cx+dx,cy+dy,Color);
                }
                else {
                    putpixel(cx-dx,cy-dy,Color^getpixel(cx-dx,cy-dy));
                    putpixel(cx+dx,cy-dy,Color^getpixel(cx+dx,cy-dy));
                    putpixel(cx-dx,cy+dy,Color^getpixel(cx-dx,cy+dy));
                    putpixel(cx+dx,cy+dy,Color^getpixel(cx+dx,cy+dy));
                }
                dy=tmp; tmp=dx; dx=(long)dx*(long)ratio1/(long)ratio2;

            if (wmode==0) {
                    putpixel(cx-dy,cy-dx,Color);
                    putpixel(cx+dy,cy-dx,Color);
                    putpixel(cx-dy,cy+dx,Color);
                    putpixel(cx+dy,cy+dx,Color);
                }
                else {
                    putpixel(cx-dy,cy-dx,Color^getpixel(cx-dy,cy-dx));
                    putpixel(cx+dy,cy-dx,Color^getpixel(cx+dy,cy-dx));
                    putpixel(cx-dy,cy+dx,Color^getpixel(cx-dy,cy+dx));
                    putpixel(cx+dy,cy+dx,Color^getpixel(cx+dy,cy+dx));
                }
                dx=tmp;
            }
        }
    }

    void drawfillcircle(int cx,int cy, int r, int ratio1, int ratio2)
    // 'r' refer to radius IN X AXIS //
    // ratio1 and ratio2 will be used to plot ... by ratio1/ratio2 //
    {
        int curx,cury,dx=0,dy=0,tmp,c;
        if (ratio2==0) ratio2=1;
        setcolor(FillColor);
        for(curx=cx-((long)r*14143/20000);curx<=cx;curx++)
        {
            dx=cx-curx;
            while(((long)dy*(long)dy)+((long)dx*(long)dx) < (long)r*(long)r)
    dy++;
            dy--; // makes x^2 + y^2 always less than r //
            tmp=dy;
            dy=(long)dy*(long)ratio1/(long)ratio2;
            line(cx-dx,cy-dy,cx-dx,cy+dy);
            line(cx+dx,cy-dy,cx+dx,cy+dy);
            dy=tmp; tmp=dx; dx=(long)dx*(long)ratio1/(long)ratio2;
            line(cx-dy,cy-dx,cx-dy,cy+dx);
            line(cx+dy,cy-dx,cx+dy,cy+dx);
            dx=tmp;
```

```
    }
}

void thickcircle(int cx, int cy, int r, int ratio1, int ratio2)
{
    int curx,cury,dx=0,dy=0,tmp,Filltmp,i;
    if (ratio2==0) ratio2=1;
    Filltmp=FillColor;
    FillColor=Color;
    if(ratio1<ratio2)
    {
        for(curx=cx-((long)r*14143/20000);curx<=cx;curx++)
        {
            dx=cx-curx;
            while(((long)dy*(long)dy)+((long)dx*(long)dx) <
(long)r*(long)r) dy++;
            dy--; // makes x^2 + y^2 always less than r //
            tmp=dy;
            dy=(long)dy*(long)ratio1/(long)ratio2;
            drawfillcircle(cx-dx,cy-dy,thick,1,1);
            drawfillcircle(cx+dx,cy-dy,thick,1,1);
            drawfillcircle(cx-dx,cy+dy,thick,1,1);
            drawfillcircle(cx+dx,cy+dy,thick,1,1);
            dy=tmp; tmp=dx; dx=(long)dx*ratio1/ratio2;
            drawfillcircle(cx-dy,cy-dx,thick,1,1);
            drawfillcircle(cx+dy,cy-dx,thick,1,1);
            drawfillcircle(cx-dy,cy+dx,thick,1,1);
            drawfillcircle(cx+dy,cy+dx,thick,1,1);
            dx=tmp;
        }
    }
    else
    {
        for(cury=cy-((long)r*14143/20000);cury<=cy;cury++)
        {
            dy=cy-cury;
            while(((long)dy*(long)dy)+((long)dx*(long)dx) <
(long)r*(long)r) dx++;
            dx--; // makes x^2 + y^2 always less than r //
            tmp=dy;
            dy=(long)dy*(long)ratio1/(long)ratio2;
            drawfillcircle(cx-dx,cy-dy,thick,1,1);
            drawfillcircle(cx+dx,cy-dy,thick,1,1);
            drawfillcircle(cx-dx,cy+dy,thick,1,1);
            drawfillcircle(cx+dx,cy+dy,thick,1,1);
            dy=tmp; tmp=dx; dx=(long)dx*ratio1/ratio2;
            drawfillcircle(cx-dy,cy-dx,thick,1,1);
            drawfillcircle(cx+dy,cy-dx,thick,1,1);
            drawfillcircle(cx-dy,cy+dx,thick,1,1);
            drawfillcircle(cx+dy,cy+dx,thick,1,1);
            dx=tmp;
        }
    }

    FillColor=Filltmp;
}

class queue
{
    /* tail always point at space, if tail=head means queue full */
    /* if queue empty head will be -1 */

    private:
        int head,tail;
```

```
        drawinfo drw[QLONG];
        void dechead() { if (head>0) head--; else head=QLONG-1; }
        void dectail() {  if (tail>0) tail--; else tail=QLONG-1; }
        void inchead() { if (head<QLONG-1) head++; else head=0; }
        void inctail() { if (tail<QLONG-1) tail++; else tail=0; }

    public:
        queue();
        int put(drawinfo d);
        int get(drawinfo *d);
        void clear(void);
};

queue::queue()
{
    head=-1; tail=0;
}

int queue::put(drawinfo d)
{
    if (tail!=head)
    {
        drw[tail]=d;
        if (head<0) head=tail;
        inctail();
        return 1;
    }
    else {
        dectail();
        return 0;
    }
}

int queue::get(drawinfo *d)
{
    if (head>=0)
    {
        *d=drw[head];
        inchead();
        if (head==tail) head=-1;
        return 1;
    }
    else return 0;
}

void queue::clear()
{
    queue();
}

queue sendq;

inline void fillstatus(void)
{
    tosend.thickness=thick;
    if (Fillflag==1) tosend.fillcol=FillColor;
    else tosend.fillcol=-1;
    tosend.col=Color;
}

class element
{
    public:
        int x1,y1,x2,y2;
```

```
        int ID;
        virtual void serve(void) {} ; /* called when element is clicked
*/
        virtual void redraw(void) {} ; /* use to redraw element */
        virtual void drawing(eventrecord e, int x, int y) {}; /* use to
drawing in board */
        virtual ~element() {};
        virtual void drawcomplete(drawinfo *) {};
        element *next;
        void setxy(int ax1, int ay1, int ax2, int ay2);
        bitmap buttonbitmap;
};

void element::setxy(int ax1, int ay1, int ax2, int ay2)
{
    x1=ax1; x2=ax2; y1=ay1; y2=ay2;
}

class elementlist
{
    private:
        element *headele, *tailele;
        element *temp;
        int l;

    public:
        elementlist();
        ~elementlist();
        void *add(element *e);
        void del(int ID);
        void drawall(void);
        element* getelement(int x, int y);
        element* getbyID(int id);
};

elementlist::elementlist()
{
    headele=new element;
    headele->setxy(0,0,0,0);
    headele->ID=HeadID;
    headele->next=NULL;
    tailele=headele;
    l=0;
}

elementlist::~elementlist()
{
    struct element *last;
    temp=headele;
    while(temp!=NULL)
    {
        last=temp;
        temp=temp->next;
        free(last);
    }
}

void* elementlist::add(element *e)
{
    tailele->next=e;
    tailele=e;
    e->next=NULL;
    l++;
    return e;
```

```
}

void elementlist::del(int ID)
{
    static element *last;
    temp=headele->next;
    last=headele;
    while (temp!=NULL)
    {
        if (temp->ID == ID)
        {
            last->next = temp->next;
            if (temp==tailele) tailele=last;
            delete temp;
            break;
        }
    }
}

void elementlist::drawall(void)
{
    temp=headele->next;
    while(temp!=NULL)
    {
        temp->redraw();
        temp=temp->next;
    }
}

element* elementlist::getelement(int x, int y)
{
    temp=headele->next;
    while(temp!=NULL)
    {
        if ((x>=temp->x1)&&(x<=temp->x2)&&(y>=temp->y1)&&(y<=temp->y2))
        {
            return temp;
        }
        temp=temp->next;
    }
    return headele;
}

element* elementlist::getbyID(int id)
{
    temp=headele->next;
    while(temp!=NULL)
    {
        if (temp->ID==id) return temp;
        temp=temp->next;
    }
    return headele;
}

void thickline(int x1, int y1, int x2, int y2)
{
    static int i;
    for(i=-thick;i<=thick;i++)
    {
        line(x1+thick, y1+i, x2+thick, y2+i);
        line(x1+i, y1+thick, x2+i, y2+thick);
        line(x1-thick, y1+i, x2-thick, y2+i);
        line(x1+i, y1-thick, x2+i, y2-thick);
    }
```

```
}

void thickbox(int x1, int y1, int x2, int y2)
{
    int i,tmp;
    if((x1>x2)&&(y1>y2))
    {
        tmp=x1;x1=x2;x2=tmp;
        tmp=y1;y1=y2;y2=tmp;
    }
    for(i=-thick;i<=thick;i++)
    {
        line(x1+i,y1-thick, x1+i, y2+thick);
        line(x1-thick, y2+i, x2+thick, y2+i);
        line(x2+i, y2+thick, x2+i, y1-thick);
        line(x2+thick, y1+i, x1-thick, y1+i);
    }
}


/*  thickline(x1,y1,x2,y1);
    thickline(x2,y1,x2,y2);
    thickline(x2,y2,x1,y2);
    thickline(x1,y2,x1,y1);*/
class freehand : public element
{
    public:
        freehand();
        ~freehand();
        void serve(void);
        void redraw(void);
        void drawing(eventrecord e, int x, int y);
        void drawcomplete(drawinfo *drw);
};

void freehand::drawcomplete(drawinfo *drw)
{
   int lastthick;
   setcolor(drw->col);
   lastthick=thick;
   thick=drw->thickness;
   thickline(drw->x1, drw->y1, drw->x2, drw->y2);
   thick=lastthick;
}

freehand::freehand()
{
    setxy(FREE_X1, FREE_Y1, FREE_X2, FREE_Y2);
    buttonbitmap=readbitmap("bfh.bmp");
    ID=Freehand;
}

freehand::~freehand()
{
    delbitmap(buttonbitmap);
}

void freehand::serve(void)
{
    currentbutton=Freehand;
    firstflag=1;
}

void freehand::redraw(void)
```

```
{
    blackbox(x1,y1,x2,y2);
    putbitmap(x1,y1,buttonbitmap);
}

void freehand::drawing(eventrecord e, int x, int y)
{
    static int pendown;
    static int lastx, lasty;
    if ((e.type==LeftDown)&&(inboard==1))
    {
        setcolor(Color);
        setfillstyle(SOLID_FILL,Color);
        pendown=1;
        m.hide();
        bar(e.x-thick, e.y-thick, e.x+thick, e.y+thick);
        m.show();
        lastx=e.x;
        lasty=e.y;
    }
    if ((e.type==LeftRel)&&(pendown==1))
    {
        pendown=0;
    }
    if (pendown==1)
    {
        setcolor(Color);
        if (lastx!=x||lasty!=y)
        {
            m.hide();
            thickline(lastx, lasty, x, y);
            m.show();
            tosend.ID=Freehand;
            tosend.x1=lastx; tosend.y1=lasty;
            tosend.x2=x; tosend.y2=y;
            fillstatus();
            sendq.put(tosend);
            lastx=x;
            lasty=y;
        }
    }
}

class myline : public element
{
    public:
        myline();
        ~myline();
        void serve(void);
        void redraw(void);
        void drawing(eventrecord e, int x, int y);
        void drawcomplete(drawinfo *drw);
};

myline::myline()
{
    setxy(LINE_X1, LINE_Y1, LINE_X2, LINE_Y2);
    buttonbitmap=readbitmap("bline.bmp");
    ID=Line;
}

myline::~myline()
{
    delbitmap(buttonbitmap);
```

```
}

void myline::serve(void)
{
    currentbutton=Line;
    firstflag=1;
}

void myline::redraw(void)
{
    blackbox(x1,y1,x2,y2);
    putbitmap(x1,y1,buttonbitmap);
}

void myline::drawing(eventrecord e, int x, int y)
{
    static int pendown;
    static int firstx, firsty, lastx, lasty, lastcolor;
    if ((e.type==LeftDown)&&(inboard==1))
    {
        pendown=1;
        lastx=e.x;
        lasty=e.y;
        firstx=lastx;
        firsty=lasty;
        lastcolor=0;
    }
    if ((e.type==LeftRel)&&(pendown==1))
    {
        m.hide();
        pendown=0;
        setwritemode(XOR_PUT);
        if (lastcolor!=0) setcolor(lastcolor); else setcolor(1);
        thickline(firstx, firsty, lastx, lasty);
        setwritemode(COPY_PUT);
        setcolor(Color);
        thickline(firstx, firsty, x, y);
        setfillstyle(SOLID_FILL,Color);
        bar(firstx-thick, firsty-thick, firstx+thick, firsty+thick);
        bar(x-thick, y-thick, x+thick, y+thick);
        m.show();
        tosend.ID=Line;
        tosend.x1=firstx; tosend.y1=firsty;
        tosend.x2=x; tosend.y2=y;
        fillstatus();
        sendq.put(tosend);
    }
    if (pendown==1)
    {
        if ((lastx!=x)||(lasty!=y))
        {
            m.hide();
            setwritemode(XOR_PUT);
            if (lastcolor!=0) setcolor(lastcolor); else setcolor(1);
            thickline(firstx,firsty,lastx,lasty);
            if (Color!=0) setcolor(Color); else setcolor(1);
            thickline(firstx,firsty,x,y);
            setwritemode(COPY_PUT);
            m.show();
            lastx=x;
            lasty=y;
            lastcolor=Color;
        }
    }
```

```
}

void myline::drawcomplete(drawinfo *drw)
{
    int lastthick;
    setcolor(drw->col);
    setfillstyle(SOLID_FILL,drw->col);
    lastthick=thick;
    thick=drw->thickness;
    thickline(drw->x1, drw->y1, drw->x2, drw->y2);
    bar(drw->x1-thick, drw->y1-thick, drw->x1+thick, drw->y1+thick);
    bar(drw->x2-thick, drw->y2, drw->x2+thick, drw->y2+thick);
    thick=lastthick;
}


class rect : public element
{
    public:
        rect();
        ~rect();
        void serve(void); /* virtual */
        void redraw(void); /* virtual */
        void drawing(eventrecord e, int x, int y); /* virtual */
        void drawcomplete(drawinfo *drw);
};

rect::rect()
{
    setxy(RECT_X1, RECT_Y1, RECT_X2, RECT_Y2);
    buttonbitmap=readbitmap("brect.bmp");
    ID=Rect;
}

rect::~rect()
{
    delbitmap(buttonbitmap);
}

void rect::redraw(void)
{
    blackbox(x1,y1,x2,y2);
    putbitmap(x1,y1,buttonbitmap);
}

void rect::serve(void)
{
    currentbutton=Rect;
    firstflag=1;
}

void rect::drawing(eventrecord e, int x, int y)
{
    static int pendown;
    static int firstx, firsty, lastx, lasty, lastcolor;
    if ((e.type==LeftDown)&&(inboard==1))
    {
        pendown=1;
        lastx=e.x;
        lasty=e.y;
        firstx=lastx;
        firsty=lasty;
        lastcolor=Color;
    }
```

```
        if ((e.type==LeftRel)&&(pendown==1))
        {
            m.hide();
            pendown=0;
            setwritemode(XOR_PUT);
            if (lastcolor!=0) setcolor(lastcolor); else setcolor(1);
            rectangle(firstx, firsty, lastx, lasty);
            setwritemode(COPY_PUT);
            if (e.x<LEFTBOARD) e.x=LEFTBOARD;
            if(Fillflag==1)
            {
                setfillstyle(SOLID_FILL,FillColor);
                bar(firstx,firsty,e.x,e.y);
            }
            setcolor(Color);
            thickbox(firstx, firsty, e.x, e.y);
            m.show();
            tosend.ID=ID;
            tosend.x1=firstx; tosend.y1=firsty;
            tosend.x2=x; tosend.y2=y;
            fillstatus();
            sendq.put(tosend);
        }
        if (pendown==1)
        {
            if ((lastx!=x)||(lasty!=y))
            {
                m.hide();
                setwritemode(XOR_PUT);
                if (lastcolor!=0) setcolor(lastcolor); else setcolor(1);
                rectangle(firstx,firsty,lastx,lasty);
                if (Color!=0) setcolor(Color); else setcolor(1);
                rectangle(firstx,firsty,x,y);
                setwritemode(COPY_PUT);
                m.show();
                lastx=x;
                lasty=y;
                lastcolor=Color;
            }
        }
    }

    void rect::drawcomplete(drawinfo *drw)
    {
        int lastthick;
        lastthick=thick;
        thick=drw->thickness;
        if(drw->fillcol>0)
        {
            blackbox(drw->x1, drw->y1, drw->x2, drw->y2);
            setfillstyle(SOLID_FILL, drw->fillcol);
            bar(drw->x1, drw->y1, drw->x2, drw->y2);
        }
        setcolor(drw->col);
        thickbox(drw->x1, drw->y1, drw->x2, drw->y2);
        thick=lastthick;
    }

    class mycircle : public element
    {
        public:
            mycircle();
            ~mycircle();
            void serve(void);
```

```
        void redraw(void);
        void drawing(eventrecord e, int x, int y);
        void drawcomplete(drawinfo *drw);
};

mycircle::mycircle()
{
    setxy(CIRC_X1, CIRC_Y1, CIRC_X2, CIRC_Y2);
    buttonbitmap=readbitmap("bcircle.bmp");
    ID=Circle;
}

mycircle::~mycircle()
{
    delbitmap(buttonbitmap);
}

void mycircle::serve(void)
{
    currentbutton=Circle;
    firstflag=1;
}

void mycircle::redraw(void)
{
    blackbox(x1,y1,x2,y2);
    putbitmap(x1,y1,buttonbitmap);
}

void mycircle::drawing(eventrecord e, int x, int y)
{
    static int pendown;
    int r,dx,dy;
    static int firstx, firsty, lastx, lasty;
    if ((e.type==LeftDown)&&(inboard==1))
    {
        pendown=1;
        lastx=e.x;
        lasty=e.y;
        firstx=lastx;
        firsty=lasty;
    }
    if ((e.type==LeftRel)&&(pendown==1))
    {
        m.hide();
        pendown=0;
        setwritemode(XOR_PUT);
        dx=lastx-firstx; dy=lasty-firsty;
        r=dx/2; if (r<0) r=-r;
        drawcircle(firstx+dx/2, firsty+dy/2, r,dy,dx,1);
        setwritemode(COPY_PUT);
        dx=x-firstx; dy=y-firsty;
        r=dx/2; if (r<0) r=-r;
        if(Fillflag==1) drawfillcircle(firstx+dx/2, firsty+dy/2,r,dy,dx);
        thickcircle(firstx+dx/2, firsty+dy/2,r,dy,dx);
        m.show();
        tosend.ID=ID;
        tosend.x1=firstx+dx/2; tosend.y1=firsty+dy/2;
        tosend.x2=dx; tosend.y2=dy; tosend.radius=r;
        fillstatus();
        sendq.put(tosend);
    }
    if (pendown==1)
    {
```

```
            if ((lastx!=x)||(lasty!=y))
            {
                m.hide();
                dx=lastx-firstx; dy=lasty-firsty;
                r=dx/2; if (r<0) r=-r;
                if (Color==0) Color=19;
                drawcircle(firstx+dx/2,firsty+dy/2,r,dy,dx,1);
                dx=x-firstx; dy=y-firsty;
                r=dx/2; if (r<0) r=-r;
                drawcircle(firstx+dx/2,firsty+dy/2,r,dy,dx,1);
                if (Color==19) Color=0;
                m.show();
                lastx=x;
                lasty=y;
            }
        }
    }

    void mycircle::drawcomplete(drawinfo *drw)
    {
        int r, filltmp, coltmp;
        filltmp=FillColor; coltmp=Color;
        FillColor=drw->fillcol; Color=drw->col;
        if (drw->fillcol>0)
            drawfillcircle(drw->x1, drw->y1 ,drw->radius,drw->y2,drw->x2);

        thickcircle(drw->x1, drw->y1,drw->radius,drw->y2,drw->x2);
        FillColor=filltmp; Color=coltmp;
    }

    class text : public element
    {
        private:
            int pendown;
            char mesg[MAXTEXTLONG];
            int curx, cury;
            void *bgimage;
            void finishtext(void);
            void beginnewtext(int x, int y);

        public:
            text();
            ~text();
            void serve(void);
            void redraw(void);
            void drawing(eventrecord e, int x, int y);
            void drawcomplete(drawinfo *drw);
    };

    text::text()
    {
        setxy(TEXT_X1,TEXT_Y1,TEXT_X2,TEXT_Y2);
        buttonbitmap=readbitmap("btext.bmp");
        ID=Text;
    }

    void text::finishtext(void)
    {
        setcolor(Color);
        m.hide();
        if (bgimage!=NULL) putimage(curx,cury-
    textheight("Z"),bgimage,COPY_PUT);
        settextstyle(SANS_SERIF_FONT,HORIZ_DIR,0);
        outtextxy(curx,cury,mesg);
```

```
    m.show();
    tosend.ID=ID;
    tosend.x1=curx; tosend.y1=cury;
    strcpy(tosend.text,mesg);
    fillstatus();
    sendq.put(tosend);
    mesg[0]='\0';  // clear message //
    free(bgimage);
    bgimage=NULL;
}

void text::beginnewtext(int x, int y)
{
    curx=x; cury=y;
    pendown=1;
    settextstyle(SANS_SERIF_FONT,HORIZ_DIR,0);
    bgimage=malloc(imagesize(curx,cury-textheight("Z"),639,cury+8));
    m.hide();
    getimage(curx, cury-textheight("Z"),639,cury+8,bgimage);
    m.show();
}

void text::drawcomplete(drawinfo *drw)
{
    settextstyle(SANS_SERIF_FONT,HORIZ_DIR,0);
    settextjustify(LEFT_TEXT,BOTTOM_TEXT);
    setcolor(drw->col);
    outtextxy(drw->x1,drw->y1,drw->text);
    settextjustify(LEFT_TEXT,TOP_TEXT);
}

text::~text()
{
    if (bgimage!=NULL) free(bgimage);
    delbitmap(buttonbitmap);
}

void text::serve(void)
{
    currentbutton=Circle;
    firstflag=1;
    pendown=0;
    mesg[0]='\0';
    settextstyle(SANS_SERIF_FONT,HORIZ_DIR,0);
}

void text::redraw(void)
{
    blackbox(x1,y1,x2,y2);
    putbitmap(x1,y1,buttonbitmap);
}

void text::drawing(eventrecord e, int x, int y)
{
    static eventrecord lastleft;
    settextjustify(LEFT_TEXT, BOTTOM_TEXT);
    if (strlen(mesg)>MAXTEXTLONG) mesg[strlen(mesg)-1]='\0';
    if ((e.type==LeftDown)&&(inboard==1))
    {
        lastleft=e;
    }
    if ((e.type==LeftDown)&&(inboard==0))
    {
        if (e.y<120) finishtext();
```

```
    }
    if ((e.type==LeftRel)&&(lastleft.type==LeftDown)&&(inboard==1))
    {
        if (e.x-lastleft.x+e.y-lastleft.y<8)
        {
            if (pendown==1) finishtext();
            beginnewtext(e.x, e.y);
        }
        lastleft=e;
    }
    if ((pendown==1)&&(e.type==Keyboard))
    {
        int l;
        l=strlen(mesg);
        switch (e.x) {
        case 8  :   if (l>0) {
        /* backspace */mesg[l-1]='\0';
                        m.hide();
                        putimage(curx,cury-
textheight("Z"),bgimage,COPY_PUT);
                        settextstyle(SANS_SERIF_FONT,HORIZ_DIR,0);
                        outtextxy(curx,cury,mesg);
                        m.show();
                    }; break;

        case 13 : {
        /* enter */     finishtext();
                        if (cury+textheight("Z")<getmaxy())
                        beginnewtext(curx, cury+textheight("Z"));
                    }; break;

        default :   {
                        if (l<MAXTEXTLONG-1)
                        {
                            mesg[l]=e.x;
                            mesg[l+1]='\0';
                            setcolor(Color);
                            m.hide();
                            settextstyle(SANS_SERIF_FONT,HORIZ_DIR,0);
                            outtextxy(curx,cury,mesg);
                            m.show();
                        }
                    } break;
        }
    }
    settextjustify(LEFT_TEXT, TOP_TEXT);
}

class colorbutton
{
    private:
        char title[8];
        int *c;
        int x1,y1;
    public:
        colorbutton(int x, int y, int *colorptr, char *t);
        void drawbutton(void);
        void work(int x, int y);
};

colorbutton::colorbutton(int x, int y, int *colorptr, char *t)
{
    strcpy(title, t);
    c = colorptr;
```

```
    x1 = x;
    y1 = y;
}

void colorbutton::drawbutton(void)
{
    int i, x ,curx, cury;
    blackbox(x1,y1,x1+89,y1+89);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
    setcolor(WHITE);
    outtextxy(x1,y1,title);
    for(i=0;i<=3;i++)
    {
        for(x=0;x<=3;x++)
        {
            setfillstyle(SOLID_FILL,x+(i*4));
            bar(x1+x*15+2,y1+i*15+15+2,x1+(x+1)*15-3, y1+(i+1)*15-3+15);
            if (x+(i*4)==(*c)) { curx=x1+x*15; cury=y1+i*15; }
        }
    }
    setcolor(15);
    rectangle(curx,cury+15,curx+14,cury+14+15);
}

void colorbutton::work(int x, int y)
{
    x=x-x1; y=y-y1;
    y-=15; /* minus title height */
    if (y>=0)
    {
        x/=15; y/=15;
        *c=x+(y*4);
    }
    drawbutton();
}

void thickness()
{
    setcolor(WHITE);
    setfillstyle(SOLID_FILL, MENUBGCOLOR);
    bar(BUT_X1,BUT_Y1-45, BUT_X1+60, BUT_Y1-1);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 0);
    outtextxy(BUT_X1, BUT_Y1-45, "Thick");
    setfillstyle(SOLID_FILL, Color);
    if (Color!=0) bar(BUT_X1,BUT_Y1-30,BUT_X1+(thick*2),BUT_Y1-
30+(thick*2));
    else rectangle(BUT_X1,BUT_Y1-30,BUT_X1+(thick*2),BUT_Y1-
30+(thick*2));
}

void gethex(int x, int y, unsigned char *buffer, int byte)
{
    unsigned char ch[40],idx=0,lastch,cx=0;
    ch[0]='\0';
    while(((lastch!=13)||(idx!=byte*2))&&(lastch!=27))
    {
        setfillstyle(SOLID_FILL,WHITE);
        bar(x+cx,y+10,x+cx+8,y+12);
        lastch=getch();
        if ((lastch==8)&&(idx>0))
        {
            idx--; /* backspace case */
            ch[idx]='\0';
        }
```

```
        if (((lastch>='0')&&(lastch<='9'))||
            ((lastch>='a')&&(lastch<='f'))||
            ((lastch>='A')&&(lastch<='F')))
        {
            ch[idx]=lastch;
            idx++; /* check character */
            ch[idx]='\0';
        }
        if (idx>byte*2) { idx--; ch[idx]='\0'; }
        blackbox(x,y,LEFTBOARD-12,y+10);
        outtextxy(x,y,ch);
        blackbox(x+cx,y+10,x+cx+8,y+12);
        cx=idx*8;
    }
    for(idx=0;idx<strlen(ch);idx++)
    {
        if ((ch[idx]>='0')&&(ch[idx]<='9')) lastch=ch[idx]-'0';
        if ((ch[idx]>='a')&&(ch[idx]<='f')) lastch=ch[idx]-'a'+10;
        if ((ch[idx]>='A')&&(ch[idx]<='F')) lastch=ch[idx]-'A'+10;
        if (idx%2==0) buffer[idx/2]=lastch*16;
        else buffer[idx/2]+=lastch;
    }
}

void gettextxy(int x, int y, char *buffer, int byte)
{
    unsigned char ch[40],lastch=0, cx=0, idx=0;
    ch[0]='\0';
    while((lastch!=13)&&(lastch!=27))
    {
        setfillstyle(SOLID_FILL,WHITE);
        bar(x+cx,y+10,x+cx+8,y+12);
        lastch=getch();
        if ((lastch==8)&&(idx>0))
        {
            idx--; /* backspace case */
            ch[idx]='\0';
        }
        if (((lastch>='0')&&(lastch<='9'))||
            ((lastch>='a')&&(lastch<='z'))||
            ((lastch>='A')&&(lastch<='Z')))
        {
            ch[idx]=lastch;
            idx++; /* check character */
            ch[idx]='\0';
        }

        if (idx>byte-1) { idx--; ch[idx]='\0'; }
        blackbox(x,y,LEFTBOARD-12,y+10);
        outtextxy(x,y,ch);
        blackbox(x+cx,y+10,x+cx+8,y+12);
    }
    buffer[0]='\0';
    if (lastch!=27) strcpy(buffer,ch);
}


int main(void)
{
    struct eventrecord e, lastleft, te;
    elementlist ele;
    element *temp, *curr;
    mycircle BCircle;
    myline BLine;
```

```
rect BRect;
freehand BFreehand;
text BText;
colorbutton BFill(BUT_X1,BUT_Y1,&FillColor,"Fill");
colorbutton BDraw(BUT_X1, BUT_Y1+90, &Color, "Draw");
drawinfo drw,toget;
NetAddr remote;
int grdrv=DETECT, grmode=VGAHI;
int lastx, lasty, i;
int shutdown=0, cx, cy;
int lastipxstat=0, stat=0;
char c,s[20]; /* use input address */
char ctmp;
char buffer[80];
IPX *ipxa;
queue recvq;
char usertofind[USERNAMELONG];
char username[USERNAMELONG];
union {
    int word;
    char byte[2];
} ch;
ipxa=new IPX;
printf("Enter username :");
scanf("%s",username);
ipxa->changeusername(username);
initgraph(&grdrv,&grmode,"c:\\data\\tia\\project\\");

if (graphresult() != grOk)  /* an error occurred */
{
   printf("Graphics error: %s\n", grapherrormsg(graphresult()));
   printf("Press any key to halt:");
   getch();
   exit(1);             /* return with error code */
}
m.setvscreen(getmaxx(),getmaxy());
floodfill(0,0,BGColor);
m.show();
BFill.drawbutton();
BDraw.drawbutton();
ele.add(&BFreehand);
ele.add(&BLine);
ele.add(&BRect);
ele.add(&BCircle);
ele.add(&BText);
ele.drawall();
thickness();
curr=&BFreehand;
rectangle(curr->x1,curr->y1, curr->x2, curr->y2);
lastleft.type=NoEvent;
setfillstyle(SOLID_FILL, WHITE);
m.hide();
bar(LEFTBOARD-10,0,getmaxx(),getmaxy()); /* fill whiteboard */
m.show();
settextjustify(LEFT_TEXT,TOP_TEXT);

ipxa->waiting();
ipxa->changeusername(username);
while(shutdown==0) /* Main loop */
{
    m.alwayscall();
    ipxa->alwayscall();
    stat=ipxa->getstat(0);
    if(stat!=lastipxstat)
```

```
        {
            m.hide();
            blackbox(STAT_X1,STAT_Y1,STAT_X2,STAT_Y2);
            settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
            setcolor(WHITE);
            switch (ipxa->getstat(0)) {
            case AVAIL : outtextxy(STAT_X1, STAT_Y1, "AVAILABLE"); break;
            case WAITING : outtextxy(STAT_X1, STAT_Y1, "WAITING"); break;
            case CONNECTING : outtextxy(STAT_X1, STAT_Y1, "CONNECTING");
break;
            case CONNECTED : outtextxy(STAT_X1, STAT_Y1, "CONNECTED");
break;
            case DISCONNECTING : outtextxy(STAT_X1, STAT_Y1,
"DISCONNECTING"); break;
            case DISCONNECTED : outtextxy(STAT_X1, STAT_Y1,
"DISCONNECTED"); break;
            case FINDING :  outtextxy(STAT_X1, STAT_Y1, "FINDING");
break;
            case FOUNDUSER : outtextxy(STAT_X1, STAT_Y1, "FOUNDUSER");
                             m.hide();
                             setcolor(WHITE);
                             settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
                             outtextxy(BUT_X1,BUT_Y1+185,"Found user");
                             outtextxy(BUT_X1, BUT_Y1+195, "Connect to?");
                             ctmp=getch();
                             if ((ctmp=='y')||(ctmp=='Y'))
                             {
                                ipxa->connect(ipxa->getuseraddr());
                                ipxa->alwayscall();
//                              ipxa->sendsignal(0,ACCEPTCONN);
                             }
                             else ipxa->waiting();
                             blackbox(BUT_X1, BUT_Y1+185, LEFTBOARD-
12,getmaxy()-10);
                             m.show();
                             sendq.clear();
                             break;

            case WAITFORUSERCONFIRM :
                             m.hide();
                             setcolor(WHITE);
                             settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
                             outtextxy(BUT_X1,BUT_Y1+185,"Incoming");
                             outtextxy(BUT_X1,BUT_Y1+195, "connection,");
                             outtextxy(BUT_X1,BUT_Y1+205, "Accept?,");
                             ctmp=getch();
                             if ((ctmp=='y')||(ctmp=='Y'))
                                 ipxa->accept(ipxa->getremoteaddr());
                             else ipxa->waiting();
                             blackbox(BUT_X1, BUT_Y1+185, LEFTBOARD-
12,getmaxy()-10);
                             m.show();
                             break;
            }
            lastipxstat=stat;
            m.show();
        }

        ch.word=bioskey(1);
        e=m.event.getevent();
        if (ch.word!=0)
        {
            ch.word=bioskey(0);
            if (ch.byte[0]==27)    shutdown=1;
```

```
            if (ch.byte[0]==4)
            {
                if ((ipxa->getstat(0)==CONNECTING)||(ipxa-
>getstat(0)==CONNECTED))
                ipxa->disconnect(0);
            }
            if (ch.byte[0]==1) /* Ctrl-A use to clear screen */
            {
                m.hide();
                setfillstyle(SOLID_FILL, WHITE);
                bar(LEFTBOARD-10,0,getmaxx(),getmaxy());
                toget.ID=ClearScr;
                sendq.put(toget);
                m.show();
            }
            switch (ch.byte[1]) {
                case 73 :   if (thick<10) thick++; m.hide(); thickness();
m.show(); break;
                case 81 :   if (thick>1) thick--; m.hide(); thickness();
m.show(); break;
                case 71 :   if (Color<16) Color++; else Color=0;
                            m.hide();
                            BDraw.drawbutton();
                            thickness();
                            m.show();
                            break;
                case 79 :   if (Color>0) Color--; else Color=15;
                            m.hide();
                            BDraw.drawbutton();
                            thickness();
                            m.show();
                            break;
                case 59 :   if (Fillflag==1)
                /* F1 */    {
                                Fillflag=0;
                                setfillstyle(CLOSE_DOT_FILL, DARKGRAY);
                                m.hide();
                                bar(BUT_X1,BUT_Y1+15, BUT_X1+60,
BUT_Y1+75);
                                m.show();
                            }
                            else
                            {
                                Fillflag=1;
                                BFill.drawbutton();
                            }
                            break;
                case 67 :    if(ipxa->getstat(0)!=CONNECTED)
                            {
                                m.hide();
                                setcolor(WHITE);
                                settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
                                outtextxy(BUT_X1,BUT_Y1+185,"Enter node
:");
                                gethex(BUT_X1, BUT_Y1+195 ,(char *)
&(remote.node),6);
                                outtextxy(BUT_X1,BUT_Y1+225, "Enter
network :");
                                gethex(BUT_X1, BUT_Y1+235, (char *)
&(remote.network), 4);
                                m.show();
                                ipxa->connect(remote);
                                delay(100);
                                ipxa->alwayscall();
```

```
                                sendq.clear();
                                blackbox(BUT_X1, BUT_Y1+185, LEFTBOARD-
12,getmaxy()-10);
                            }
                            break;
                case 68 :   if(ipxa->getstat(0)!=CONNECTED)
                /* F10 */   {
                                m.hide();
                                settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
                                outtextxy(BUT_X1,BUT_Y1+225, "User to find
:");
                                gettextxy(BUT_X1, BUT_Y1+235, (char *)
&(usertofind), 10);
                                m.show();
                                ipxa->finduser(usertofind);
                                blackbox(BUT_X1, BUT_Y1+185, LEFTBOARD-
12,getmaxy()-10);
                            } break;
            }
            inputc=ch.byte[0];
            te.type=Keyboard;
            te.x=ch.byte[0];
            m.event.createevent(te);
        }
//      if (e.type==RightDown) shutdown=1; /* for debug */
        if (e.type!=NoEvent)
        {
            if (e.type==LeftDown)
            {
                lastleft=e;
                if((e.x<60)&&(e.y>=BUT_Y1)&&(e.y<BUT_Y1+165))
                {
                    m.hide();
                    if (e.y>=BUT_Y1+90)
                    {
                        BDraw.work(e.x, e.y);
                        BDraw.drawbutton();
                    }
                    else
                    {
                        Fillflag=1;
                        BFill.work(e.x, e.y);
                        BFill.drawbutton();
                    }
                    m.show();
                }
            }
            if ((e.type==LeftRel)&&(lastleft.type==LeftDown))
            {
                temp=ele.getelement(e.x, e.y);
                if((temp->ID==(ele.getelement(lastleft.x,lastleft.y))->ID)
                    &&(temp->ID!=HeadID))/*checkfor click and rel point */
                {
                    m.hide();
                    temp->serve();
                    curr=temp;
                    ele.drawall();
                    setcolor(WHITE);
                    rectangle(temp->x1,temp->y1,temp->x2,temp->y2);
                    m.show();
                    lastleft.type=NoEvent;
                }
            }
        }
```

```
            m.refreshxy();
            cx=m.xpos(), cy=m.ypos();
            if (cx<LEFTBOARD)
            {
                cx=LEFTBOARD;
                inboard=0;
            } else inboard=1;
            curr->drawing(e, cx, cy);
            while(ipxa->abletosend(0))
            {
                if (sendq.get(&toget)==0) break;
                ipxa->senddata(0, (char far *)&toget, sizeof(drawinfo));
            }
            while(ipxa->getdata(0,(char far *)&toget)!=0) recvq.put(toget);
            while(recvq.get(&toget)!=0)
            {
                m.hide();
                if (toget.ID > HeadID)
                    (ele.getbyID(toget.ID))->drawcomplete(&toget);
                else
                {
                    switch (toget.ID)
                    {
                        case ClearScr : {
                            m.hide();
                            setfillstyle(SOLID_FILL, WHITE);
                            bar(LEFTBOARD-10,0,getmaxx(),getmaxy());
                            m.show(); } break;
                    }
                }
                m.show();
            }
            m.event.delevent();
        }
        closegraph();
        ipxa->endIPX();
        delete(ipxa);
        printf("%d\n", shutdown);
        for(i=0;i<=5;i++) printf("%x",remote.node[i]);
        return 0;
    }
```

Barry Nance. **Client/Server LAN Programming** Indianapolis : Que Corporation, 1994.

William A. Shay. **Understanding Data Communications and Networks** Boston : PWS Publishing Company, 1995.

Arron M. Tenenbaum, Moshe J. Augenstein. **Data Structures Using Pascal** 2nd Ed. New York : Prentice Hall, Inc., 1986