ระบบชุมสายโทรศัพท์อัตโนมัติผ่านเครือข่ายคอมพิวเตอร์
(เล่มที่ 2 ส่วนซอฟต์แวร์)

DIGITAL AUTOMATIC BRANCH EXCHANGE IN LOCAL AREA
(Part II Software)

โดย

นายชูเกียรติ เจียเจษฎากุล

นายธนวิชญ์ พันธุ์เจริญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2538

ระบบชุมสายโทรศัพท์อัตโนมัติผ่านเครือข่ายคอมพิวเตอร์
(เล่มที่ 2 ส่วนซอฟต์แวร์)
Digital Automatic Branch Exchange in Local Area Network
(Part II Software)

โดย

นายชูเกียรติ เจียเจษฎากุล
นายธนวิชญ์ พันธุ์เจริญ

อาจารย์ที่ปรึกษา

อ.ฐุรสิทธิ์ วรรณไกรโรจน์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2538

ปริญญานิพนธ์ ปีการศึกษา 2538

ภาควิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบชุมสายโทรศัพท์อัตโนมัติผ่านเครือข่ายคอมพิวเตอร์ (เล่มที่ 2 ส่วนซอฟต์แวร์)

(Digital Automatic Branch Exchange in Local Area Network (Part II Software))

ผู้จัดทำ

1. นายชูเกียรติ เจียเจษฎากุล
2. นายธนวิชญ์ พันธุ์เจริญ

อาจารย์ที่ปรึกษา

....

(อ.สุรสิทธิ์ วรรณไกรโรจน์)

ระบบชุมสายโทรศัพท์อัตโนมัติผ่านเครือข่ายคอมพิวเตอร์

นายชูเกียรติ เจียเจษฎากุล
นายธนวิชญ์ พันธุ์เจริญ
อ.สุรสิทธิ์ วรรณไกรโรจน์
ปีการศึกษา 2538

บทคัดย่อ

ระบบชุมสายโทรศัพท์ผ่านเครือข่ายคอมพิวเตอร์นี้ มาจากแนวความคิดที่จะพัฒนานำไปใช้ในการเชื่อมโยงเครือข่าย โดยผ่านทางระบบเครือข่ายคอมพิวเตอร์ท้องถิ่น ให้สามารถโทรศัพท์ติดต่อถึงกันได้เช่นเดียวกับที่ผ่านชุมสายโทรศัพท์ โดยใช้ไมโครคอมพิวเตอร์ลูกข่ายทำงานนี้แทน

โครงงานนี้แบ่งเนื้อหาสำคัญเป็น 2 เล่มคือ เล่ม 1 จะเป็นส่วนที่กล่าวฮาร์ดแวร์ของแผ่นการ์ดที่ใช้กับเครื่องไมโครคอมพิวเตอร์ในการติดต่อสื่อสาร และ เล่ม 2 เป็นเนื้อหาของโปรแกรมที่นำมาประยุกต์ใช้งาน ในการควบคุมหลัก

เนื้อหาภายในเล่ม 2 นี้จะเป็นโปรแกรมประยุกต์ทั้งหมดและฟังก์ชันการใช้งาน ที่ใช้ทำงานร่วมกับส่วนฮาร์ดแวร์ (เล่ม 1) ที่พัฒนาขึ้น เพื่อให้สามารถทำงานควบคุมติดต่อสื่อสารผ่านในระบบเครือข่าย ที่ติดต่อกับอุปกรณ์ปลายทางที่ติดต่อกับไมโครคอมพิวเตอร์เครื่องเดียวกันได้ เช่น การควบคุมการส่งสัญญาณของโทรศัพท์ การอินเตอร์รัพท์การติดต่อ การสวิตช์ชิ่ง รวมทั้งการส่งผ่านข้อมูลโดยใช้ฟังก์ชันของ Netware เป็นต้น

Digital Automatic Branch Exchange in Local Area Network

Mr. Chukiat Jearjesdakul

Mr. Tanawit Punjaroen

Advisor Surasit Wannakrairoj

Academic Year 2538

Abstract

Digital Automatic Branch Exchange in Local Area Network is another development of telecommunication through LAN - Using microcomputer client to make a telephone call and connect to another in the same way as common PABX (Private Automatic Branch Exchange) does .

This thesis has two main parts : Book I will give the hardware information of the designed card which develope to operate on PC (Personal Computer) and Book II will demonstrate the application software, Netware's communication service function to control the system .

The content of Book II is the Application software that support hardware perfromance (Book I) to control ther supervisory function of PABX either on LAN or connect with another destination telephone on the same node , i.e. control sending all signal ,interrupt service switch and transfer data by Netware distribute communication function

สารบัญ

สารบัญรูปภาพ

ชื่อรูป

หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญและความเป็นมาในการทำการศึกษา

ในปัจจุบันเป็นโลกแห่งการแข่งขันในด้านต่าง ๆ ทางธุรกิจ ซึ่งผู้ที่จะประสบความสำเร็จและอยู่รอดได้นั้นจะต้องมีหูตาที่กว้างไกล มีวิสัยทัศน์ที่ดี หรือพูดง่าย ๆ ก็คือ ต้องมีข้อมูลที่ดีเพียงพอต่อการที่จะตัดสินใจในการจัดการต่าง ๆ อีกต้องแข่งขันกันทางด้านเวลาด้วย ซึ่งโทรศัพท์ก็เป็นอุปกรณ์สื่อสารที่สำคัญอย่างหนึ่งที่สามารถ ส่งข้อมูลและข่าวสารได้รวดเร็วและ นับว่าเป็นสิ่งจำเป็นที่สุดสิ่งหนึ่งสำหรับสำนักงานทั่ว ๆ ไป เพื่อจะดำเนินธุรกิจให้สำเร็จลุล่วงตามเป้าหมายที่กำหนดไว้ จึงมีความสนใจในการพัฒนาระบบโทรศัพท์เพื่อที่จะใช้ในสำนักงาน และ จากการที่สำนักงานส่วนใหญ่มักจะมีระบบเครือข่ายคอมพิวเตอร์อยู่แล้ว ซึ่งการใช้งานส่วนใหญ่จะใช้ในการพิมพ์เอกสารและและทำรายงานต่าง ๆ ซึ่งถือว่าใช้ประโยชน์จากระบบเครือข่ายคอมพิวเตอร์อยู่ในปริมาณที่ น้อยถือว่าไม่คุ้มค่ากับราคา ซึ่งน่าจะสามารถใช้ประโยชน์ได้มากกว่านี้ จึงเกิดความคิดที่จะพัฒนาระบบโทรศัพท์ ซึ่งจะใช้คอมพิวเตอร์มาทำงานเป็นชุมสายอัตโนมัติ ซึ่งมีข้อดีคือทำให้คอมพิวเตอร์มีประโยชน์มากขึ้น และไม่ต้องวางระบบสายโทรศัพท์เพิ่มเนื่องจาก มีระบบเครือข่ายคอมพิวเตอร์อยู่แล้ว ซึ่งจะใช้ระบบเครือข่ายคอมพิวเตอร์ใช้ติดต่อระหว่างชุมสายอัตโนมัติ

1.2 วัตถุประสงค์

เป็นการประยุกต์ใช้งานระบบเครือข่ายคอมพิวเตอร์ ให้เกิดประโยชน์มากขึ้นโดยใช้การติดต่อระหว่างคอมพิวเตอร์ โดยเพียงแต่นำโทรศัพท์เชื่อมต่อกับเครื่องคอมพิวเตอร์ที่ต่อบนระบบเครือข่ายคอมพิวเตอร์อยู่แล้ว ซึ่งคอมพิวเตอร์จะทำหน้าเป็นชุมสายโทรศัพท์อัตโนมัติย่อยโดยสามารถมีลูกข่ายได้อย่างน้อย 4 เครื่องและสามารถติดต่อข้ามชุมสายได้โดยผ่านระบบเครือข่ายคอมพิวเตอร์ ซึ่งจะทำหน้าที่เป็นทรั้ง (TRANK) ระหว่างชุมสายและเป็นการใช้ไมโครคอมพิวเตอร์ให้ได้ประโยชน์มากขึ้นด้วย

1.3 ขอบข่ายความสามารถของโครงงานมีดังนี้

1. คอมพิวเตอร์ 1 เครื่องสามารถติดต่อโทรศัพท์ได้อย่างน้อย 4 เครื่อง
2. สามารถติดต่อกันภายในชุมสายได้
3. สามารถติดต่อระหว่างชุมสายโดยอาศัยระบบเครือข่ายคอมพิวเตอร์ได้

4. ขณะที่มีการติดต่อกันระหว่างโทรศัพท์ เครื่องคอมพิวเตอร์สามารถทำงานอย่างอื่นได้ด้วย

1.4 วิธีการและแผนการทำงาน

วิธีการและแผนการทำงานสามารถแบ่งแผนการดำเนินงานได้ดังนี้

1. ศึกษาระบบโทรศัพท์และคอมพิวเตอร์
2. ออกแบบโครงสร้างและขั้นตอนการทำงานของชุมสาย (การติดต่อภายในชุมสายและภายนอกชุมสาย)
3. ศึกษาการเขียนซอฟท์แวร์ (Software) ด้วยภาษา C
4. ศึกษาการทำงานของระบบคอมเครือข่ายคอมพิวเตอร์
5. ศึกษาการทำงานของ Netware
6. ศึกษาการทำงานและการใช้งาน Communication Service ของ Netware (IPX and SPX Protocol)
7. เขียนโปรแกรมทั้งหมดด้วยภาษา C โดยใช้ Library ของ Netware C Interface - Dos
8. ทดลองใช้โปรแกรมกับส่วนฮาร์ดแวร์ (Hardware)
9. สรุปและวิจารณ์ผลการทดลอง

1.5 ประโยชน์ที่คาดว่าจะได้รับ

- เพื่อประโยชน์ในการสื่อสาร โดยการใช้โทรศัพท์ให้มีประสิทธิภาพมากยิ่งขึ้น
- เพื่อใช้ระบบเครือข่ายคอมพิวเตอร์ให้เป็นประโยชน์มากยิ่งขึ้น
- เพื่อใช้คอมพิวเตอร์ให้เป็นเป็นประโยชน์มากยิ่งขึ้น
- เพื่อเป็นประโยชน์ทางธุรกิจในการติดต่อสื่อสารต่าง ๆ

บทที่ 2
เครือข่ายคอมพิวเตอร์ท้องถิ่น

ระบบเครือข่ายคอมพิวเตอร์ (computer network) ประกอบด้วยคอมพิวเตอร์หลายตัวที่สามารถติดต่อเพื่อแลกเปลี่ยนข้อมูลกันได้ การติดต่อจะผ่านทางช่องทางการสื่อสารต่าง ๆ สามารถแบ่งประเภทของระบบเครือข่ายได้ดังนี้

1. Wide Area Network (WAN) เป็น ระบบเครือข่ายที่ติดตั้งใช้งานอยู่ในบริเวณกว้าง เช่น Internet มีโมเด็ม (Modem) เป็นอุปกรณ์ในการสื่อสาร
2. Local Area Network (LAN) เป็นระบบเครือข่ายที่ใช้อยู่ในบริเวณไม่กว้างมากนัก เช่น ภายในมหาวิทยาลัย การส่งข้อมูลได้ด้วยความเร็วสูงผิดพลาดน้อย ระบบเครือข่ายท้องถิ่นออกแบบมาเพื่อลดต้นทุนและเพิ่มประสิทธิภาพ การใช้งานอุปกรณ์ต่าง ๆ ร่วมกัน
3. Metropolitan Area Network (MAN) เป็นระบบเครือข่ายที่มีขนาดอยู่ระหว่าง LAN กับ WAN ใช้ในตัวเมืองหรือจังหวัดเท่านั้น

2.1 ความสามารถของระบบเครือข่ายท้องถิ่น

LAN มีข้อดีเหนือมินิคอมพิวเตอร์หรือเมนเฟรม (Mainframe) คือ มีการประมวลผลแบบกระจายงาน (distributed processing) ความรวดเร็วในการติดต่อสื่อสารการติดต่อระหว่างสถานีผู้ใช้งาน (interconnect station) ใช้โปรแกรมและข้อมูลร่วมกันได้ และใช้ฮาร์ดแวร์และ ทรัพยากรที่อยู่ร่วมกันได้

การประมวลผลแบบกระจายงาน ในการประมวลผลแบบกระจายเมื่อสถานีของ ผู้ใช้บริการขอโปรแกรม และข้อมูลจาก file server โปรแกรมคำสั่งจะถูกสำเนาจากศูนย์บริการข้อมูลไปยังหน่วยความจำที่สถานีของผู้ใช้ เมื่อเสร็จสิ้นการใช้งานโปรแกรมและไฟล์ (File) ข้อมูลแล้วสถานีผู้ใช้จะส่งข้อมูลกลับไปเก็บยัง file server เพื่อใช้งานต่อไป

การประมวลผลแบบกระจายงานจะช่วยให้สถานีหลายๆ สถานีใช้งานร่วมกันในระบบได้ โดยไม่ลดความสามารถในการประมวลผลข้อมูลของแต่ละสถานี ในเครื่องคอมพิวเตอร์แบบเมนเฟรมความสามารถในการประมวลผลจะถูกแบ่งออกไปในแต่ละสถานี ถ้ายิ่งมีสถานีมากก็จะทำให้ประสิทธิภาพของการทำงานลดลง ในการกระจายการประมวลผลจะทำให้เพิ่มความเร็วและประสิทธิภาพของระบบได้

ความรวดเร็วในการติดต่อสื่อสาร LAN มีสื่อใช้ในการส่งข้อมูลเป็นของตนเอง ทำให้การติดต่อสื่อสารมีความเร็วสูงมากกว่า 1 ล้านบิตต่อวินาที (Megabit/sec) ช่วยทำให้ข้อมูลทันต่อเหตุการณ์ ลดความซ้ำซ้อนของข้อมูล และเพิ่มความถูกต้องแม่นยำ

การติดต่อระหว่างสถานีผู้ใช้งาน แต่ละสถานีถูกเชื่อมต่อเข้าด้วยกัน ทำให้ผู้ใช้ติดต่อหรือส่งข้อมูลระหว่างกันได้ เช่น จดหมายอิเล็กทรอนิกส์

การใช้ประโยชน์ร่วมกัน ในระบบเครือข่ายคอมพิวเตอร์ผู้ใช้สามารถจะใช้ข้อมูลและซอฟต์แวร์ร่วมกันได้เพื่อลดความซ้ำซ้อนของข้อมูล สื่อบันทึกข้อมูล ค้นหาตรวจสอบข้อมูลได้รวดเร็วถูกต้อง

การใช้ทรัพยากรร่วมกัน ในระบบเครือข่ายคอมพิวเตอร์อนุญาตให้ใช้อุปกรณ์ร่วมกันช่วยลดความซ้ำซ้อนของอุปกรณ์ทำให้ประหยัดค่าใช้จ่าย

ระบบเครือข่ายคอมพิวเตอร์ท้องถิ่นได้พัฒนาขึ้นมาหลายแบบ แต่ที่นิยมใช้กันมากกว้างขวาง เป็นระบบเครือข่ายท้องถิ่นแบบ ISO-OSI (Open System Interconnection) ซึ่งมีคุณสมบัติต่าง ๆ ดังนี้

1. มีความยืดหยุ่นสูง ง่ายต่อการเปลี่ยนแปลงโครงสร้างในการเพิ่มหรือลดอุปกรณ์ในเครือข่าย
2. เป็นโครงสร้างที่สามารถขยายสาขาพิเศษออกไปได้เมื่อต้องการ
3. ตัวกลางในการส่งข้อมูลและระบบควบคุมสามารถขยายออกไปได้
4. เป็นระบบที่ความเชื่อถือได้สูง ถ้าสายส่งข้อมูลหรือเครื่องใดเสียหายจะไม่ทำให้ การรับส่งข้อมูลของเครื่องอื่นเสียหายไปด้วย
5. เป็นระบบที่กระจายการควบคุม เพื่อเพิ่มความเชื่อถือได้ของข้อมูล

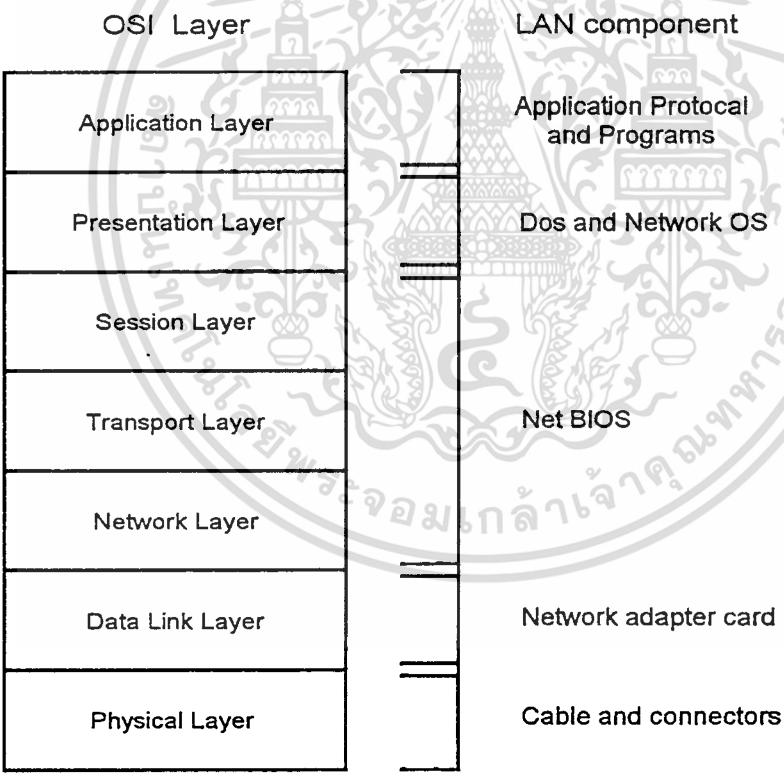
2.2 มาตราฐานของระบบเครือข่ายท้องถิ่น

ระบบเครือข่ายท้องถิ่นมีการใช้งานอย่างกว้างขวาง จึงมีการกำหนดมาตราฐานของระบบขึ้นเพื่อให้ระบบเครือข่ายของ แต่ละบริษัทที่มีความแตกต่างกันในด้านโทโพโลยี (Topology) และชนิดของสายส่งข้อมูล สามารถเชื่อมต่อกันได้

สถาบันที่สำคัญในการกำหนดมาตราฐานก็เช่น IEEE (Institute of Electrical and Electronic Engineers) และ ANSI (American National Standards Institute) ทั้งสองสถาบันได้กำหนดมาตราฐานฮาร์ดแวร์ของระบบเครือข่ายท้องถิ่นขึ้นมา รวมทั้งคำจำกัดความอย่างเป็นทางการของระบบเครือข่าย เช่น Ethernet ของบริษัท Xerox ,IBM Token-Ring ของ บริษัท IBM เป็นต้น

2.3 OSI Model

Open System Interconnection (OSI) เป็นโมเดล (Model) ที่พัฒนาขึ้นโดย Inter National Standard Organization (ISO) เพื่อเป็นมาตรฐานในการออกแบบโทโพโลยี ในการติดต่อสื่อสารระหว่างคอมพิวเตอร์ และเป็นโครงสร้างที่ใช้กันอย่างแพร่หลายสำหรับเปรียบเทียบระบบเครือข่ายทางทฤษฎี โมเดลของ OSI แบ่งออกเป็น 7 ชั้น ซึ่งแต่ละชั้นจะประกอบกันเป็นพื้นฐานเพื่อเป็นแนวความคิดในการออกแบบและการนำระบบเครือข่ายไปใช้งาน โมเดล OSI จะบรรยายกระบวนการติดต่อสื่อสารตามลำดับขั้นของชั้น แต่ละชั้นจะกำหนดการติดต่อกับชั้นที่ติดกัน ซึ่งการติดต่อเหล่านี้มีความยืดหยุ่นที่จะให้ผู้ออกแบบระบบใช้โปรโตคอล ได้หลายแบบ โดยที่ระบบยังอยู่ในมาตรฐาน



รูปที่ 2.1 โมเดล OSI เทียบกับองค์ประกอบในระบบเครือข่ายท้องถิ่น

ชั้นที่ 1 Physical Layer

- Voltage และ Electrical Pulse
- สื่อ (Media) และตัวติดต่อกับสื่อ (Media Interface)

- การกำหนดพิน (pin assignments)
- โทโพโลยีของระบบเครือข่าย

ในชั้นนี้เป็นการกำหนดกลไก สัญญาณไฟฟ้า การติดต่อกับระบบย่อย (Subnet) และกำหนดโทโพโลยีของระบบเครือข่าย

ชั้นที่ 2 Data Link Layer

- Framing
- Error Control
- Data transperency

กำหนดโปรโตคอลที่เครื่องแม่จะต้องใช้ในการส่งและรับข้อมูลบนระบบเครือข่าย Data Link Layer จะประกอบข้อมูล เป็น Physical- layer-service-data -units หรือเฟรม (Frame) นอกจากนี้ชั้นนี้ยังใช้การแก้ไขข้อผิดพลาดทางกายภาพ (Physical Error) ของระบบเครือข่าย แต่ไม่มีหน้าที่แก้ไขข้อผิดพลาดของโปรโตคอล

ชั้นที่ 3 Network Layer

- Routing ภายใน Subnet
- Sequenced Delivery
- Congestion Control

มีหน้าที่ควบคุมการทำงานของ Subnet และรับผิดชอบในการส่งกลุ่มข้อมูลสื่อสารให้ถูกต้องในระบบเครือข่าย

ชั้นที่ 4 Transport Layer

- การจัดการเชื่อมต่อระหว่างเครื่องแม่กับเครื่องแม่
- Message Segmentation และ Multiplexing
- Reliable End-to-end Delivery
- End-to-end Flow Control

รับผิดชอบในการติดต่อสื่อสาร ระหว่างกระบวนการของเครื่องแม่โดยไม่มีข้อผิดพลาด อาจทำโดยผ่าน Circuit service หรือ Datagram Service ก็ได้ แบบ Circuit จะเหมือนกับการพูดโทรศัพท์ ส่วนแบบ Datagram จะเหมือนการส่งจดหมาย

ชั้นที่ 5 Session Layer

- การเริ่มเชื่อมต่อ การเลิกการติดต่อ
- Dialogue Discipline
- การ Synchronize ระหว่าง Task ของผู้ใช้ (End-user)

ทำงานเหมือนกับการติดต่อระหว่างผู้ใช้กับระบบเครือข่าย ให้การบริการโดยตรงที่ให้อุปกรณ์ต่าง ๆ อ้างอิงด้วยชื่อแทนที่จะอ้างถึงโดยตำแหน่งของมัน และมีหน้าที่แก้ไขข้อผิดพลาดที่เกิดจากการส่งข้อมูลที่ขาดความเชื่อถือ

ชั้นที่ 6 Presentation Layer

- การกำหนดรูปแบบของข้อมูล (Syntax)
- การลดขนาดของข้อมูล (Data Compression)
- การเข้ารหัสข้อมูล (Data Eneryption)

ชั้นนี้จะกำหนดว่าโปรแกรมประยุกต์จะเข้าสู่ระบบเครือข่ายได้อย่างไร และทำงานบางอย่าง เช่น การลดขนาดข้อมูล

ชั้นที่ 7 Application

- การติดต่อสื่อสารระหว่างกระบวนการกับกระบวนการในโปรแกรม
- ตัวอย่างของโปรโตคอล เช่น
  - File Transfer Access and Management (FTAM)
  - Virtual Terminal (VT)
  - Message Handling System (MHS)
  - Transaction Processing (TP)

2.4 โมเดล OSI กับระบบเครือข่ายท้องถิ่น

ความสัมพันธ์ขององค์ประกอบในระบบเครือข่ายท้องถิ่นกับโมเดล OSI มีดังนี้

ชั้นที่ 1 รวมถึงสื่อที่ใช้ในการส่งข้อมูล เช่น สายโทรศัพท์ สายโคแอกเชียล (Coaxial) หรือเส้นใยแก้วนำแสง (Optic Fiber) รวมถึงตัวเชื่อมต่อ ตัวขยายสัญญาณและอุปกรณ์อื่น ๆ ในการรับส่งบิตข้อมูล

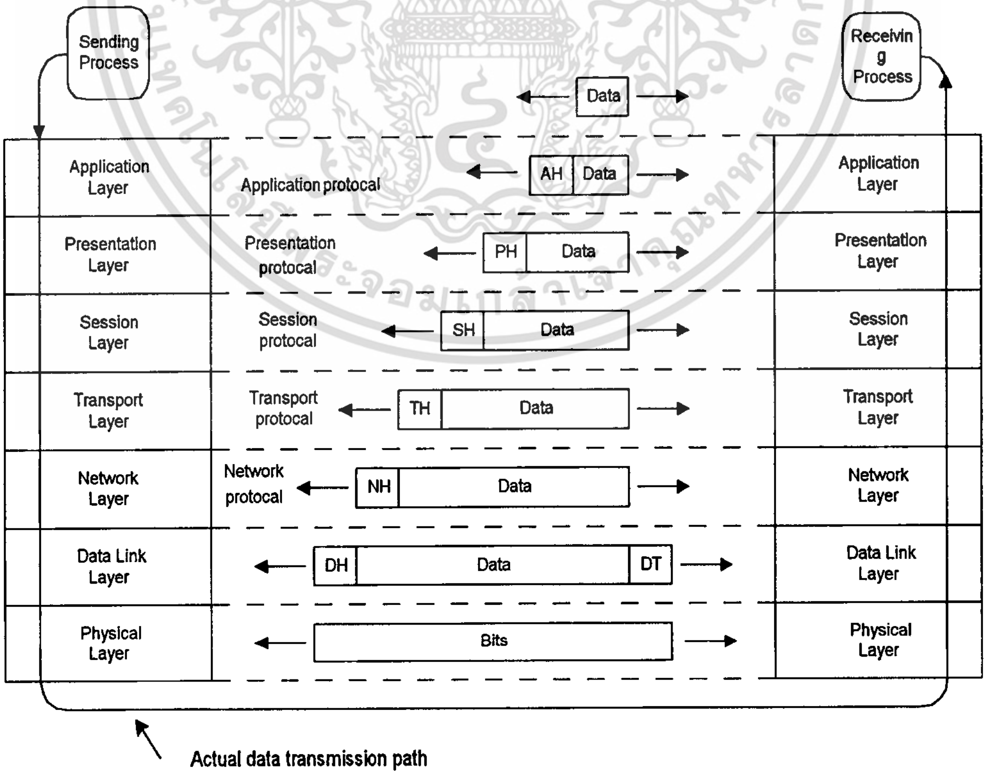
ชั้นที่ 2 นำไปใช้งานบน Network Interface Card (NIC) กำหนดข้อมูลจากชั้นที่สูงขึ้นไปให้อยู่ในลำดับที่กำหนด Ethernet , Token ring , star LAN , ARCNET และรูปแบบเฟรมอื่น ๆ จะแตกต่างกันออกไป ซึ่งแต่ละเฟรมจะเป็นข้อมูลพื้นฐานของระบบเครือข่ายท้องถิ่น การทำงานอื่น ๆ ในชั้นนี้ก็คือ กำหนดตำแหน่งของต้นทางและปลายทางเพื่อระบุผู้ส่งและผู้รับของเฟรม รวมทั้งเช็คความผิดพลาดต่าง ๆ โดยวิธี CRC (cyclic redundancy check)

ชั้นที่ 3 ถึง 5 นำไปใช้งานใน NetBIOS (Network Basic Input/Output System) NetBIOS จะมีฟังก์ชันสำหรับ Session Layer เพื่อจัดการเชื่อมต่อทางตรรกะ ระหว่างสถานีผู้ใช้ในระบบเครือข่ายท้องถิ่น ส่วนฟังก์ชันใน Transport Layer และ Network Layer ไม่ต้องใช้ในระบบเครือข่ายท้องถิ่น แต่จะพบได้ใน WAN

NetBIOS อาจจะอยู่ใน ROM บนการ์ดเชื่อมโยงระบบเครือข่าย หรือจำลองอยู่ในซอฟท์แวร์ที่สถานีผู้ใช้และศูนย์บริการข้อมูล

ชั้นที่ 6 นำไปใช้งานระบบปฏิบัติการของระบบเครือข่าย (NOS) และการจำลองใน DOS ที่สัมพันธ์กัน

ชั้นที่ 7 โปรแกรม Protocol และโปรแกรมประยุกต์



รูปที่ 2.2 ลักษณะของข้อมูลที่ถูกส่งผ่านในแต่ละ Layer

2.5 องค์ประกอบของ LAN

ระบบ LAN มีองค์ประกอบหลักๆ 3 ส่วนดังนี้

- อุปกรณ์ด้านฮาร์ดแวร์
- โปรแกรมควบคุมระบบ หรือ ซอฟท์แวร์
- ตัวกลางนำข้อมูล (Media)

ฮาร์ดแวร์ที่ใช้เชื่อมต่อระบบ LAN ส่วนใหญ่จะออกแบบมาเป็นการ์ด หรือ แผงวงจรที่ใส่ลงในช่องสล็อต (slot) ของไมโครคอมพิวเตอร์ ซึ่งการ์ดเหล่านี้จะมีช่องอยู่ที่ด้านหลังของเครื่องไมโครคอมพิวเตอร์ เพื่อให้ต่อเข้ากับสายที่ใช้เชื่อมโยงเครือข่าย หรือที่เรียกว่า ตัวกลางนำข้อมูล ซึ่งโดยทั่วไปจะใช้สายได้หลายประเภท เช่น สายโทรศัพท์ สายโคแอกเชียล สายนำสัญญาณด้วยแสง เป็นต้น ขึ้นอยู่กับ Network card ที่ใช้ แต่ทั่ว ๆ ไปมักใช้สายโคแอกเชียล เป็นส่วนใหญ่

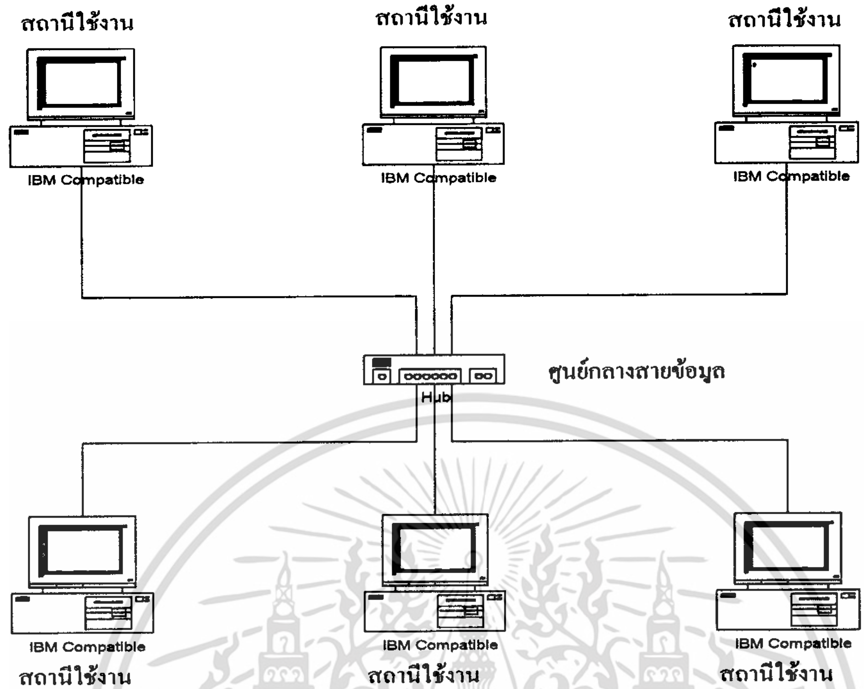
นอกจากอุปกรณ์สองส่วนที่กล่าวมาแล้ว โปรแกรมควบคุมระบบ Network หรือที่เรียกว่า Network Operating System (NOS) ก็เป็นอีกส่วนหนึ่งที่มีความสำคัญมาก เพราะนอกจากฮาร์ดแวร์แล้วการที่ระบบจะรู้ว่าข้อมูลที่ใช้งานอยู่ในเครื่องคอมพิวเตอร์ของตนเอง หรือ เครื่องอื่นในระบบนั้นโปรแกรมควบคุมระบบ Network จะเป็นผู้จัดการ ดังนั้นโปรแกรมควบคุมระบบที่ดีจะต้องทำให้ผู้ใช้ไม่เห็นความแตกต่างของการใช้งาน ไม่ว่าจะอยู่ในเครื่องคอมพิวเตอร์ของตนเอง หรือทำงานโดยใช้ข้อมูลจากเครื่องคอมพิวเตอร์อื่นในระบบเครือข่ายคอมพิวเตอร์ ซึ่งโปรแกรมควบคุมระบบ LAN ที่ทำงานภายใต้ระบบปฏิบัติการอื่นเช่น OS/2 LAN Manager อีกประเภทหนึ่งคือ โปรแกรมควบคุมระบบ LAN ที่สร้างระบบปฏิบัติการของตนเองขึ้นมา เช่น NetWare เป็นต้น

2.6 ลักษณะการต่อสาย LAN (Topology)

มีอยู่ 3 แบบใหญ่ ๆ คือ

1. Star Topology

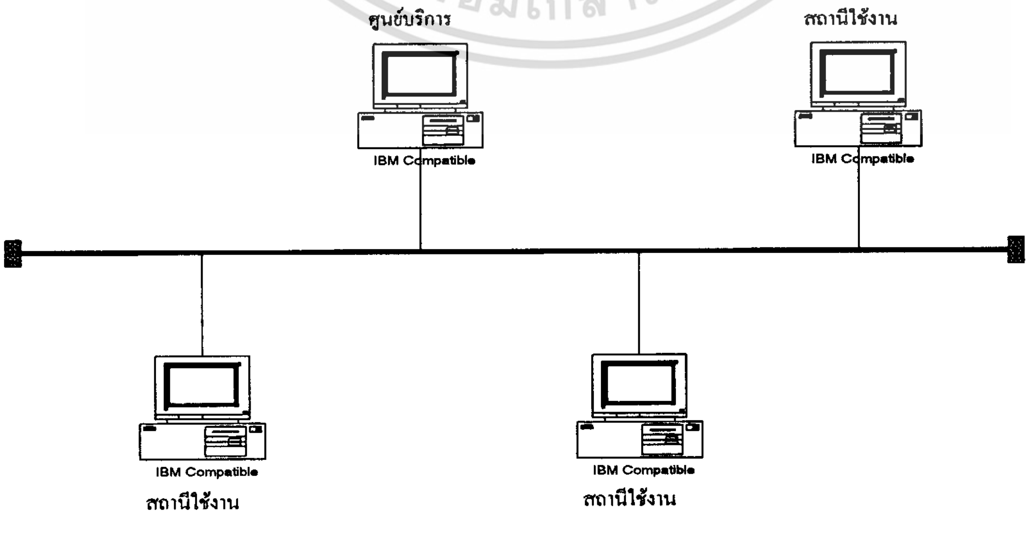
เทอร์มินัลของผู้ใช้ (User Terminal) จะต่ออยู่กับศูนย์กลางสายส่งข้อมูลเพียงตัวเดียว เมื่อเทอร์มินัลใดต้องการติดต่อกับเครื่องอื่นๆ ก็ต้องผ่านตัวกลางนี้ก่อน



รูปที่ 2.3 โครงสร้างการเชื่อมต่อแบบดาว

2. Bus Topology

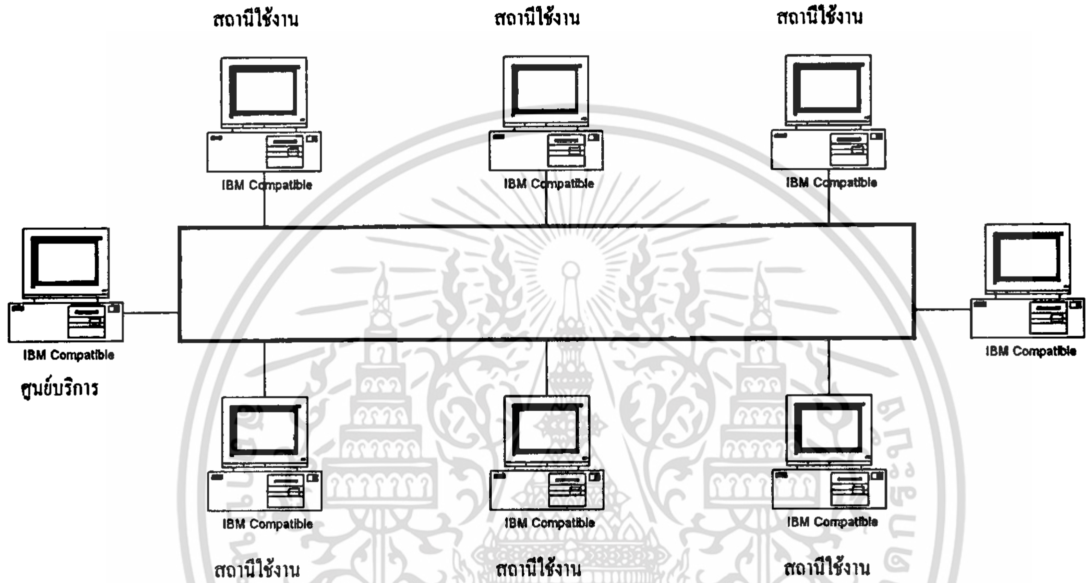
มีสายส่งข้อมูลอันเดียวเรียกว่า Bus หรือ Trunk ทุกสถานีต่ออยู่กับ bus และที่ปลายทั้งสองด้านปิดด้วย terminator



รูปที่ 2.4 โครงสร้างการเชื่อมต่อแบบบัส

3.Ring Topology

มีสถานีในระบบต่อกันเป็น Loop โดยสถานีสุดท้ายจะต่อกับสถานีแรก ทำให้รูปแบบของระบบเป็นวงแหวน ข้อมูลที่ส่งในระบบนี้จะผ่านทุกสถานี โดยอุปกรณ์หรือสถานีที่ส่งจะต้องคอยดูจากกลุ่มข้อมูลสื่อสาร (packet) ที่มันส่งเพื่อที่จะไม่ส่งข้อมูลสื่อสารนั้นซ้ำอีก



รูปที่ 2.5 โครงสร้างการเชื่อมต่อแบบวงแหวน

2.7 ลักษณะการแบ่งการใช้สาย (Media Access Control)

โดยทั่วไปมี 2 แบบ คือ

1. CSMA/CD(Carrier Sense Multiple Access/Collision Detect)

วิธีนี้ใช้ในกรณีทุกเครื่องต่ออยู่กับสายชุดเดียวกัน คือ Bus Topology โดยที่ขณะใดขณะหนึ่งคอมพิวเตอร์แต่ละเครื่องจะคอย "ฟัง" ว่าสายว่างหรือไม่ (carrier sense) ถ้าว่างก็จะเริ่มส่งสัญญาณออกมา ถ้าสายว่างข้อมูลก็จะไปถึงผู้รับได้เลย แต่การเริ่มส่งสัญญาณนี้อาจเกิดขึ้นจากหลายๆสถานี พร้อมๆกันได้ ผลก็คือ สัญญาณที่ได้จะมีการชนกันในสายทำให้ข้อมูลใช้ไม่ได้ เพราะฉะนั้นแต่ละเครื่องต้องสามารถตรวจสอบการชนกัน (Collision Detect) ได้ จากนั้นแต่ละเครื่องก็จะหยุดส่งและรอ โดยนับถอยหลังในเวลาที่สุ่มขึ้นมาให้แตกต่างกันระหว่างแต่ละเครื่อง เมื่อครบเวลาที่นับของแต่ละเครื่องแล้วก็ค่อยส่งข้อมูลออกมาใหม่ ซึ่งเนื่องจากเวลาที่ใช้รอแตกต่างกัน การส่งครั้งใหม่จึงไม่โอกาสชนกันระหว่างคู่เดิมอีก

2. Token-passing

วิธีนี้ใช้กับโทโพโลยีได้หลายแบบ ไม่ว่าจะเป็น Bus , Star หรือ Ring โดยในขณะใดขณะหนึ่งจะมีคอมพิวเตอร์เพียงเครื่องเดียวในระบบเครือข่าย ที่มีสิทธิในการส่งข้อมูลโดยมีรหัสที่เรียกว่า Token เก็บไว้ เมื่อส่งข้อมูลออกไปเสร็จแล้วก็จะส่งรหัส Token นี้ออกไปให้เครื่องอื่นๆ ตามลำดับที่กำหนดไว้ล่วงหน้า เครื่องอื่น ๆ เมื่อได้รับรหัสแล้ว ถ้าเครื่องไหนยังไม่ต้องการส่งข้อมูลก็จะส่งรหัสต่อไปเลย แต่ถ้าต้องการส่งข้อมูลก็ให้ข้อมูลออกมาก่อน แล้วค่อยส่งรหัสตามออกไปให้เครื่องอื่นตามลำดับด้วยวิธีนี้ทุกเครื่องจะได้รับสิทธิในการส่งข้อมูล 1 ครั้งภายใน 1 รอบการทำงาน ทำให้สามารถจำกัดเวลาได้ว่าจะส่งข้อมูลออกไปภายในเวลาไม่เกินกี่มิลลิวินาที

บทที่ 3

NetWare and NetWare Interface

NetWare เป็นโปรแกรมควบคุมระบบเครือข่ายคอมพิวเตอร์ หรือ "Network Operating System" โปรแกรมหนึ่งที่พัฒนาขึ้นโดยบริษัท Novell ระบบปฏิบัติการ NetWare จะถูกติดตั้งในเครื่องคอมพิวเตอร์เครื่องใดเครื่องหนึ่งในระบบ Network หลังจากที่ติดตั้งเรียบร้อยแล้ว คอมพิวเตอร์เครื่องนั้นจะถูกเรียกว่า "เซอร์เวอร์" (Server)โดยทำหน้าที่บริการแก่เครื่องคอมพิวเตอร์ลูกข่าย โดยให้บริการฮาร์ดดิสก์ และพรินเตอร์ เครื่องอื่นๆ ที่มาใช้บริการจะเรียกว่า "สถานีใช้งาน" (Workstation) โดยทำงานภายใต้ระบบปฏิบัติการต่างๆ เช่น DOS OS/2 เป็นต้น และจะเข้าไป ใช้อุปกรณ์ต่าง ๆ ของเซอร์เวอร์ได้ ต้องโหลด (Load) โปรแกรม "LAN Shell" ซึ่งเป็นโปรแกรมฝังตัวขึ้นมาก่อน

โปรแกรม NetWare ผู้ใช้สามารถกำหนดได้ว่า จะใช้งานร่วมกับการ์ดประเภทไหน ดังนั้น NetWare จึงใช้งานได้กับการ์ดหลายแบบและการเชื่อมต่อในแบบต่าง ๆ กันและสามารถใช้งานได้กับผลิตภัณฑ์ของผู้ผลิตต่าง ๆ กัน โดยผู้ใช้จะต้องทำการระบุขั้นตอนที่ทำการติดตั้งโปรแกรม NetWare ร่วมกับฮาร์ดแวร์ต่างชนิดกัน อาจมีผลทำให้ความเร็วในการทำงานต่างกันบ้าง ขึ้นอยู่กับความเร็วในการส่งข้อมูลประเภทนั้น และไดรเวอร์ (Driver) ของ NetWare ที่ใช้งานร่วมกับการ์ดนั้นๆ ซึ่งความเร็วของการใช้งานจะยิ่งเห็นได้ชัดเจนเมื่อมีจำนวนสถานีใช้งานเพิ่มมากขึ้นในระบบ

3.1 ประเภทต่าง ๆ ของ NetWare

แบ่งเป็น 4 ระดับ คือ

1. ELS NetWare ELS ย่อมาจากคำว่า "Entry Level System" ซึ่งเป็นเวอร์ชัน (Version) ที่เล็กที่สุดของ NetWare มี 2 ระดับ คือ ใช้งาน 4 คน กับ ใช้งาน 8 คน

2. Advanced NetWare 286 เป็น NetWare มาตรฐานที่ใช้เซอร์เวอร์ที่มีหน่วยประมวลผลเป็น โปรเซสเซอร์ 80286

3. SFT NetWare การทำงานคล้ายกับ Advanced NetWare 286 แต่สามารถทำงานในลักษณะ "System Fault Tolerant" ได้

4. NetWare 386 มีประสิทธิภาพสูงสุด โดยใช้เครื่องคอมพิวเตอร์ที่มีหน่วยประมวลผลโปรเซสเซอร์ 80386 หรือ 80486 เป็นเซอร์เวอร์

3.2 ข้อจำกัดของ NetWare

การทำงาน	Netware 2.2	Netware 3.11
จำนวนผู้ใช้ต่อเซอร์เวอร์	5,10,50,100	20,100,250
หน่วยความจำหลักของเซอร์เวอร์ที่ใช้น้อยที่สุด	2.5 MB	4 MB
หน่วยความจำหลักของเซอร์เวอร์ที่ใช้มากที่สุด	12 MB	4 MB
ขนาดของระบบปฏิบัติการในฮาร์ดดิสค์	5 MB	9 MB
ขนาดของฮาร์ดดิสค์มากที่สุด	2 GB	2048 GB
จำนวน VOLUMES ต่อ เซอร์เวอร์	32	64
จำนวนดิสค์ไดรฟ์ ต่อ เซอร์เวอร์	32	1024
จำนวน VOLUMES สูงสุด	255 MB	2048 MB
ขนาดไฟล์สูงสุด	255 MB	2048 MB
ใช้ไฟล์ได้พร้อมกันสูงสุด(ต่อเซอร์เวอร์)	1,000	100,000

รูปที่ 3.1 ตารางแสดงข้อจำกัดของ Netware

3.3 ส่วนประกอบของระบบเครือข่าย NetWare

ระบบเครือข่าย NetWare จะประกอบด้วยศูนย์บริการข้อมูล สถานีผู้ใช้และอุปกรณ์ อื่นๆ ที่ต่อเชื่อมกับศูนย์บริการข้อมูล

ศูนย์บริการข้อมูล เป็นหัวใจของระบบเครือข่าย NetWare ทำหน้าที่จัดการเกี่ยวกับการใช้ไฟล์ร่วมกัน ระบบรักษาความปลอดภัยของข้อมูล การทำงานติดต่อกันระหว่างสถานีการใช้ดิสก์ร่วมกัน ผู้ใช้ระบบเครือข่ายอาจตรวจสอบและควบคุมการทำงานของศูนย์บริการข้อมูล ได้จากจอภาพของศูนย์บริการข้อมูล ผู้ใช้สามารถเริ่มและหยุดศูนย์บริการข้อมูล กำหนดเวลาการใช้งาน ควบคุมการทำงานอื่น ๆ ได้

เครื่องพิมพ์ของระบบเครือข่าย ศูนย์บริการข้อมูลแต่ละตัวจะต่อเครื่องพิมพ์ได้ไม่เกิน 5 ตัวและจะเป็นเครื่องพิมพ์ของระบบผู้ใช้ใด ๆ สามารถใช้งานได้โดยเรียกผ่านศูนย์บริการข้อมูล

ดิสก์ของระบบเครือข่าย ศูนย์บริการแต่ละตัวอาจมีฮาร์ดดิสก์ ตั้งแต่ 1 ตัว หรือ มากกว่าก็ได้ เรียกฮาร์ดดิสก์ ชนิดนี้ว่า Network Disks

ดิสก์ของระบบจะแบ่งเป็น Volume, Directory, File

- Volume Server แต่ละตัวจะต้องมีที่เก็บข้อมูลอย่างน้อย 1 Volume คือ SYS :VOLUME ซึ่งถูกสร้างขึ้นเมื่อทำการติดตั้งเซอร์เวอร์

- ไดเร็กทรอรี (Directory) แต่ละโวลุ่ม (Volume) จะแบ่งเป็นหน่วยทางลอจิก (Logic) ที่เรียกว่า "ไดเร็กทรอรี" ในไดเร็กทรอรีหนึ่งๆอาจประกอบด้วยไดเร็กทรอรีย่อยอีกก็ได้ แต่ละระดับของ ไดเร็กทรอรีในโวลุ่มของระบบเครือข่ายจะประกอบเป็นโครงสร้างตามลำดับขั้น หรือ "Hierarchical Directory Structure" ผู้ติดตั้งระบบจะเป็นผู้กำหนดจำนวนสูงสุดของไดเร็กทรอรีที่มีในแต่ละโวลุ่ม ในระหว่างการติดตั้งระบบและหลังจากการติดตั้งระบบแล้ว ผู้ควบคุมระบบจะเป็นคนกำหนดสิทธิในการใช้ไดเรกทรอรีให้กับผู้ใช้แต่ละคน

- ไฟล์ เป็นชุดข้อมูลที่ถูกเก็บไว้รวมกัน

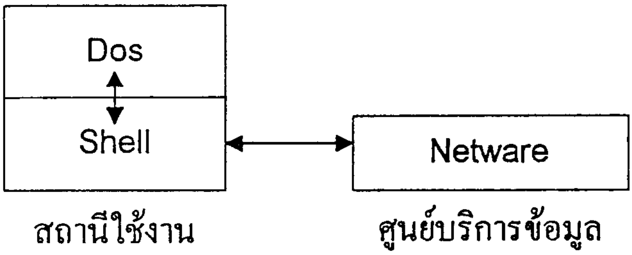
สถานีของผู้ใช้ ผู้ใช้แต่ละคนสามารถสร้างเก็บและเรียกใช้งานไฟล์ที่อยู่ในเซอร์เวอร์ ได้จากสถานีของผู้ใช้จะประกอบของ จอภาพ แป้นพิมพ์ ดิสก์ที่สถานีผู้ใช้ และหน่วยประมวลผล (CPU) ติดต่อกับระบบปฏิบัติการของเซอร์เวอร์ โดย Network Shell

ดิสก์ประจำสถานีผู้ใช้ ที่สถานีของผู้ใช้งานจะประกอบด้วย Local Disk ตั้งแต่ 1 ตัวขึ้นในซึ่งดิสก์เหล่านี้ไม่ได้ใช้สำหรับการชี้ไปยังไดรฟ์ของระบบเครือข่ายหรือไดเร็กทรอรี ของระบบเครือข่าย

ไดรฟ์ของระบบเครือข่าย ในระบบเครือข่ายสามารถกำหนดอักษรหมายเลขไดรฟ์ ให้ชี้ไปยังไดเร็กทรอรี ของระบบเครือข่ายได้ ซึ่งไดรฟ์ดังกล่าวจะเรียกว่า " Network Drive " ระบบเครือข่าย NetWare แต่ละสถานีสามารถกำหนดอักษรกำกับไดร์ฟได้ 26 ไดรฟ์

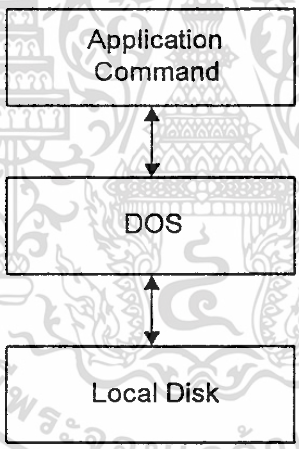
3.4 การทำงานของ NetWare

ระบบปฏิบัติการ NetWare จะสนับสนุนการใช้งานเครื่องไมโครคอมพิวเตอร์ที่ต่างกันหรือมีระบบปฏิบัติการที่ต่างกัน โดยไม่ต้องมีการแบ่งเนื้อที่บนดิสก์ ไฟล์ทั้งหมดและสถานีและสถานีผู้ใช้ เพื่อให้ DOS ติดต่อกับ NetWare ได้



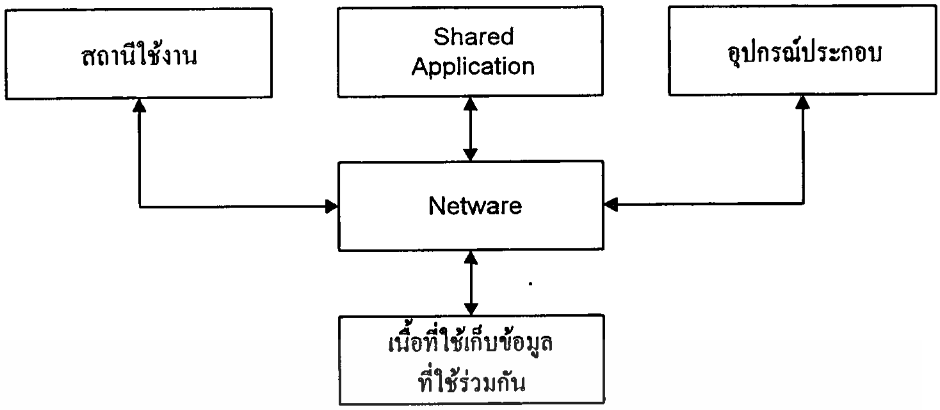
รูปที่ 3.2 Shell ของ NetWare ทำหน้าที่ติดต่อระหว่างดอสของสถานีผู้ใช้งานกับเซอร์เวอร์

DOS,NetWare และ Shell จะทำงานร่วมกันเพื่อช่วยในการใช้งานข้อมูล และฮาร์ดแวร์ร่วมกัน ดอส ออกแบบให้ประมวลผลโปรแกรมที่สถานีผู้ใช้ในสภาพแวดล้อมแบบผู้ใช้คนเดียว



รูปที่ 3.3 การทำงานของดอส (สภาวะแวดล้อมแบบมีผู้ใช้คนเดียว)

NetWare ออกแบบมาให้ประมวลผลในสภาพแวดล้อมที่มีผู้ใช้หลายคนซึ่งจะช่วยในการใช้งานอุปกรณ์ต่างๆร่วมกัน ใช้งานโปรแกรมร่วมกัน ควบคุมไดเร็กทรอรีที่ซับซ้อน และ file allocation tables(FAT) ที่เซอร์เวอร์



รูปที่ 3.4 การทำงานของ NetWare (สภาวะแวดล้อมแบบมีผู้ใช้หลายคน)

การรักษาความปลอดภัยและการจัดการไฟล์ที่ ดอสไม่ได้จัดการให้ จะถูกจัดการโดย NetWare ที่เซอร์เวอร์ โดย NetWare กำหนดบุคคลที่มีคุณสมบัติดังนี้

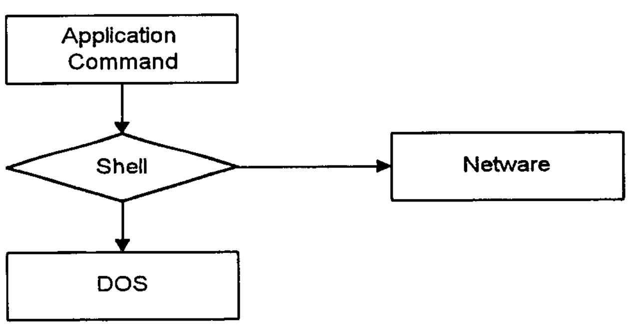
- ผู้ที่สามารถใช้งานเซอร์เวอร์ได้
- ผู้ที่สามารถใช้งานไฟล์ได้
- ผู้ที่จะสามารถใช้งานไฟล์ร่วมกันได้

NetWare นั้นไม่ใช้ดอส ในสภาพแวดล้อม ของระบบเครือข่าย แต่ดอสจะถูกใช้ที่สถานีผู้ใช้เท่านั้น เพื่อทำงานบนไดรฟ์ที่เป็นของตัวเอง

NetWare ควบคุมการสร้าง และจัดการบนเซอร์เวอร์ ในขณะที่ Network Shell จะช่วยให้ ดอส และ NetWare ติดต่อกันได้

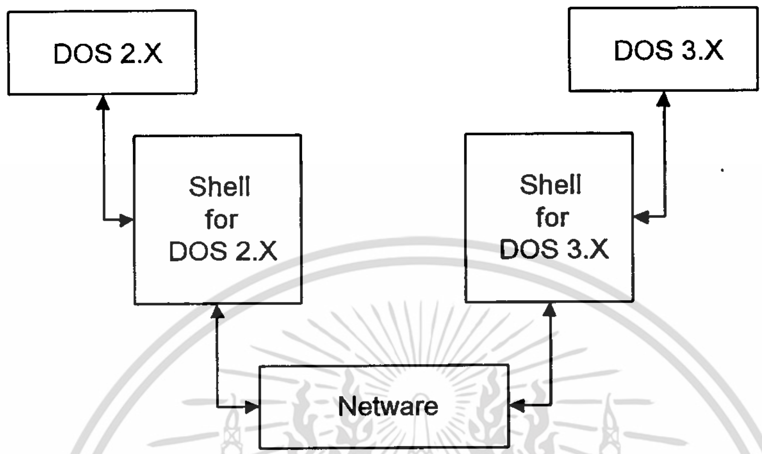
NetWare Shell ซึ่งมีขนาดประมาณ 40 Kbytes จะอยู่ที่สถานีของผู้ใช้ และดักการเรียกใช้อินเตอร์รัพท์ 21H (การขอข้อมูลหรือคำสั่งของ NetWare ที่อยู่บนเซอร์เวอร์)

ดอส จะเปรียบเทียบได้กับ ผู้ที่พูดและเข้าใจ เฉพาะภาษาอังกฤษ NetWare เป็นผู้ที่พูดและเข้าใจเฉพาะภาษาไทย shell จะเป็นล่ามระหว่าง ดอส และ NetWare



รูปที่ 3.5 การทำงานของ Shell ภายใต้ NetWare

Shell จะทำให้ NetWare ต่างไปจากระบบปฏิบัติการของระบบเครือข่ายอื่นๆ เพราะไม่จำกัดว่าสถานีของผู้ใช้ดอส รุ่นไหน NetWare จะสนับสนุนดอส ตั้งแต่ 2.xx ขึ้นไป



รูปที่ 3.6 การสนับสนุนการใช้ดอสภายใต้ NetWare

3.5 การวิเคราะห์ NetWare เปรียบเทียบกับ OSI Model

Novell's NetWare คือ Network Operating System (NOS) มีพื้นฐานจาก Xerox Network System (XNS) Protocol ซึ่งพัฒนาโดย Xerox Corporation Palo Alto Research Center XNS มีลักษณะสถาปัตยกรรมที่เหมาะสมสำหรับการจัดฟังก์ชันในแต่ละ Layer XNS ใช้วิธีตรงๆในการ Addressing ใน Network และ Internetwork XNS จึงเหมาะที่จะนำมาประยุกต์ใช้กับ LAN โดยบริษัทอื่นที่ผลิตผลิตภัณฑ์เกี่ยวกับ LAN เช่น 3com ใช้ XNS เป็นพื้นฐานด้วยเช่นกัน

ข้อดีของ NetWare ที่เป็นที่ยอมรับนิยมกับผู้ใช้คือ มีหลายเวอร์ชัน ให้เลือกเช่น EIS (Entry Level System) Level 1,2 ได้ถูกออกแบบสำหรับผู้ใช้จำนวนน้อยๆ ประมาณ 4 ถึง 8 workstations Network ที่มีขนาดใหญ่ขึ้นอาจใช้ NetWare เวอร์ชัน 2.15 ซึ่ง support ถึง 100 สถานีผู้ใช้ และสามารถติดต่อกับ Netware ที่ใช้กับ Macintosh และ NetWare ที่ใช้กับ OS/2 Workstation NetWare 386 เวอร์ชัน 3.0 ถูกออกแบบมาเพื่อใช้กับ Microprocessor 80386 และสามารถใช้กับ 250 สถานีผู้ใช้

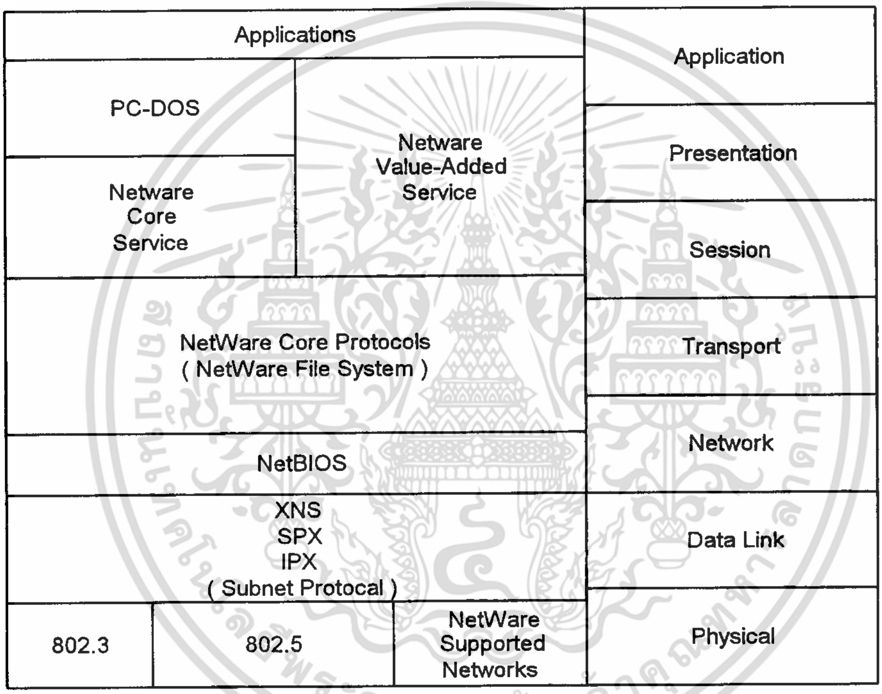
โครงสร้างของ NetWare ได้ถูกออกแบบมาอิงกับมาตรฐาน OSI Model โดยสามารถเปรียบเทียบได้ดังนี้

-Physical and Data Link Layers NetWare จะอิงกับ Ethernet,IEEE 802.5 เช่นเดียวกับ NOS ของบริษัทอื่นเช่น ARCNET

-Network and Transport Layers ซึ่งจะทำงานโดยใช้ Internetwork Packet Exchange (IPX) และ Sequenced Packet Exchange (SPX) Protocols ทั้งสองนี้เป็นส่วนหนึ่งของ XNS

-Session Layer จะใช้ Network Basic Input Output (NetBIOS) interface ซึ่งถูกพัฒนาโดย IBM และ Sytek

-Presentation and Application Layers จะประกอบด้วย NetWare Core Protocols (NCP) , NetWare Core and Value-Added Service , Dos และ User Applications



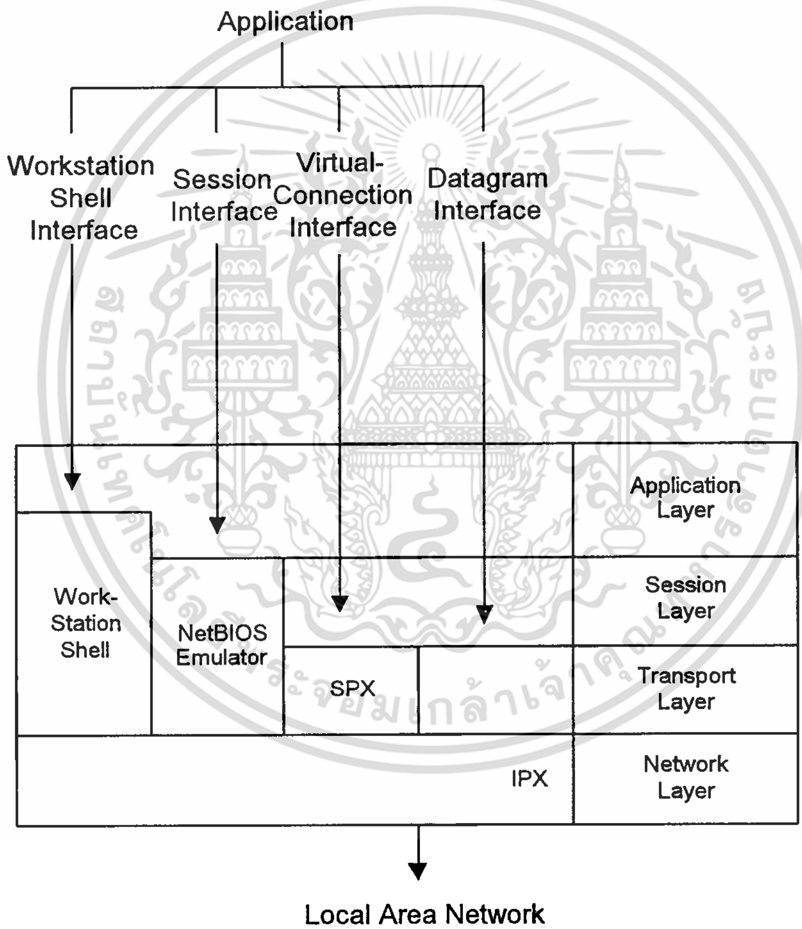
รูปที่ 3.7 ความสัมพันธ์ระหว่างโครงสร้างของ NetWare กับ OSI Model

โปรแกรม Application ต่างๆสามารถใช้ NetWare protocols ได้ 4 วิธี

- การติดต่อกับ Network Layer คือ IPX ซึ่งเป็นการติดต่อแบบ Connectionless (Datagram) ระหว่างสถานีผู้ใช้ซึ่ง Packet ที่ถูกส่งจะไม่รับประกันการได้รับของฝ่ายรับ

- การติดต่อกับ Transport Layer คือ SPX ซึ่งเป็นการติดต่อแบบ Connection-Oriented ระหว่างสถานีผู้ใช้ ซึ่งเส้นทางติดต่อต้องสร้างขึ้นก่อนการส่งข้อมูล โดย SPX จะรับ การติดต่อกับ Session Layer คือใช้ NetBios emulator ดังนั้นโปรแกรมประยุกต์ ซึ่งใช้โปรโตคอล NetBios สามารถใช้งานบน NetWare ได้เช่นกัน

- การติดต่อกับ Application Layer ทำโดยผ่าน Workstation Shell โดยทำการติดต่อผ่านฟังก์ชันของ DOS โดย Shell จะทำหน้าที่ติดต่อระหว่าง DOS กับ IPX Protocol ซึ่งใช้ในการรับส่งข้อมูลในระบบเครือข่าย โปรแกรมประยุกต์ จะใช้การติดต่อในชั้นนี้ เพื่อติดต่อระหว่างสถานีผู้ใช้ กับ เซอร์เวอร์ เพื่อขอใช้ฟังก์ชันต่างๆในระบบเครือข่าย เช่น การขอใช้ไฟล์ข้อมูลในสถานีผู้ใช้กับ เซอร์เวอร์ เพื่อขอใช้ฟังก์ชันต่างๆ ใน Network เช่น การขอใช้ไฟล์ข้อมูลในเซอร์เวอร์ หรือการพิมพ์ผ่าน พรินเตอร์ ของระบบ การติดต่อในชั้นนี้เรียกว่า "Workstation Shell Interface"



รูปที่ 3.8 วิธี interface กับ NetWare

บทที่ 4

IPX and SPX Protocal

4.1 โครงสร้างกลุ่มข้อมูล (Packet)

NetWare IPX และ SPX เป็นรูปแบบโปรโตคอลพื้นฐาน ของการสื่อสารเครือข่าย IPX เป็นแบบ Connectionless และ SPX เป็นแบบ Connection-oriented ในบทนี้จะกล่าวถึงข้อดีข้อเสีย และรูปแบบกลุ่มข้อมูล ของแต่ละแบบไว้ด้วย

IPX โปรโตคอล

Netware IPX เป็นแบบ Connectionless หรือ Datagram โปรโตคอล ความหมายของคำว่า Connectionless คือ เมื่อการประยุกต์ใช้งานของ สถานีใช้งาน (Workstation) ใช้ IPX ในการสื่อสารกับสถานีใช้งานอื่น จะไม่มีการติดต่อหรือไม่มีเส้นทางระหว่างสองสถานีเกิดขึ้น ดังนั้นกลุ่มข้อมูล IPX จะประกอบด้วย ข้อมูล ซึ่งเป็นแอดเดรส (Address) และจะถูกส่งไปยังปลายทาง แต่จะไม่รับประกันว่า การส่งจะสำเร็จหรือไม่ ส่วน Datagram หมายถึง กลุ่มข้อมูลแต่ละอัน จะถูกแบ่งเป็นกลุ่มข้อมูลเฉพาะซึ่งไม่มีความสำคัญเป็นลำดับกับกลุ่มข้อมูลอื่นเลย IPX จะทำหน้าที่ในส่วนของ ชั้นเน็ตเวิร์ก (Network Layer) ของรูปแบบ OSI หน้าที่นี้รวมถึง การกำหนดแอดเดรสที่อยู่ การกำหนดเส้นทาง และการสับเปลี่ยนกลุ่มข้อมูล เพื่อที่จะส่งข้อมูลใด ๆ จากตำแหน่งหนึ่งไปสู่อีกตำแหน่งหนึ่งในเครือข่าย

ข้อได้เปรียบและเสียเปรียบ

เมื่อ IPX ทำหน้าที่เพียงในส่วนของชั้นเน็ตเวิร์ก จะทำให้เกิดผลดีในเรื่องความเร็ว และการแสดงผลอย่างไรก็ตามการบริการ IPX ยังไม่มีประสิทธิภาพเพียงพอถ้าต้องการความมั่นใจในการส่งข้องมูลของชั้นการส่ง (Transport Layer) ผู้ที่จะนำไปใช้จะต้องพิจารณาถึงความเหมาะสมในด้านข้อดีข้อเสียของIPX เพื่อที่จะเลือกใช้IPXหรือ SPX ได้อย่างถูกต้อง

- การใช้งาน ในเวลาเดียวกันของ ผู้ส่งและผู้รับไม่มีความจำเป็น เพราะจะไม่มีการติดต่อล่วงหน้าในการส่ง อย่างไรก็ตาม ผู้ส่งจะไม่มั่นใจว่าการรับส่งข้อมูลจะถูกต้อง
- การกำหนดเส้นทางของกลุ่มข้อมูลจะ ทำได้มาก หลายเส้นทาง เพราะไม่มีการกำหนดเส้นทางตายตัวเอาไว้ ผลเสียคือ ข้อมูลจะได้รับไม่เป็นลำดับ
- ข้อมูลอาจส่งไปถึงจุดหมายได้หลายที่โดยง่าย ด้วยการทำเลียนแบบข้อมูลแบะเปลี่ยนที่แอดเดรสปลายทาง เท่านั้น

การส่งกลุ่มข้อมูล IPX

ข้อมูลที่จะถูกส่งจะถูกรวบรวมและใส่ลงในส่วนของข้อมูล ในกลุ่มข้อมูล IPX เช่นเดียวกับการส่งจดหมายลงในซอง ส่วนหัวของกลุ่มข้อมูลจะรวม เครือข่ายปลายทาง, จุดต่อ (Node) และหมายเลขซอคเก็ต (Socket Number) กลุ่มข้อมูลจะถูกส่งไป IPX จะแยกส่งกลุ่มข้อมูลไปสู่เครือข่ายย่อย (ผ่านเส้นทางที่แตกต่างกันเพื่อให้เครือข่ายไม่หนาแน่น) กระทั่งกลุ่มข้อมูลถึงจุดหมาย ดังนั้น เส้นทางต่าง ๆ ของลำดับการส่งกลุ่มข้อมูลจะต้องแตกต่างกันไป

เมื่อได้รับกลุ่มข้อมูลแล้ว ฝ่ายผู้ส่งจะไม่ได้รับการยืนยันของการได้รับข้อมูล มีเพียงแต่ จุดหมายปลายทางจะส่งข้อมูล ร้องขอกลับมา (Request) แต่ เชื่อได้ว่าจะประสบความสำเร็จ 95 %

โครงสร้างกลุ่มข้อมูล IPX

กลุ่มข้อมูล IPX จะคล้ายกับ กลุ่มข้อมูล Xerox IDP จะประกอบด้วยสองส่วน คือ ส่วนหัว จำนวน 30 ไบต์ และ ส่วนข้อมูล 0-546 ไบต์ ความยาวกลุ่มข้อมูลน้อยสุด 30 ไบต์ (ส่วนหัว) และความยาวสูงสุด 576 ไบต์

โครงสร้างกลุ่มข้อมูล IPX จะมีส่วนต่าง ๆ และคำอธิบายดังต่อไปนี้

Offset	Content	Type
0	Checksum	BYTE[2]
2	Length	BYTE[2]
4	Transport Control	BYTE
5	Packet Type	BYTE
6	Destination Network	BYTE[4]
10	Destination Node	BYTE[6]
16	Destination Socket	BYTE[2]
18	Source Network	BYTE[4]
22	Source Node	BYTE[6]
28	Source Socket	BYTE[2]
30	Data portion	byte[0 to 546]

ทุกส่วนย่อย เป็นสูง - ต่ำ ซึ่งหมายความถึงไบต์ที่มีนัยสำคัญสูงสุดในส่วน จะถูกส่งเป็นไบต์แรก ซึ่งแตกต่างจากรูปแบบทั่วไป ของ อินเทลโปรเซสเซอร์ (Intel Processor) ซึ่งไบต์ที่มีนัยสำคัญต่ำสุดจะถูกส่งก่อน (ต่ำ-สูง)

Checksum

ในส่วนนี้จะถูกตั้งค่า 0xFFFF โดย IPX ซึ่งจะทำให้ การ์ด LAN ทำหน้าที่เป็นส่วนของฮาร์ดแวร์ Checksum เอง

ความยาว (Length)

ส่วนนี้บอกความยาวของข้อมูลกลุ่มข้อมูลรวม ส่วนหัวและส่วนข้อมูล ซึ่งความยาวต่ำสุดเท่ากับ 30 ไบต์ และสูงสุด เท่ากับ IPX จะกำหนดค่านี้เอง

Transport Control

ส่วนนี้จะใช้ในการติดต่อข้ามเครือข่าย IPX จะกำหนดให้มีค่าเป็น 0 เสมอ ก่อนส่งกลุ่มข้อมูล

ชนิดของกลุ่มข้อมูล (Packet Type)

เป็นชนิดของการบริการชนิดใด ซึ่งจะถูกร้องขอโดยกลุ่มข้อมูล จะมีค่าต่าง ๆ ดังนี้ ผู้ใช้ IPX ควรจะ กำหนด ให้มีค่า 0 หรือ 4 เสมอ ส่วน SPX จะกำหนดเป็น 5

0 ไม่กำหนดชนิดกลุ่มข้อมูล

1 กลุ่มข้อมูลของข้อมูลเส้นทาง

2 กลุ่มข้อมูล Echo

3 กลุ่มข้อมูลผิดพลาด

4 กลุ่มข้อมูลของการแลกเปลี่ยนกลุ่มข้อมูล

5 กลุ่มข้อมูลของโปรโตคอลกลุ่มข้อมูลมีกลุ่มข้อมูลเป็นลำดับ (Sequenced Packet)

16-31 โปรโตคอลทดลอง

17 โปโตคอลแกนกลางเน็ตเวิร์ก (NetWare Core Protocols)

เครือข่ายปลายทาง (Destination Network)

ส่วนนี้จะมีหมายเลขเครือข่าย ซึ่งเป็นของเครือข่ายที่มีจุดต่อปลายทาง (Destination Node) อยู่ ภายในนั้น ใน Netware ผู้จัดการเครือข่าย จะกำหนดค่า 4 ไบต์ นี้ขึ้นมา แต่เมื่อตั้งค่านี้เป็น 0 Netware สันนิษฐานว่า จุดต่อปลายทางอยู่ในเครือข่ายเดียวกันกับ แหล่งต้นทาง ซึ่งไม่ถูกกำหนดโดย Netware

จุดต่อปลายทาง

จะเป็นส่วนของแอดเดรสของจุดต่อปลายทาง ซึ่งรูปแบบการติดต่อของ LAN ไม่ได้ใช้รูปแบบขนาดที่อยู่เดียวกันหมด จุดต่อของเครือข่าย อีเทอร์เน็ต (Ethernet) ต้องการ 6 ไบต์ ของการกำหนดแอดเดรสในขณะ ที่เครือข่าย โอมนิเนท (Omninet) ต้องการเพียง 1 ไบต์ ถ้าเครือข่ายต้องการจำนวนแอดเดรสน้อยกว่า 6 ไบต์ ที่อยู่ควรจะ อยู่ในส่วนของเลขนัยสำคัญน้อยสุด และ เลขนัย

สำคัญสูงสุดควรมีค่าเป็น 0 ที่อยู่ของจุดต่อปลายทางที่มีค่า 0xFF FF FF FF FF FF กลุ่มข้อมูลจะถูกส่งไปยัง จุดต่อปลายทางทุกจุดต่อ ภายในเครือข่ายปลายทางนั้น

Destination Socket

ส่วนนี้จะบรรจุหมายเลขซอคเก็ตปลายทางของกลุ่มข้อมูล ในการส่งข้อมูลไปซอคเก็ตจะกำหนด กลุ่มข้อมูลให้ถูกประมวลผลรูปแบบต่าง ๆ ภายใน จุดต่อ Xerox ได้กำหนดหมายเลขซอคเก็ต ดังต่อไปนี้

1	กลุ่มข้อมูลที่มีข้อมูลของเส้นทาง
2	กลุ่มข้อมูลของ Echo Protocol
3	กลุ่มข้อมูลของการจัดการความผิดพลาด
0x20-0x3F	ใช้ในการทดลอง
0x1-0xBB8	การจัดการของ Xerox
0xBB 9-	สามารถกำหนดการทำงานได้

Xerox ได้กำหนดค่าสำหรับใช้ใน Netware ดังนี้

451	กลุ่มข้อมูลของการบริการแฟ้ม (File Service)
452	กลุ่มข้อมูลของการประกาศบริการ (Service Advertising)
453	กลุ่มข้อมูลของข้อมูลเส้นทาง
455	กลุ่มข้อมูลของเนทไบออส (NetBIOS)
456	กลุ่มข้อมูลของการตรวจสอบวินิจฉัย (Diagnostic Packet)

ตัวอย่าง เช่น เครื่องบริการแฟ้มของ Netware ได้รับคำร้องขอหมายเลขซอคเก็ต 0x0451 Novell จะจัดการกลุ่มของซอคเก็ตให้เป็นที่รู้กันในระบบ Netware หมายเลขซอคเก็ตแบบพลวัตเริ่มที่ 0x4000 หมายเลขซอคเก็ตที่แสดงให้รู้กันโดย Novell หมายเลขที่มากกว่า 0x8000 จะไม่ถูกนำมาใช้ ถ้า ไม่มีการจัดการสำหรับการประยุกต์นั้น ๆ โดย Novell

เครือข่ายต้นทาง (Source Network)

ในส่วนนี้จะบรรจุหมายเลขของเครือข่าย ซึ่งจุดต่อต้นทางต่ออยู่ภายใน เครือข่าย ของ Netware จะกำหนดให้ใช้ หมายเลขเครือข่าย 4 ไบต์ IPX จะกำหนดค่าในส่วนนี้ ให้เป็นหมายเลขเครือข่าย ซึ่งครอบครองอยู่ ถ้ามีค่าเป็น 0 หมายความว่า เครือข่าย ซึ่งต้นทางต่ออยู่ ไม่ทราบค่า

จุดต่อต้นทาง (Source Node)

IPX จะกำหนดค่านี้ ตามที่อยู่ของ จุดต่อต้นทาง (source node)

ซอคเก็ตต้นทาง (Source Socket)

IPX จะกำหนดหมายเลขซอคเก็ตของกระบานการ ในการส่งกลุ่มข้อมูล กระบานการสื่อสารจากจุด สถานีงานไปสู่สถานีงานในการรับส่งไม่อยู่บนหมายเลขซอคเก็ตเดียวกัน ในการติดต่อของลูกข่ายเครื่องบริการแฟ้มจะรออยู่ที่ซอคเก็ตเฉพาะ สำหรับการร้องขอในการบริการ เช่น ซอคเก็ตต้นทางไม่จำเป็นต้องมีหมายเลขเหมือนกัน ถ้าเซอร์เวอร์ยังสามารถ ต่อซอคเก็ตต้นทางได้ใน เครื่องบริการแฟ้มของ Netware จะใช้ที่อยู่ของซอคเก็ตเดียวกันแต่การร้องขออาจมาจากหมายเลขซอคเก็ตที่แตกต่างกันไปในกรณีของซอคเก็ตปลายทาง อาจจะเป็นแบบสถิต หรือแบบพลวัตก็ได้

SPX โปรโตคอล (SPX Protocol)

จะคล้ายกับ IPX ยกเว้นในส่วนหัวที่เพิ่มเติมขึ้นมาของชั้นการส่ง (Transport Layer) ใน OSI หน้าที่ที่เพิ่มเติมขึ้นมาจะทำให้ SPX มีการติดต่อ แบบ connection - Oriented หมายความว่าก่อนการส่ง กลุ่มข้อมูล SPX การติดต่อหรือเส้นทางระหว่างผู้ส่งหรือผู้รับจะถูกสร้างขึ้นมา และ ยังรับประกันในการส่ง กลุ่มข้อมูลจะถูกส่งตามลำดับ ตรวจสอบและแก้ไขข้อผิดพลาด การทำซ้ำย่อของข้อมูลกลุ่มข้อมูล (Suppressing Packet)

ข้อได้เปรียบและเสียเปรียบ

ในการรับประกันการส่งข้อมูลทำให้ SPX มีความเร็วและการแสดงผลได้ไม่ดีเท่า IPX ดังนั้น ผู้พัฒนาข้อมูล ควรตัดสินใจในการเลือกที่จะใช้ ความเร็วและการแสดงผล IPX หรือการรับประกันการติดต่อด้วย SPX อย่างไรจะสำคัญ กว่ากันในแต่ละกรณี ซึ่ง SPX จะมีข้อดีอย่างย่อดังต่อไปนี้

- การรับประกันการส่ง จะมีการต่อเส้นทางเกิดขึ้นก่อนที่ข้อมูลจะถูกส่ง และมีสัญญาณในการรับข้อมูลแล้วส่งกลับมายังผู้ส่งแต่ จะมีข้อเสีย ถ้าหาก ผู้รับไม่ว่าง กลุ่มข้อมูล จะไม่สามารถส่งได้ การส่งกระจายข้อมูลไปยังสถานี หลายสถานีทำได้ยากเนื่องจากต้องสร้างเส้นทางการติดต่อ ก่อนส่งและ บางครั้ง การประยุกต์ ไม่จำเป็นต้องมีการรับประกันการส่งสำหรับกลุ่มข้อมูลทุกครั้งอีกด้วย
- รับประกันการเป็นลำดับของข้อมูล เชื่อได้ว่า ข้อมูลที่ได้รับจะเป็นลำดับกัน โดยไม่สนใจว่า ข้อความต้องการส่งกี่กลุ่มข้อมูล ก็จะถูกส่งอย่างเหมาะสมตามลำดับ
- การบีบอัดข้อมูลซ้ำของกลุ่มข้อมูล ระหว่างกระบวนการการยืนยันการส่งข้อมูล เป็นไปได้ที่จะมีข้อมูล ซ้ำได้รับที่ฝ่ายผู้รับ SPX จะเป็นตัวกำจัดกลุ่มข้อมูล นี้ไป เพื่อที่ฝ่ายรับจะได้รับเพียงข้อมูลเดียวจากคู่สายการติดต่อ

โครงสร้างของกลุ่มข้อมูล SPX

กลุ่มข้อมูล SPX จะเหมือนกับกลุ่มข้อมูล IPX เว้นแต่ในส่วนหัวจะมี 12 ไบต์ เพิ่มเข้ามา 1 กลุ่มข้อมูล จะประกอบด้วย 2 ส่วน ส่วนหัว จำนวน 42 ไบต์ และส่วนข้อมูล 0-534 ไบต์ ความยาวกลุ่มข้อมูล ต่ำสุด 42 ไบต์ และความยาวสูงสุด 576 ไบต์

ส่วนของกลุ่มข้อมูล SPX จาก Checksum จนถึง ซอคเก็ตต้นทาง จะมีความหมายเหมือนเดิมเช่นเดียวกับ กลุ่มข้อมูล IPX โดยมี ข้อแตกต่างดังต่อไปนี้

- ชนิดของกลุ่มข้อมูล จะถูกกำหนดค่าเป็น 5 โดย SPX
- จุดต่อปลายทางจะไม่บรรจุที่อยู่การส่งกระจายข้อมูล (Broadcast Address) การติดต่อจะเกิดจาก SPX โดย คู่ติดต่อสื่อสารอื่น การส่งกระจายข้อมูลหรือการส่งข้อมูลหลายสถานี ไม่อนุญาตในการส่งข้อมูลแบบ SPX โครงสร้าง กลุ่มข้อมูล แบบ SPX จะอธิบาย ได้ดังนี้

Offset	เนื้อหา	ชนิดของคำ
0	Checksum	BYTE[2]
2	Length	BYTE[2]
4	Transport Control	BYTE
5	Packet Type	BYTE
6	Destination Network	BYTE[4]
10	Destination Node	BYTE[6]
16	Destination Socket	BYTE[2]
18	Source Network	BYTE[4]
22	Source Node	BYTE[6]
28	Source Socket	BYTE[2]
30	connect.Control	BYTE
31	Data Stream Type	BYTE
32	Source Connect.ID	BYTE[2]
34	Dest.Connect.ID	BYTE[2]
36	Sequence Number	BYTE[2]
38	Acknowledge Num	BYTE[2]
40	Allocation Num.	BYTE[2]
42	Data Portion	BYTE[0-534]

ทุกส่วนเป็นแบบสูงต่ำไบต์นัยสำคัญสูงสุด คือตัวแรกซึ่งแตกต่างจาก รูปแบบทั่วไป ซึ่งเลขนัยสำคัญสูงสุดจะเป็นตัวสุดท้าย (ต่ำสูง)

การควบคุมการติดต่อ (Connection Control)

ในส่วนนี้ตัวบ่งชี้เดี่ยวจำนวน 4 ตัวใช้โดย SPX ในการควบคุมการไหลของข้อมูลสองทิศทางระหว่างการติดต่อ

0x1-0x8 SPX ไม่ใช้

0x10 จบข้อมูล (End-of-Message) ลูกข่ายหนดค่าตัวบ่งชี้ในการส่งสัญญาณ End-of-message ไปยังคู่ติดต่อ SPX ไม่สนใจค่าแต่ผ่านค่าไปยังคู่ติดต่อ ตัวบ่งชี้นี้ใช้ในการประยุกต์เพื่อส่งสัญญาณ สิ้นสุดข้อมูลหลายกลุ่มข้อมูลไปยังจุดต่อที่ไกลออกไป

0x20 Attention ลูกข่ายกำหนดค่าของกลุ่มข้อมูลระวัง (Attention) SPX ไม่สนใจและผ่านค่าไปยังคู่ติดต่อ

0x40 ต้องการการตอบรับ SPX จะกำหนดค่านี้ถ้าต้องการกลุ่มข้อมูลตอบรับโดยที่ลูกข่ายไม่ต้องจัดการ

0x80 System Packet SPX กำหนดค่าและค่าบ่งชี้นี้จะถูกใช้ภายในโดยไม่ได้ส่งให้คู่ติดต่อ

ลูกข่ายไม่ควรเปลี่ยนแปลงหรือใช้ตัวบ่งชี้ที่ไม่ใช้,ตัวบ่งชี้การตอบรับหรือ system packet ตัวบ่งชี้เหล่านี้สงวนไว้

ชนิดของกระแสข้อมูล (Datastream Type)

ส่วนนี้จะเป็นข้อมูลตัวบ่งชี้ 1ไบต์ ในการระบุชนิดของข้อมูลในกลุ่มข้อมูล จะมีค่าและความหมายดังต่อไปนี้

0x0-0xFD ลูกข่ายระบุ SPX ไม่ใช้

0xFE จบการติดต่อ (End-of-Connection) เมื่อลูกข่ายเรียกทำการยกเลิกการติดต่อ SPX จะทำการสร้างกลุ่มข้อมูล End-of-connection ส่งไปยังคู่ติดต่อเป็นกลุ่มข้อมูลสุดท้าย

0xFF รับรู้การจบการติดต่อSPX จะสร้างกลุ่มข้อมูลนี้อัตโนมัติเป็นกลุ่มข้อมูลระบบและจะไม่ถูกส่งไปยังคู่ติดต่อ

Source Connection ID

ในส่วนนี้จะมีหมายเลขประจำตัวในการติดต่อกำหนดโดย SPX ของจุดต้นทาง

Destination Connection ID

เหมือน Source Connection ID เพียงแต่เป็นของปลายทาง อาจจำเป็นต้องใช้การดีมัลติเพล็กซ์ (Demultiplex) กลุ่มข้อมูลที่เข้ามาจากการติดต่อหลายทางในซอคเก็ตเดียวกันด้วย

เลขลำดับ (Sequence Number)

ส่วนนี้จะเก็บจำนวนนับกลุ่มข้อมูลแลกเปลี่ยนทิศทางหนึ่งในการติดต่อ แต่ละด้านจะนับค่าเก็บไว้ ค่าจะกลับมาเริ่มที่ 0x0 หลังการนับ 0xFFFF โดย SPX จะจัดการในส่วนนี้เอง

เลขการตอบรับ (Acknowledge Number)

บอกค่าจำนวนลำดับของกลุ่มข้อมูลต่อไปที่จะได้รับ ถ้ากลุ่มข้อมูลใดมีค่าเลขลำดับน้อยกว่าค่านี้แสดงว่าลำดับการส่งถูกต้องและไม่ต้องทำการส่งซ้ำ SPX จัดการส่วนนี้

จำนวนหน่วยพักข้อมูล (Allocation Number)

ส่วนของจำนวนหน่วยพักข้อมูลฝ่ายรับในทิศทางหนึ่งๆของการติดต่อ โดยที่ SPX จะทำการส่งจนกระทั่งเลขลำดับมีค่าเท่ากับจำนวนหน่วยพักข้อมูล

4.2 Event Control Block (ECB)

ECB เป็นโครงสร้างข้อมูลที่ใช้ในการควบคุมการส่งหรือรับกลุ่มข้อมูลแบบ IPX หรือ SPX จะอ้างในรูปแบบ SendECB หรือ RecieveECB ซึ่งเป็นตัวเดียวกัน แตกต่างขึ้นอยู่กับการทำงาน ถ้าเป็น SendECB จะต้องการแอดเดรสของปลายทางในส่วนของ Immediate Address ซึ่ง RecieveECB ไม่ต้องใช้

โครงสร้าง ECB

ECB ประกอบด้วย 2 ส่วนที่สำคัญได้แก่ ส่วนความยาวที่จำกัดไว้ 36 ไบต์ และส่วนข้อมูลที่มีจำนวนขึ้นกับจำนวนหน่วยพักข้อมูล IPX หรือ SPX (Fragment) ที่ใช้ ECB จะมีส่วนต่าง ๆ และคำอธิบายดังนี้

Offset	Content	Type	Order
0	Link Address	BYTE[4]	offset- segment
4	ESR Address	BYTE[4]	offset-segment
8	In Use Flag	BYTE	
9	Completion Code	BYTE	
10	Socket Number	WORD	high-low
12	IPX Workspace	BYTE[4]	
16	Driver Workspace	BYTE[12]	
28	Immediate Address	BYTE[6]	
34	Fragment Count	WORD	low-high
36	Fragment Address1	BYTE[4]	offset-segment

40	Fragment Size1	BYTE[2]
42	Fragment Address2	BYTE[4]
46	Fragment Size2	BYTE[2]

แอดเดรสเชื่อมโยง (Link Address)

ในขณะที่มีการใช้ ECB อยู่ IPX จะครอบครองในส่วนนี้ แต่ถ้าไม่มีการใช้งาน ส่วนECB นี้จะสามารถใช้ในการประยุกต์ได้ (ถ้าจำเป็น)

ESR Address

ส่วนนี้จะมีเลขแอดเดรสของโปรแกรมประยุกต์ ESR ที่ IPX สามารถเรียกใช้เมื่อมีการรับหรือส่งข้อมูลเรียบร้อยแล้ว เนื่องจาก IPX ยังคงมีค่า In Use Flag และ Completion Code อยู่ โปรแกรมสามารถเรียกสถานะการร้องขอ IPX (ECB)ในช่วงเวลาที่เหมาะสม และไม่ใช้ ESRได้ ถ้า ESR ไม่ถูกใช้ ESR Address ควรกำหนดให้มีค่า 0x0000 (Null Pointer)

In Use Flag

เมื่อ IPX เรียกใช้ ECB ส่วนนี้จะถูกกำหนดให้มีค่า และเมื่อการร้องขอเสร็จ IPX จะตั้งค่า 0 ให้ ค่าที่ถูกกำหนดให้ ได้แก่

0xF8 การพยายามในการส่งขณะที่ IPX ไม่ว่าง กลุ่มข้อมูลที่ส่งและ ECB จะถูกเก็บเข้าแถวในการประมวลผลต่อไป

0xFA ECB กำลังถูกประมวลผลโดย IPX

0xFB เหตุการณ์ของการส่งหรือรับเกิดขึ้นแต่ ECB ยังคงถูกพักไว้เพื่อรอการประมวลผล

0xFC เหตุการณ์ถูกกำหนดการจัดการโดย AES และ AES กำลังรอให้หมดช่วงเวลา

0xFD เหตุการณ์ถูกกำหนดการจัดการขึ้นและ IPX กำลังรอให้หมดช่วงเวลา

0xFE IPX เฝ้ารอกลุ่มข้อมูลที่จะเข้ามาที่ซอคเก็ต

0xFF ECB ถูกใช้งานในการส่งกลุ่มข้อมูล

เมื่อการร้องขอเสร็จสมบูรณ์ IPX จะตั้งค่าในส่วนนี้เป็น 0 ใหม่

Completion Code

IPX จะกำหนดค่านี้เพื่อบอกสถานะสุดท้ายของการส่งหรือรับ ECB ส่วนนี้จะไม่ถูกพิจารณาว่ามีค่าจนกว่า IPXจะตั้งค่า In Use Flag ใน ECB เป็น 0 เสียก่อน

เมื่อ ECB ถูก IPX เรียกใช้ในการส่งกลุ่มข้อมูล Completion Code จะแสดงค่าใดๆ ได้ดังนี้

0x00 Successful ข้อมูลถูกส่งเสร็จสมบูรณ์

0xFC Canceled การร้องขอการส่งถูกยกเลิก

0xFD Malformed - กลุ่มข้อมูลผิดรูปแบบ ปัญหาที่อาจเกิดขึ้นคือ

- ความยาวกลุ่มข้อมูลรวมมีค่าน้อยกว่า 30 ไบต์ (IPX) หรือน้อยกว่า 42 ไบต์ (SPX)
- ความยาวของกลุ่มข้อมูลรวมมีความยาวเกิน 576 ไบต์
- ส่วนย่อย (Fragment) แรกมีขนาดเล็กเกินไปสำหรับส่วนหัวของ IPX หรือ SPX
- จำนวนนับของส่วนย่อยมีค่า 0

0xFE Undelivered กลุ่มข้อมูลไม่สามารถส่งได้ ปลายทางอยู่ที่เครื่องเดียวกันแต่ไม่พร้อมรับกลุ่มข้อมูล

0xFF Hardware/Network Failure ไม่สามารถส่งกลุ่มข้อมูลได้ เกิดขัดข้องในส่วนฮาร์ดแวร์หรือระบบเครือข่าย

เมื่อ ECB ถูก IPX เรียกใช้ในการรับกลุ่มข้อมูล Completion Code จะแสดงค่าใดๆ ได้ดังนี้

0x00 Successful กลุ่มข้อมูลได้รับเรียบร้อยแล้ว

0xFC Canceled การร้องขอการรับถูกยกเลิก

0xFD Overflow กลุ่มข้อมูลเกิดการล้น , ได้รับกลุ่มข้อมูลแต่ Fragment Count ยังเป็น 0 หรือที่ว่างที่กำหนดใน Fragment Descriptor List ไม่เพียงพอในการใส่กลุ่มข้อมูลได้หมด

0xFF Closed ซอคเก็ตที่ ECB กำหนดให้รับข้อมูลไม่ได้เปิด

หมายเลขซอคเก็ต (Socket Number)

เป็นส่วนของหมายเลขซอกเก็ตที่ ECB เรียกใช้ เมื่อ ECB ส่งข้อมูลจะใช้หมายเลขซอกเก็ต จากที่ซึ่งกลุ่มข้อมูลจะถูกส่งไป และถ้าใช้รับข้อมูลส่วนนี้จะมีหมายเลขซอคเก็ตที่กลุ่มข้อมูลจะถูกรับเข้ามา

IPX Workspace

IPX จะสงวน 4 ไบต์นี้ไว้โดยไม่ต้องกำหนดค่าเริ่มต้นและต้องไม่มีการเปลี่ยนแปลงค่าในขณะที่มีการใช้ ECB แต่ถ้า IPX ไม่เรียกใช้ ECBแล้ว โปรแกรมประยุกต์ก็สามารถใช้พื้นที่นี้ได้

Driver Workspace

เนื้อที่ 12 ไบต์จะถูกสงวนไว้โดยไดรเวอร์ของเครือข่าย โดยไม่ต้องกำหนดค่าเริ่มต้นและต้องไม่มีการเปลี่ยนแปลงค่าในขณะที่มีการใช้ ECB แต่ถ้า IPX ไม่เรียกใช้ ECBแล้ว โปรแกรมประยุกต์ก็สามารถใช้พื้นที่นี้ได้

Immediate Address

เนื้อที่ 6 ไบต์นี้จะใส่ที่อยู่ของจุดต่อที่กลุ่มข้อมูลจะส่งไปหรือที่ซึ่งกลุ่มข้อมูลส่งมาถึง โดยที่โปรแกรมประยุกต์ต้องกำหนดค่าเริ่มต้นสำหรับกลุ่มข้อมูลก่อนที่ IPXจะเรียกใช้ ECB ในการส่งข้อมูล แต่ SPXจะใส่ค่านี้ในระหว่างที่สร้างการติดต่อเพื่อที่ ECB จะถูกเรียกใช้ในโปรโตคอลนั้นๆ

Fragment Count

ส่วนที่ใส่จำนวนนับของหน่วยพักข้อมูล (มีค่ามากกว่า 0) จากที่ซึ่งกลุ่มข้อมูลถูกสร้างขึ้นเมื่อจะส่งหรือจะถูกแยกออกเมื่อได้รับข้อมูล ถ้ามีค่าเป็น 1 แสดงว่ากลุ่มข้อมูลที่ส่งมาหรือที่ได้รับมาจากหน่วยพักข้อมูล เพียงหน่วยเดียว

Fragment Descriptor

ส่วนนี้แสดงรายละเอียดของต้นทางหรือปลายทางในกลุ่มข้อมูลที่จะถูกส่ง ECB ที่ใช้จะต้องมี Fragment Descriptor อย่างน้อย 1 ส่วน (ส่วนที่อยู่ของหัว IPX หรือ SPX) กับจำนวนของส่วนข้อมูลเพิ่มเติมรวมเรียก Fragment Descriptor List

เมื่อมีหน่วยพักข้อมูลทำให้ไม่จำเป็นต้องมีการสำเนาข้อมูลและเมื่อข้อมูลมีจำนวนน้อยพอที่จะใส่ลงในหน่วยพักข้อมูลเดียวก็สามารถทำได้ง่ายโดยกำหนดส่วน Fragment Count มีค่า 1

ส่วน Fragment Descriptor ยังประกอบด้วย 2 ส่วน:

- แอดเดรสของหน่วยพักข้อมูลจากที่ซึ่ง กลุ่มข้อมูล IPX/SPX จะถูกส่งหรือรับข้อมูลเข้ามาแล้วนำไปเก็บ
- Fragment Count บอกขนาดของหน่วยพักข้อมูลที่ที่อยู่นั้นชี้อยู่

Offset	Content	Type	Order
0	Address	BYTE[4]	offset-segment
4	Fragment Count	BYTE[2]	low-high

หมายเหตุ Fragment ของหน่วยพักข้อมูลในส่วนแรกของ Fragment Descriptor List จะต้องมีค่าอย่างน้อย 30 ไบต์ (IPX) หรือ 42 ไบต์ (SPX) จะต้องมีส่วนหัวของ IPX/SPX อย่างสมบูรณ์และความยาวรวมของกลุ่มข้อมูลต้องไม่เกิน 576 ไบต์

4.3 Asynchronous Event Scheduler (AES)

AES เป็นกระบวนการของ IPX ในการจับเวลาและสร้างเหตุการณ์ขึ้นท้ายสุดของช่วงเวลานั้นๆ เช่น Watch Dog ของ NetWare เป็นตัวอย่างที่ดีตัวอย่างหนึ่งของการทำงานของ AES ถ้าเครื่องบริการไม่ได้รับกลุ่มข้อมูลจากสถานีตามตารางการติดต่อเกิน 5 นาที ก็จะทำการส่งกลุ่ม

ข้อมูล Watch Dog ไปยังสถานีงานนั้น และถ้ายังไม่มีการตอบกลับ เครื่องบริการก็จะทำการส่งกลุ่มข้อมูลไปเรื่อยทุกๆ 1 นาทีเป็นเวลา 15 นาที

หลังจาก 15 นาทีไปแล้วสถานีงานยังคงไม่ตอบ เครื่องบริการจะทำการลงบันทึกออกไป AES จะจับเวลาและกระตุ้นเหตุการณ์ที่เครื่องบริการจะกำจัดสถานีที่ไม่สามารถทำงานได้ออกจากตารางการติดต่อ

EVENT SERVICE ROUTINE (ESR)

ESR เป็นโปรแกรมประยุกต์ที่มีกระบวนการกำหนดตายตัวจะถูก IPX หรือ AES เรียกใช้หลังจากเหตุการณ์ที่เฉพาะเจาะจงได้เกิดขึ้นมา ซึ่งเกิดได้เมื่อมีการร้องขอการส่งข้อมูลหรือการรับข้อมูลได้เสร็จสิ้นสมบูรณ์ รวมทั้งเหตุการณ์ที่ IPX ดำเนินการจัดการตัวมันเองใหม่เสร็จหรือเหตุการณ์ที่ได้กำหนดขึ้นเพื่อเรียกใช้ ESR

ESR มักจะถูกเรียกท้ายสุดของเหตุการณ์นั้นๆ ในขณะเดียวกับที่ InUse Flag ได้ถูกตั้งค่าศูนย์ใหม่ และ Completion Code จะถูกเซต ข้อมูลที่เกี่ยวข้องกับเหตุการณ์ก็จะถูกบันทึกไว้ใน ECB ค่า ECB ก็จะถูกดึงออกจากข้อมูลภายใน IPX หรือ AES

ECB จะชี้ไปที่ ESR ที่จะถูกเรียกหลังจากเหตุการณ์ที่คาดหวังเกิดขึ้น และเข้าควบคุม ESR โดย ESR ก็จะได้รับค่าตัวชี้จาก ECB นั้นๆ

ESR เป็นรูปแบบหนึ่งของโปรแกรมบริการอินเตอร์รัพท์ และจะถูกเรียกเมื่อโปรเซสเซอร์ถูกดิสเอเบิ้ล(Disable) มีลักษณะเดียวกับโปรแกรมอินเตอร์รัพท์ทั่วไป เช่น ESR ควรจะใช้เวลาในการทำงานสั้นที่สุดภายในช่วงเวลาที่โปรเซสเซอร์ดิสเอเบิ้ล และ ESR ก็ไม่ควรหน้าที่เป็นโปรแกรมหลักหรือโปรแกรมแสดงผลในการทำงาน

โปรแกรมประยุกต์ที่ใช้ในการรับหรือส่งกลุ่มข้อมูลจำนวนมากโดยเฉพาะการส่งข้ามจุดต่อภายในเครือข่ายซึ่งควรจะใช้ ECB หลายๆตัว ในโปรแกรม

ตัวอย่างเช่นในกระบวนการรับข้อมูล IPX ควรจะจัดการรับกลุ่มข้อมูลแบบไม่ต่อเนื่องได้ในขณะที่ ESR จะเรียงลำดับกลุ่มข้อมูลตามรายการที่จะประมวลผล หลักการสำคัญคือต้องประมวลผลให้ได้รวดเร็วตามพื้นฐาน FCFS (First- come- first- serve)

การติดต่อ ESR โดยทั่วไป

เนื่องจาก ESR เป็นโปรแกรมบริการอินเตอร์รัพท์ของ ECB และ AES จึงต้องมีเงื่อนไขดังนี้

- รีจีสเตอร์ AL ต้องระบุค่าดังนี้

0x00 AES เรียก ESR

0xFF IPX เรียก ESR

- ECB จะระบุค่า ตัวชี้ ESR ใน ES:SI
- Flags และ รีจีสเตอร์ต่างๆ ยกเว้น SS และ SP จะถูกเก็บค่าลงในสแต็ก ESR จะเก็บค่า SS และ SP ไว้เพื่อคืนกลับให้หลังการทำงาน
- อินเตอร์รัพท์ดิสเอเบิ้ล

เมื่อ ESR ถูกเรียก In Use Flag ใน ECB จะถูกตั้งค่าเป็นศูนย์ใหม่ และ Completion Code จะถูกเซต ข้อมูลที่เกี่ยวข้องกับเหตุการณ์ก็จะถูกบันทึกไว้ใน ECB ค่า ECB ก็จะถูกดึงออกจากข้อมูลภายใน IPX หรือ AES

- รีจีสเตอร์ DS จะต้องตั้งค่าเริ่มต้นใหม่โดย ESR ก่อนที่จะมีค่าอ้างอิงใดๆถูกกำหนดขึ้นในโปรแกรม
- ESR จะไม่คืนค่าหรือ Flag ใด
- ESR จะกลับมาโปรแกรมหลักด้วยการดิสเอเบิ้ลอินเตอร์รัพท์
- ESR จะต้องใช้คำสั่ง RETF ในการกลับสู่โปรแกรมหลัก ถ้าโปรแกรมที่เรียกใช้มีความยาวมากๆหรือซับซ้อนก็ตาม
- ESR ที่ถูกเรียก อาจเอนาเบิ้ลอินเตอร์รัพท์ได้ ซึ่งต้องระมัดระวัง

4.4 AES Function

IPXRelinquishControl

ฟังก์ชันนี้ใช้สำหรับยกเลิกการควบคุมของซีพียูของสถานีฐาน

ไวยากรณ์ (Syntax) void IPXRelinquishControl (void)

ค่าส่งกลับ (Return Values) ไม่มี

ข้อกำหนด (Remarks)

โปรแกรมประยุกต์สามารถใช้ฟังก์ชันนี้ในการยกเลิกการควบคุมของซีพียูของสถานีฐานได้ชั่วคราวเพื่อให้การประมวลผลอื่นสามารถทำได้ ในขณะที่โปรแกรมประยุกต์กำลังรอสัญญาณอินพุท

โปรแกรมประยุกต์ของ IPX ควรเรียกฟังก์ชันนี้ซ้ำ ในขณะที่ซีพียูได้ไปทำการประมวลผลอื่นแล้วซึ่งมีความสำคัญมากเมื่อโปรแกรมเป็นแบบฝังตัวเพราะจะทำให้เครื่องบริการแฟ้มหรือบริดจ์มีประสิทธิภาพขึ้น

การเรียกใช้ฟังก์ชันนี้จะทำให้ไดรเวอร์ที่ใช้ในการสื่อสารใน IPX ทำงาน ซึ่งเป็นผลดีทำให้ไม่ต้องเกิดการอินเตอร์รัพท์ ในระบบทั่วไปไดรเวอร์ของเครือข่ายจะทำการหยั่งสัญญาณโดยใช้ฟังก์ชันนี้ ซึ่งหมายถึงเป็นโอกาสเดียวที่ไดรเวอร์จะใช้ซีพียูในการส่งรับกลุ่มข้อมูล รวมทั้งเหตุการณ์ที่ใช้ประยุกต์ในตัวมัน

ตัวอย่าง (Example)

```
#include <nxt.h>
IPXRelinquishControl ();
```

IPXCloseSocket

ใช้สำหรับปิดซอคเก็ต IPX

ไวยากรณ์ void IPXCloseSocket (WORD SocketNumber)

SocketNumber (Input) ใส่หมายเลขซอคเก็ตที่ต้องการจะปิด

ค่าส่งกลับ ไม่มี

ข้อกำหนด

ฟังก์ชันนี้จะใช้ปิดซอคเก็ตเฉพาะของ IPX และยกเลิกเหตุการณ์ที่ ECB ทำงานบนซอคเก็ตนั้น IPX จะคืนค่า 0xFC ใน Completion Code ของแต่ละ ECB เพื่อบ่งชี้ว่าเหตุการณ์นั้นได้ถูกยกเลิกไปแล้ว IPX ก็จะเซตค่า InUseFlag ECB เป็น 0x00 (AVAILABLE_ FOR_USE)

หมายเลขซอคเก็ตสามารถที่จะปิดได้ และจะไม่เกิดข้อผิดพลาดขึ้นถ้าซอคเก็ตนั้นยังไม่เปิด ก่อนที่โปรแกรมนั้นจะจบการทำงานและลบค่า ESR ทิ้ง จำเป็นต้องปิดซอคเก็ตที่ถูกเปิดแบบ long-lived มิฉะนั้นแล้ว ESR ก็จะยังคงถูกเรียกได้ ซึ่งจะทำให้ระบบหยุดชะงักได้ ถ้าซอคเก็ตถูกเปิดแบบ short-lived ก็จะถูกปิดเองโดยอัตโนมัติ และ ฟังก์ชันนี้ไม่ควรถูกเรียกมาจากภายในตัวของ ESR

ดูเพิ่มเติม IPXOpenSocket

ตัวอย่าง

```
#include <nxt.h>
WORD socketNumber;
IPXCloseSocket (socketNumber);
```

IPXGetDataAddress

ฟังก์ชันนี้จะคืนค่าแอดเดรสของข้อมูลในหน่วยพักข้อมูลที่ถูกชี้โดยแอดเดรส

ไวยากรณ์ void IPXGetDataAddress (BYTE *data , WORD *address)

data (Input) ตัวชี้ข้อมูลที่ต้องการทราบแอดเดรส

address (Output) แอดเดรสของข้อมูล

ค่าส่งกลับ ไม่มี

ข้อกำหนด

แอดเดรสจะถูกคืนมาในลักษณะของลำดับออฟเซต- เซกเมนต์ ตัวอย่างเช่น

แอดเดรส = ออฟเซต แอดเดรส+2 = เซกเมนต์

ตัวอย่าง

```
#include <nit.h>
#include <nxt.h>
int completionCode;
BYTE data;
WORD address[2];
CompletionCode = IPXGetDataAddress (&data,address);
```

IPXGetInternetworkAddress

ฟังก์ชันนี้คืนค่าแอดเดรสของเครือข่ายและแอดเดรสจุดต่อของสถานีงานที่ร้องขอมา

ไวยากรณ์ void IPXGetInternetworkAddress (BYTE *networkAddress)

networkAddress (output) เป็นตัวชี้แอดเดรสของเครือข่ายท้องถิ่น 1 ไบต์ดังนี้

ไบต์ 0 - 3 = หมายเลขเครือข่าย (Network number)

ไบต์ 4 - 9 = หมายเลขจุดอื่น (Node number)

ค่าส่งกลับ ไม่มี

ข้อกำหนด

ฟังก์ชันนี้จะเป็นประโยชน์อย่างมากสำหรับโปรแกรมประยุกต์ในการแจ้งจุดต่ออื่นๆ ว่าสถานีที่ร้องขอมีหมายเลขเท่าใดในจุดต่อ โดยจะไม่ได้คืนค่าที่เป็นหมายเลขซอคเก็ตจะเรียกใช้หมายเลขซอคเก็ตก็ต่อเมื่อจะเปิดซอคเก็ตเท่านั้น โปรแกรมประยุกต์ที่ได้รับหมายเลขเครือข่ายครบทั้ง 12 ไบต์ จะต้องใช้หมายเลขซอคเก็ตด้วย

ตัวอย่าง

```
#include<nxt.h>
IPXAddress networkAddress;
IPXGetInternetworkAddress (&networkAddress);
```

IPXInitialize

ฟังก์ชันนี้จะไปเอาค่าหมายเลขของการติดต่อของ IPX มา

ไวยากรณ์ int IPXinitialize (void)

ค่าส่งกลับ 0 (0x00) SUCCESSFUL

240 (0xF0) IPX_NOT_INSTALLED

ข้อกำหนด

ฟังก์ชันนี้จะทำการตั้งค่าเริ่มต้นต่างๆ ในไลบรารี่ NIT C ด้วยค่าหมายเลขการบริการของ IPX ฟังก์ชันนี้จะต้องถูกเรียกก่อนที่ฟังก์ชันใดๆ ของ IPX จะถูกนำมาใช้

ดูเพิ่มเติม SPXInitialize

ตัวอย่าง

```
#include <nxt.h>
int statusCode;
statusCode = IPXInitialize ();
```

IPXListenForPacket

เป็นฟังก์ชันที่ใช้จัดการกับ IPX ให้พร้อมที่จะรับกลุ่มข้อมูล IPX

ไวยากรณ์ void IPXListenForPacket (ECB *eventControlBlock)

eventControlBlock (Input/Output) รับค่าตัวชี้โครงสร้าง ECB

ค่าส่งกลับ ไม่มี

ข้อกำหนด

ฟังก์ชันนี้จะส่งผ่านค่าหมายเลข ECB ไปยัง IPX เพื่อให้เกิดการรับกลุ่มข้อมูล IPX ได้ และจะคืนการควบคุมกลับไปยังโปรแกรมที่เรียกมา โดย IPX จะคอยและรับกลุ่มข้อมูลที่เข้ามา ก่อนเรียกฟังก์ชันนี้โปรแกรมจะต้องเปิดซอคเก็ตและกำหนดค่าเริ่มต้นของ ESRAddressใน ECB , หมายเลข ซอคเก็ต , Immediate Address , Fragment Count , Fragment Descriptor ถ้าไม่ต้องการใช้ ESR ค่า ESRAddress จะต้องตั้งค่าเป็น NULL เริ่มต้น IPX จะเซตค่า InUseFlagใน ECB เป็น 0xFE เพื่อให้ ECB คอยรับกลุ่มข้อมูล IPX สามารถเลือกใช้ ECB จำนวนไม่จำกัดในการรับข้อมูลที่เข้ามาเรื่อยๆ ผ่านซอคเก็ต

เมื่อ IPX ตรวจสอบกลุ่มข้อมูลที่เข้ามาได้ IPX จะใช้ ListenECB 1 ตัวในการรับข้อมูล IPX จะบันทึกค่าที่เหมาะสมใน CompletionCode ของ ListenECB ที่ถูกเลือก และจะใส่ค่าหมายเลขจุดต่อผู้ส่งหรือ Local บริดจ์ ใน Immediate Address ของ ECB นั้นด้วย

ท้ายสุด IPX จะเซตค่า InUseFlag กลับคืนเป็น 0x00 (AVAILABLE_FOR_USE) และเรียกใช้ ESR ตาม ESRAddress ใน ECB (ถ้าต้องการประยุกต์ใช้)

ค่า CompletionCode ของ ECB ที่จะเกิดขึ้นได้มีดังต่อไปนี้

0 (0x00) SUCCESSFUL

252 (0xfC) REQUEST_CANCELLED

253 (0xFD) BAD_PACKET

255 (0XFF) SOCKET_NOT_OPEN

ดูเพิ่มเติม IPXSendPacket

ตัวอย่าง

```
#include <nxt.h>
ECB eventControlBlock;
IPXListenForPacket (&eventControlBlock);
```

IPXOpenSocket

ใช้สำหรับเปิด IPX ซอคเก็ต

ไวยากรณ์

Int IPXOpenSocket (Byte *socketNumber,Byte socketType)

socketNumber (Input/Output) ใส่หมายเลขซอคเก็ต

socketType (Input) ชนิดของซอคเก็ต ดังนี้

0x00 เปิดซอคเก็ตไว้จนกว่าโปรแกรมจะจบ (short-lived)

0xFF เปิดซอคเก็ตไว้จนกว่าโปรแกรมจะสั่งปิดมัน(long-lived)

ค่าส่งกลับ 0 (0x00) SUCCESSFUL

240 (0xF0) IPX_NOT_INSTALLED

254 (0xFE) SOCKET_TABLE_FULL

255 (0xFF) SOCKET_ALREADY_OPEN

ข้อกำหนด

โปรแกรมจะต้องใช้ฟังก์ชันนี้ในการเปิดซอคเก็ต ก่อนที่จะรับกลุ่มข้อมูลบนซอคเก็ตนั้นได้

SocketNumber จะระบุหมายเลขซอคเก็ตที่จะถูกเปิด ถ้าใส่ค่า 0x0000 จะอนุญาติให้ IPX เปิด ซอคเก็ตใดก็ได้ในช่วง 0x4000 ถึง 0x5000 และจะคืนค่า SocketNumber มาแสดงให้เห็นด้วยจำนวน 2 ไบต์ เช่น BYTE socketNumber[2]

ถ้าฟังก์ชันนี้คืนค่า CompletionCode 0x00 แสดงว่า ซอคเก็ตเปิดได้แล้ว ถ้า 0xFE แสดงว่า ตารางซอคเก็ตเต็ม ถ้า 0xFF แสดงว่า ซอคเก็ตนั้นได้เปิดไว้อยู่ก่อนแล้ว

socketType จะบ่งชี้ถึงระยะเวลาในการเปิดซอคเก็ต ถ้าโปรแกรมเป็นแบบไม่ฝังตัว socketType ควรเป็นแบบชั่วคราว (0x00 - short-lived) ซึ่งจะปิดตัวเองอัตโนมัติหลังจบการทำงาน และ ยกเลิกเหตุการณ์ที่เกี่ยวข้องกับซอคเก็ตออกไป ถ้าซอคเก็ตเป็นแบบชนิดถาวร (0xFF - long-lived) ESR ก็ยังสามารถใช้ได้แม้ว่าโปรแกรมจะจบการทำงานไปแล้ว

โดยทั่วไป IPX สามารถเปิดซอคเก็ตได้ 20 ซอคเก็ตต่อ 1 สถานีฐาน แต่อาจใช้ได้ถึง 150 ซอคเก็ต โดยแก้ไขไฟล์ SHELL.CFG

ดูเพิ่มเติม IPXCancelEvent,IPXCloseSocket

ตัวอย่าง

```
#include <nxt.h>
BYTE[2]  socketNumber;
BYTE     socketType;
CompletionCode = IPXOpenSocket (socketNumber , socketType);
```

IPXSendPacket

ใช้ทำการส่งกลุ่มข้อมูล IPX

ไวยากรณ์ void IPXSendPacket (ECB *eventControlBlock)

eventControlBlock (Input/Output) ใส่ค่าตัวชี้โครงสร้าง ECB

ค่าส่งกลับ ไม่มี

ข้อกำหนด

ฟังก์ชันนี้จะผ่านค่าหมายเลข ECB ให้กับ IPX ในการส่งกลุ่มข้อมูล IPX ฟังก์ชันจะคืนการควบคุมให้กับโปรแกรมเดิมในขณะที่ IPX พยายามที่ส่งข้อมูล ก่อนที่จะเรียกใช้ฟังก์ชันนี้มา โปรแกรมจะต้องเปิดซอคเก็ตและกำหนดค่าเริ่มต้นของ ESRAddress ใน ECB , หมายเลขซอคเก็ต, Immediate Address , Fragment Count ,Fragment Descriptor โปรแกรมจะต้องกำหนดค่า PacketType และปลายทางลงบนส่วนหัวของ IPX ฟังก์ชันนี้ก็จะส่งผ่าน ECB ให้กับไดรเวอร์การติดต่อสื่อสารของเครือข่ายในการเริ่มต้นส่งข้อมูลโดยไม่ต้องเปิดซอคเก็ตเพื่อที่จะส่ง

เริ่มแรก IPX จะเซตค่า InUseFlag ใน ECB เป็น 0xFF เพื่อที่จะส่งกลุ่มข้อมูล หลังจากที่มีการส่งข้อมูล IPX ก็จะมีการเซต CompletionCode ให้เหมาะสม และกำหนดค่า InUseFlag เป็น 0x00 (AVAILABLE_FOR_USE) สุดท้าย IPX ก็จะเรียกใช้ ESR โดยใช้หมายเลข ESRAddress ถ้าต้องการใช้โปรแกรมประยุกต์ต่อไป

ค่า CompletionCode ที่เป็นไปได้ มีค่าดังนี้

0 (0x00) SUCCESSFUL (sent, not necessarily received)

252 (0xFC) REQUEST_CANCELLED

253 (0xFD) BAD_PACKET (fragment แรกของส่วนหัวในกลุ่มข้อมูลไม่ได้มีจำนวน 30 ไบต์ หรือความยาวรวมเกิน 576 ไบต์)

254 (0xFE) PACKET_NOT_DELIVERABLE

255 (0xFF) HAREDWARE_FAILURE

ค่า CompletionCode มีค่า 0x00 บอกถึงว่า กลุ่มข้อมูลได้ถูกส่งเสร็จสิ้นแล้ว แต่ไม่รับประกันว่ากลุ่มข้อมูลนั้นจะได้รับครบหรือไม่ที่จุดต่อปลายทาง ถ้า มีค่า 0xFE บอกว่า กลุ่มข้อมูลยังได้ส่ง ด้วยเหตุผลคือ 1. IPX ไม่สามารถหาเส้นทางของบริดจ์ไปยังเครือข่ายปลายทางได้ 2. ตรวจสอบไม่พบจุดต่อปลายทาง

โปรแกรมสามารถส่งกระจายกลุ่มข้อมูล IPX ไปยังซอคเก็ตใดก็ได้ รวมทั้งซอคเก็ตของตัวมันเอง ถ้ากลุ่มข้อมูลถูกส่งไปยังซอคเก็ตในจุดต่อเดียวกันเรียกว่า กลุ่มข้อมูลแบบ intra-node ในการส่งกระจายกลุ่มข้อมูล ส่วนหัวของกลุ่มข้อมูลจะมีหมายเลขจุดต่อปลายทางเป็น 0xFFs(-1) และถ้าส่งกระจายภายในเครือข่ายเดียวกัน ก็จะต้องเซตค่าของ ImmediateAddress เป็น 0xFF ด้วย

ดูเพิ่มเติม IPXGetLocalTarget,IPXListenForPacket

ตัวอย่าง

```
#include <nxt.h>
ECB eventControlBlock;
IPXSendPacket (&eventControlBlock);
```

4.6 SPX Function

SPXEstablishConnection

ฟังก์ชันนี้จะทำการสร้างการติดต่อกับซอคเก็ตฝ่ายรับ

ไวยากรณ์ int SPXEstablishConnection (BYTE retryCount , BYTE watchDog , WORD *connectionID , ECB *eventControlBlock) retryCount (Input) มีค่าจาก 0 ถึง 255

watchDog (Input) แฟค watchDog

connectionID (Output) ตัวชี้ตัวเลขซึ่งจะรับค่าหมายเลขการติดต่อ

eventControlBlock (Input/Output) ตัวชี้โครงสร้าง ECB

ค่าส่งกลับ 0 (0x00) SPX_CONNECTION_STARTED

239 (0xEF) SPX_CONNECTION_TABLE_FULL

253 (0xFD) SPX_MALFORMED_PACKET

255 (0xFF) SPX_SOCKET_NOT_OPENED

ข้อกำหนด

ฟังก์ชันนี้ จะผ่านค่า retryCount,SPX watchDog flag และแอดเดรสของ eventControlBlock เพื่อสร้างการติดต่อให้เกิดขึ้น ฟังก์ชันจะคืนการควบคุมให้กับโปรแกรมเดิมในขณะที่ SPX พยายามที่จะสร้างการติดต่อ

ก่อนการเรียกใช้ โปรแกรมต้องแสดงผลดังนี้

- ต้องกำหนดค่า ESRAddress, socketNumber, fragmentCount ให้มีค่า 0x01 และ fragmentDescriptor จะชี้ไปยังหน่วยพักข้อมูล 42 ไบต์ที่กำหนดไว้ในส่วนหัว ของกลุ่มข้อมูล SPX
- กำหนดปลายทางไว้ที่ส่วนหัวของ SPX ซึ่งจะไม่กำหนดค่า -1 ในการส่งกระจายข้อมูล
- จำเป็นต้องมีอย่างน้อย 2 listenECB และจะส่งผ่านโดย SPXListenForSequencedPacket เมื่อ SPX ส่งกลุ่มข้อมูล SPXEstablishConnection ไปแล้ว ก็จะใช้ listenECB อีกอันในตอบรับการรับกลุ่มข้อมูลจากซอคเก็ตปลายทาง ส่วนที่เหลืออาจใช้ในการ WatchDog ในขณะที่ SPX ทำการสร้างการติดต่อ

SPX จะสร้างการติดต่อเป็น 2 ขั้นตอนคือ ขั้นแรก SPX จะส่งกลุ่มข้อมูล Establish Connection ไปยังซอคเก็ตปลายทาง และเพื่อที่สามารถรับข้อมูลได้ ซอคเก็ตปลายทางจะต้องสร้าง ECB ที่ SPX ใช้ด้วยฟังก์ชัน SPXListenForConnection ซึ่ง SPX จะกำหนดค่า retryCount, watchDog และ ESRAddress ในการทำงานดังนี้

ค่า retryCount จะบอกค่าจำนวนครั้งในการส่งกลุ่มข้อมูลซ้ำก่อนที่จะตัดสินว่าสถานีงานนั้นไม่สามารถตอบรับได้โดยอาจตั้งค่า 0x00 ให้ SPX ใช้ retryCount ของตัวเอง หรือตั้งค่า 1-255 เป็นจำนวนครั้งที่ส่ง

กระบวนการ watchDog เป็นการตรวจสอบการติดต่อให้ทำงานเป็นไปได้ด้วยดี โปรแกรมสามารถใช้ watchDog นี้โดยให้มีค่า 0xFF ENABLE_WATCHDOG และ 0x00 ในการยกเลิกการ

ทำงานนี้ ถ้าการทำงานนี้ตรวจสอบว่าการติดต่อล้มเหลว จะยกเลิกการติดติอและแจ้งค่าใน SPXListenSequencedPacket ECB โดย completionCode มีค่า SPX_CONNECTION_FAILED (0xED) และ connectionID ของการติดต่อที่เลิกไปจะถูกเก็บไว้ใน IPXWorkspace ของ ECB ในกรณีที่สร้างการติดต่อได้ InUseFlag และ completionCode ก็จะมีค่า 0

จำเป็นต้องเปิดซอคเก็ตที่ได้ระบุไว้ก่อนเรียกใช้ฟังก์ชันนี้ด้วย ซอคเก็ตที่ถูกใช้ในการติดต่อของ SPX ต้องไม่ใช้ถูกใช้โดยฟังก์ชัน IPXSendPacket หรือ IPXListenForPacket ในการส่งหรือรับกลุ่มข้อมูลโดยตรง แต่จะขึ้นกับ SPX ถ้าซอคเก็ตไม่เปิดไว้ จะออกจบการทำงานทันที

ถ้าฟังก์ชันคืนค่า completionCode 0x00 ก็จะได้ค่า connectionID ในส่วนของ sourceConnectionID กลับมาด้วย ถึงแม้ว่าสร้างการติดต่อยังไม่สำเร็จ ข้อมูลการติดต่อก็ยังคงเก็บไว้ในตารางการติดต่อที่จุดต่ออยู่ แล้ว SPX จึงจะส่งกลุ่มข้อมูล SPXEstablishConnection ไปยังจุดต่อปลายทาง

IPXCancelEvent จะไม่สามารถยกเลิกเหตุการณ์ที่ SPXEstablishConnection ทำงานได้ ถ้าโปรแกรมไม่ต้องการให้มีการสร้างการติดต่อ จะต้องใช้ฟังก์ชัน SPXAbortConnection นี้แทนจะถูกยกเลิก

ค่า completionCode อาจมีค่าได้ดังต่อไปนี้

237 (0xED) SPX_NO_ANSWER_FROM_TARGET
มีการผิดพลาดที่ฮาร์ดแวร์หรือโปรแกรมประยุกต์ใช้ฟังก์ชัน SPXAbortConnection

239 (0xEF) SPX_CONNECTION_TABLE
จะไม่มีการติดต่อจนกว่าการติดต่อที่มีอยู่จะถูกยกเลิกไป ตารางการติดต่อจะมีที่ว่างพอสำหรับการติดต่อ 100 ครั้ง

252 (0xFC) SPX_SOCKET_CLOSED
ซอคเก็ตที่ติดต่อไม่เปิดและ เรียกใช้ ESR ไม่ได้

253 (0xFD) SPX_MALFORMED_PACKET
fragmentCount ไม่ได้มีค่า 1 หรือหน่วยพักข้อมูลไม่ใช่ 42 ไบต์

255 (0xFF) SPX_SOCKET_NOT_OPENED

ดูเพิ่มเติม SPXAbortConnection , SPXTerminateConnection

ตัวอย่าง

```
#include <nxt.h>
int completionCode
```

```
BYTE  retryCount , watchDog;
WORD  connectionID;
ECB   eventControlBlock;
completionCode = SPXEstablishConnection (retryCount , watchDog ,
                                          &connectionID , &eventControlBlock);
```

SPXInitialize

เป็นฟังก์ชันตรวจสอบการติดตั้ง SPX

ไวยากรณ์ BYTE SPXInitialize (BYTE *majorRevisionNumber ,

BYTE *minorRevisionNumber , WORD *maxConnections,

WORD *availableConnections)

majorRevisionNumber (Output)

minorRevisionNumber (Output)

maxConnections (Output)

availableConnections (Output)

ค่าส่งกลับ 0 (0x00) SPX_NOT_INSTALLED

255 (0xFF) SPX_IS_INSTALLED

ข้อกำหนด

ฟังก์ชันนี้จะตัดสินใจว่า จะติดตั้ง SPX ในจุดต่อซึ่งมีโปรแกรมประยุกต์ทำงานอยู่หรือไม่ ถ้าจะติดตั้ง SPX ฟังก์ชันนี้จะคืนค่า major และ minor revision numbers ของ SPX , maximum of connections และ SPX Connection ที่สามารถใช้ได้

CompletionCode บอกถึงการติดตั้งของ SPX (0x00 = ยังไม่ได้ติดตั้ง ,0xFF = ติดตั้งแล้ว) major และ minor revision Number จะบอก revision ของ SPX ที่ติดตั้ง maxConnections เป็นตัวบอกค่าที่มากที่สุดที่เวอร์ชันของ SPX นั้นสามารถมีได้ availableConnections บอกจำนวนการติดต่อของ SPX ที่สามารถใช้ได้ในโปรแกรม

ตัวอย่าง

```
#include <nxt.h>
BYTE  isSPXInstalled , majorRevisionNumber , minorRevisionNumber;
WORD  maxConnections, availableconnections;
isSPXInstalled = SPXInitialize (&majorRevisionNumber , &minorRevisionNumber ,
```

&maxConnections , &availbleConnections);

SPXListenForConnection

ฟังก์ชันนี้จะรับกลุ่มข้อมูลการสร้างการติดต่อ และสร้างการติดต่อ SPX กับฝ่ายตรงข้าม

ไวยากรณ์ void SPXListenForConnection (BYTE retryCount, BYTE watchDog, ECB *eventControlBlock)

retryCount (Input) มีค่า 0 ถึง 255

watchDog (Input) แฟค watchDog

eventControlBlock (Input) ตัวชี้โครงสร้าง ECB

ค่าส่งกลับ ไม่มี

ข้อกำหนด

ฟังก์ชันนี้ จะผ่านค่า retryCount, SPX watchDog flag และแอดเดรสของ eventControlBlock เพื่อรอและรับกลุ่มข้อมูล SPXEstablishConnection เมื่อได้รับแล้ว ฝ่ายส่งก็จะสร้างการติดต่อ ฟังก์ชันจะคืนการควบคุมให้กับโปรแกรมเดิมในขณะที่ SPX พยายามที่จะรอรับกลุ่มข้อมูล SPXEstablishConnection และสร้างการติดต่อ

ก่อนที่จะใช้ฟังก์ชันนี้จะต้องใช้ IPXOpenSocket ในการเปิดซอคเก็ตที่ใช้ติดต่อ เมื่อ SPX สร้างการติดต่อได้ InUseFlag ใน ECB จะมีค่า 0 และ completionCode มีค่า 0x00 แต่ถ้าเกิดข้อผิดพลาด ค่าที่คืนกลับจะมีค่าดังนี้:

239 (0xEF) SPX_CONNECTION_TABLE_FULL
ตารางการติดต่อไม่มีที่ว่าง โปรแกรมต้องรอจนกว่าการติดต่อที่มีอยู่จะออกจากการทำงาน

252 (0xFC) SPX_COMMAND_CANCELLED
IPXCancelEvent ยกเลิกคำสั่งเดิม

255 (0xFF) SPX_SOCKET_NOT_OPENED

SPX จะสร้างการติดต่อเป็น 2 ขั้นตอนคือ ขั้นแรก SPX จะรอรับกลุ่มข้อมูล Establish Connection ขั้นต่อมาเมื่อได้รับกลุ่มข้อมูลแล้วก็จะพยายามสร้างการติดต่อ ถ้าสร้างได้สำเร็จ กลุ่มข้อมูลการตอบรับจะถูกส่งกลับไปสู่จุดต่อฝ่ายส่ง ทั้งสองฝ่ายก็จะสามารถติดต่อส่งและรับข้อมูลกันได้

เพื่อที่ติดต่อดังกล่าวได้ SPX จะกำหนดค่า retryCount, watchDog และ ESRAddress ในการทำงานดังนี้

ค่า retryCount จะบอกค่าจำนวนครั้งในการส่งกลุ่มข้อมูลซ้ำก่อนที่จะตัดสินว่าสถานีงานนั้นไม่สามารถตอบรับได้โดยอาจตั้งค่า 0x00 ให้ SPX ใช้ retryCount ของตัวเอง หรือตั้งค่า 1-255 เป็นจำนวนครั้งที่ส่ง

กระบวนการ watchDog เป็นการตรวจสอบการติดต่อให้ทำงานเป็นไปได้ด้วยดี โปรแกรมสามารถใช้ watchDog นี้โดยให้มีค่า 0xFF ENABLE_WATCHDOG และ 0x00 ในการยกเลิกการทำงานนี้ ถ้าการทำงานนี้ตรวจสอบว่าการติดต่อล้มเหลว จะยกเลิกการติดต่อและระบุค่าใน completionCode ของ ECB ฝ่ายผู้รับมีค่า SPX_CONNECTION_FAILED (0xED) และ connectionID ของการติดต่อที่เลิกไปจะถูกเก็บไว้ใน IPXWorkspace ของ ECB

โปรแกรมจะมีการเรียกใช้ฟังก์ชัน SPXListenForSequencedPacket อย่างน้อย 2 ครั้งในการรอรับกลุ่มข้อมูลการสร้างการติดต่อและตอบรับการรับกลุ่มข้อมูล โดย ECB จะส่งผ่านค่าให้นี้โดยไม่ใช้กลุ่มข้อมูลหรือ fragment เลย จากนั้น SPX จะส่ง กลุ่มข้อมูลการตอบรับกลับไปยังจุดต่อเริ่มต้นและบันทึกหมายเลข connectionID ลงใน 2 ไบต์แรกของ IPXWorkspace, SPXก็จะเก็บแอดเดรสจุดต่อจำนวน 12 ไบต์ไว้ใน driverWorkspace ด้วย SPX จะกำหนดค่า 0x00 (ConnectionEstablished) ลงใน completionCode และค่า 0x00 (AVIALABLE_FOR_USE) ของ InUseFlag ใน ECB

โปรแกรมจะยกเลิกการทำงานของ SPXListenForConnection (ด้วย IPXCancelEvent) เวลาใดก็ได้ ซึ่ง SPX จะเก็บค่า 0xFC(SPX_COMMAND_CANCELED)ในส่วน completionCode ของ ECB ไว้

ดูเพิ่มเติม IPXOpenSocket , SPXEstablishConnection,
SPXListenForSequencedPacket

ตัวอย่าง

```
#include <nxt.h>
BYTE   retrycount , watchDog;
ECB    eventControlBlock;
SPXListenForConnection (retryCount, watchDog , &eventControlBlock);
```

SPXListenForSequencedPacket

ฟังก์ชันนี้จะส่ง ECB ไป SPX เพื่อใช้ในการรับกลุ่มข้อมูลที่ถูกส่งมาเรื่อยๆ

ไวยากรณ์ void SPXListenForSequencedPacket (ECB *eventControlBlock)

eventControlBlock (Input/Output) ตัวชี้ ECB

ค่าส่งกลับ ไม่มี

ข้อกำหนด

ฟังก์ชันจะส่งแอดเดรสของ ECB และหน่วยพักข้อมูลที่ถูกกำหนดไว้ใช้ในการรับกลุ่มข้อมูลที่เข้ามาให้ SPX แล้วจะคืนการควบคุมให้กับโปรแกรมเดิม SPX ทำให้โปรแกรมสามารถที่จะรับข้อมูลแบบอะซิงค์โครนัส ซึ่งฟังก์ชันนี้กำหนดให้รอรับกลุ่มข้อมูล ECB ที่ระบุไว้เท่านั้น

ก่อนที่จะเรียกใช้ฟังก์ชันนี้ โปรแกรมจะต้องเปิดซอคเก็ตและกำหนดค่าเริ่มต้นของ ESRAddressใน ECB , หมายเลขซอคเก็ต , Immediate Address , Fragment Count ,Fragment Descriptor

เมื่อ InUseFlag รีเซทค่าเป็น 0 ในส่วนของ completionCode ก็จะมีค่าดังนี้

0 (0x00) SPX_PACKET_SUCCESSFUL

237 (0xED) SPX_CONNECTION_FAILED

252 (0xFC) SPX_SOCKET_CLOSED

253 (0xFD) SPX_PACKET_OVERFLOW

255 (0xFF) SPX_SOCKET_NOT_OPENED

ถ้า completionCode มีค่า 0xED แสดงว่าการติดต่อล้มเหลวและยกเลิกไป ถ้ามีค่า 0xFD แสดงว่าได้รับกลุ่มข้อมูลแล้วแต่ที่ว่างที่กำหนดใน Fragment Descriptor ไม่เพียงพอในการเก็บทั้งหมด SPX จะไม่สามารถทำงานได้ปกติ ถ้าลูกข่ายไม่กำหนดให้ ListenECB มีขนาดไม่เพียงพอ เพราะทั้งกลุ่มข้อมูลของระบบและของผู้ใช้จะถูกเก็บเข้าไปยังหน่วยพักข้อมูลใน ListenECB เดียวกัน

ถ้า ECB ของผู้รับถูกยกเลิกไปด้วยฟังก์ชัน IPXCancelevent แล้ว ESRAddress จะต้องกำหนดค่าเริ่มต้นให้ใหม่ก่อนที่ ECB จะถูกใช้อีกครั้ง

ดูเพิ่มเติม SPXListenForConnection, SPXSendSequencedPacket

ตัวอย่าง

```
#include <nxt.h>
ECB  eventControlBlock;
SPXListenForSequencedPacket (&eventControlBlock);
```

SPXSendSequencedPacket

ใช้สำหรับส่งกลุ่มข้อมูล SPX

ไวยากรณ์ void SPXSendSequencedPacket (WORD connectionNumber,
ECB *eventControlBlock)

connectionIDNumber (Input) เลขที่กำหนดโดย SPX เมื่อสร้างการติดต่อได้แล้ว

eventControlBlock (Input) ตัวชี้โครงสร้าง ECB

ค่าส่งกลับ ไม่มี

ข้อกำหนด

ฟังก์ชันนี้จะส่งค่า ConnectionID และ Address ECB ให้กับ SPX เพื่อใช้ในการส่งข้อมูล และจะทำการคืนการควบคุมให้กับโปรแกรมเดิม ในขณะที่ SPX กำลังส่งกลุ่มข้อมูล

ก่อนที่จะทำการเรียกใช้ฟังก์ชันนี้ โปรแกรมจะต้องตั้งค่าเริ่มต้นให้กับ ESRAddress , FragmentCount , FragmentDescriptor FragmentDescriptor แรกของส่วนหัวของ กลุ่มข้อมูล SPX จะต้องมีแหล่งพักข้อมูลอย่างน้อย 42 ไบต์ โปรแกรมสามารถตั้งค่า Bit End of Message ได้ใน ส่วน Connection Control ที่ส่วนหัวของกลุ่มข้อมูล และจำเป็นต้องกำหนดค่าใน dataStreamType ด้วย ส่วนอื่นๆ ของส่วนหัวกลุ่มข้อมูลก็จะถูกกำหนดโดย SPX เอง

เริ่มต้นที่ SPX กำหนดค่าที่ไม่ใช่ 0 ใน InUseFlag เพื่อชี้ว่า ECB กำลังส่งกลุ่มข้อมูลและ SPX จะทำการจัดลำดับการส่งข้อมูลเอง เมื่อ SPX ทำการส่งข้อมูลแล้ว SPX จะเก็บค่า Completion ในส่วน CompletionCode และจะเซตค่า InUseFlag เป็น 0 และเรียกใช้ ESR ที่ได้ถูกกำหนดไว้ใน ESRAddress ถ้าต้องการจะใช้งาน

ค่า CompletionCode ของ ECB ที่เป็นไปได้

0 (0x00) กลุ่มข้อมูลถูกส่งเรียบร้อยแล้ว และได้รับเป็นลำดับโดยการตอบรับของผู้ติดต่อได้กลับคืนมา

236 (0xEC) SPX_CONNECTION_TERMINATED
ผู้ติดต่อทำการยกเลิกการติดต่อโดยไม่มีการส่งกลุ่มข้อมูลให้รู้ SPX ก็บอกไม่ ได้ว่ากลุ่มข้อมูลที่ส่งไปจะได้รับก่อนที่การติดต่อจะยกเลิกหรือไม่

237 (0xED) SPX_TERMINATED_POORLY
การติดต่อสิ้นสุดอย่างไม่สมบูรณ์ ผู้ติดต่อฝ่ายใดฝ่ายหนึ่งใช้ฟังก์ชัน SPXAbortConnection ในการยกเลิกการติดต่อหรือผู้ติดต่อไม่ได้รับกลุ่มข้อมูลในการยกเลิกนี้ ความผิดพลาดนี้จะแสดงให้รู้ได้ถ้าฮาร์ดแวร์ของระบบเกิดความผิดพลาด หรือกลุ่มข้อมูลไม่สามารถส่งไปยังปลายทางได้

238 (0xEE) SPX_INVALID_CONNECTION
ไม่ได้ระบุค่า ConnectionIDNumber ในการสร้างการติดต่อ

252 (0xFC) SPX_SOCKET_CLOSED

253 (0xFD) SPX_MALFORMED_PACKET

FragmentCount เป็น 0 , Fragment แรกมีความยาวน้อยกว่า 42 ไบต์ , ความยาวรวมของกลุ่มข้อมูลมากกว่า 576 ไบต์ ความผิดพลาดนี้สามารถทำให้การติดต่อถูกยกเลิกได้

ซอคเก็ตที่ถูกใช้ในการติดต่อของ SPX ไม่สามารถใช้ฟังก์ชันของ IPXในการส่งรับข้อมูลโดยตรงได้ ซอคเก็ตชนิดนี้จะขึ้นอยู่กับฟังก์ชันการใช้งานของ SPX เท่านั้น

เมื่อโปรแกรมผ่านค่ากลุ่มข้อมูลหรือ ECB มาพร้อมกันหลายๆ ตัว ให้กับ SPX SPX จะทำการจัดลำดับกลุ่มข้อมูล และส่งออกไปตามลำดับที่รับเข้ามา

โปรแกรมประยุกต์สามารถที่จะสร้างการติดต่อระหว่าง 2 ซอคเก็ตที่อยู่ในจุดต่อเดียวกันได้ โปรแกรมควรจะใช้ฟังก์ชัน SPXAbortConnection (ไม่ใช่ IPXCancelEvent) ในการที่จะยกเลิกเหตุการณ์ของ SPXSendSequencedPacket

ดูเพิ่มเติม SPXListenForSequencedPacket

ตัวอย่าง

```
#include <nxt.h>
WORD connectionIDNumber ;
ECB eventControlBlock ;
SPXSendSequencedPacket (connectionIDNumber , &eventControlBlock):
```

SPXTerminateConnection

ใช้สำหรับตัดการติดต่อ SPX

ไวยากรณ์ void SPXTerminateConnection (WORD connectionIDNumber ,
ECB *eventControlBlock)

connectionIDNumber (Input) เลขที่กำหนดโดย SPX เมื่อสร้างการติดต่อได้แล้ว

eventControlBlock (Input) ตัวชี้โครงสร้าง ECB

ค่าส่งกลับ ไม่มี

ข้อกำหนด

ฟังก์ชันนี้จะยกเลิกการติดต่อของ SPX โดยส่งผ่านค่า ConnectionID และ Address ECB ให้กับ SPX หลังจากนี้ฟังก์ชันจะคืนการควบคุมให้กับโปรแกรมเดิม ในขณะที่ SPX พยายามยกเลิกการติดต่อ ก่อนที่จะทำการเรียกใช้ฟังก์ชันนี้ โปรแกรมจะต้องตั้งค่าเริ่มต้นให้กับ ESRAddress , FragmentCount , FragmentDescriptor FragmentDescriptor แรกของส่วนหัวของกลุ่มข้อมูล SPX จะต้องมีแหล่งพักข้อมูลอย่างน้อย 42 ไบต์ และ FragmentCount = 1

ConnectionIDNumber จะเป็นหมายเลขในการติดต่อสร้างโดย SPX ถ้าการติดต่อถูกสร้างด้วยฟังก์ชัน SPXEstablishConnection ก็จะได้หมายเลขนี้กลับมา แต่ถ้าใช้ SPXListenForConnection หมายเลขนี้จะได้มาจาก 2 ไบต์แรก ของ ListenForConnectionECB ในส่วนของ IPXWorkSpace หลังจากที่ได้ติดต่อแล้ว

SPX จะเก็บค่า 0xFE (TERMINATE CONNECTION) ในส่วนของ dataStreamType ซึ่งจะเป็นค่าที่บอกฝ่ายรับว่า ยกเลิกการติดต่อ SPX จะส่งกลุ่มข้อมูลนี้ไปยังจุดต่อปลายทาง และได้ค่า CompletionCode กลับคืนมา และบันทึกค่า 0x00 ไว้ในส่วนของ InUseFlag CompletionCode จะมีค่าต่างๆ ดังนี้

0 (0x00) SPX_CONNECTION_TERMINATED

236 (0xEC) TERMINATED_BY_REMOTE_PARTNER

237 (0xED) SPX_TERMINATED_POORLY

ผู้ติดต่ออาจไม่ได้รับการตอบรับการร้องขอในช่วงเวลาที่เหมาะสม การติดต่อถูกยกเลิกไป ซึ่ง SPX ไม่สามารถบอกได้ว่าผู้ติดต่อจะได้รับการร้องขอ SPXTerminatConnection หรือไม่ ความผิดพลาดนี้จะแสดงให้รู้ได้ ถ้าฮาร์ดแวร์ของระบบเกิดความผิดพลาด หรือ กลุ่มข้อมูลไม่ได้ถูกส่งไปยังปลายทาง

238 (0xEE) SPX_INVALID_CONNECTION

253 (0xFD) SPX_MALFORMED_PACKET

FragmentCount ไม่ใช่ 1 หรือ หน่วยพักข้อมูลไม่ได้มี 42 ไบต์ เนื่องจากการทำงานของ SPX เป็นแบบอะซิงค์โครนัส การร้องขอการเลิกอาจจะไม่สมบูรณ์ เมื่อ SPXTerminateConnection คืนค่ากลับมา ดังนั้นจะไม่สามารถใช้ ECB ได้จนกว่า InUseFlag จะถูกเปลี่ยนค่าเป็น 0

เมื่อ SPX ยกเลิกการติดต่อแล้ว ตำแหน่งของ ConnectionIDNumber ที่อยู่ภายในการติดต่อก็จะว่างขึ้น สำหรับการติดต่อครั้งต่อไป ถ้าโปรแกรมต้องการได้โดยใช้ฟังก์ชัน SPXEstablishConnection โปรแกรมควรใช้ฟังก์ชัน SPXAbortConnection (ไม่ใช่ IPXCancelEvent ในการยกเลิก SPXTerminateConnection)

ดูเพิ่มเติม SPXAbortconnection , SPXEstablishConnection , SPXListenForconnection

ตัวอย่าง

```
WORD  connectionIDNumber;
ECB   eventControlBlock;

SPXTerminatConnection (connectionIDNumber , &eventControlBlock);
```

บทที่ 5
การเขียนโปรแกรมและขั้นตอนในการใช้โทรศัพท์

5.1 ส่วนโปรแกรมอินเตอร์รัพท์ (Interrupt)

หลักทั่วไปของโปรแกรมอินเตอร์รัพท์

เราแยกประเภทของการอินเตอร์รัพท์ได้สองชนิด คือ ฮาร์ดแวร์อินเตอร์รัพท์ (Hardware Interrupt) เป็นอินเตอร์รัพท์ที่ถูกร้องขอโดยฮาร์ดแวร์ และ ซอฟต์แวร์อินเตอร์รัพท์ (Software Interrupt) จะถูกร้องขอจากโปรเซสให้โปรเซสหนึ่งที่กำลังทำงานภายในระบบ

เมื่อมีการร้องขออินเตอร์รัพท์ ไม่ว่าจะมาจากแหล่งใดก็ตามจะมีการแจ้งหมายเลขอินเตอร์รัพท์มาด้วยเสมอ ค่าหมายเลขอินเตอร์รัพท์นี้จะถูกนำมาเปิดตารางอินเตอร์รัพท์เวกเตอร์ และนำค่าในตารางที่ได้มาเป็นค่าตำแหน่งหน่วยความจำเริ่มต้นของอินเตอร์รัพท์ฟังก์ชัน เพื่อใช้เป็นจุดกระโดดไปทำงานของระบบ ซึ่งจะเรียกโปรแกรมที่จะกระโดดไปทำว่า โปรแกรมบริการอินเตอร์รัพท์ (Interrupt Service Routine)

หลังจากที่ระบบทำงานตามอินเตอร์รัพท์ฟังก์ชันเสร็จสิ้นแล้ว ที่คำสั่งสุดท้ายของของอินเตอร์รัพท์ฟังก์ชันจะมีคำสั่งที่ใช้กระโดดกลับไปยังจุดที่เรียกมา คำสั่งนี้จะทำให้ระบบกระโดดกลับไปทำงานในโปรเซสที่ค้างอยู่ต่อไป

ในตารางอินเตอร์รัพท์ จะมีอยู่ 256 หน่วย แต่ละหน่วยเป็น ค่าการชี้ฟังก์ชันแบบไกล ด้วยเหตุนี้ เครื่องพีซีจึงมีหมายเลขหมายเลขอินเตอร์รัพท์ได้ตั้งแต่หมายเลข 0 ถึง 255 การส่งค่าไปยังอินเตอร์รัพท์ฟังก์ชัน หรือส่งค่ากลับ จะใช้การส่งผ่านค่าทางรีจีสเตอร์ ซึ่งรายละเอียดของอินเตอร์รัพท์ฟังก์ชันที่ใช้งานบนพีซี สามารถอ่านได้จากคู่มือระบบทั่วไป

นอกจากการเขียนโปรแกรมเพื่อขออินเตอร์รัพท์จากอินเตอร์รัพท์ฟังก์ชันที่มีอยู่ในระบบแล้ว เราอาจเขียนอินเตอร์รัพท์ฟังก์ชันเองก็ได้ โดยอาจถูกเรียกโดยฮาร์ดแวร์ หรือ ซอฟต์แวร์ก็ได้ ซึ่งมีขั้นตอนการทำงานของโปรแกรมอินเตอร์รัพท์ทั่วไปดังนี้

1. กำหนดหมายเลขอินเตอร์รัพท์ที่ต้องการใช้สำหรับอินเตอร์รัพท์ฟังก์ชันเขียนขึ้น ซึ่งอาจใช้หมายเลขที่ยังไม่มีอินเตอร์รัพท์ฟังก์ชันใดใช้ หรืออาจใช้ร่วมกับหมายเลขที่มีอินเตอร์รัพท์ใช้งานอยู่ก่อนแล้ว โดยเมื่อมีการร้องขออินเตอร์รัพท์ที่ไปเรียกอินเตอร์รัพท์ฟังก์ชันเดิมก็จะเกิดการกระโดดไปทำงานในอินเตอร์รัพท์ฟังก์ชันใหม่แทน
2. เก็บค่าการชี้อินเตอร์รัพท์ฟังก์ชันเดิมที่ใช้ (ในโปรแกรมใช้คำสั่ง getvect())
3. กำหนดค่าการชี้อินเตอร์รัพท์ฟังก์ชันของตารางอินเตอร์รัพท์เวกเตอร์ในหมายเลขของการด้วยค่าการชี้อินเตอร์รัพท์ฟังก์ชันสร้างขึ้น (ในโปรแกรมใช้คำสั่ง setvect ())

4. ก่อนจะจบการทำงานของโปรแกรม จะต้องคืนค่าการชี้อินเตอร์รัพท์ฟังก์ชันเดิมแก่ตารางอินเตอร์รัพท์ มิฉะนั้น เมื่อโปรแกรมจบการทำงาน มีการนำหน่วยความจำที่โปรแกรมนั้นเคยใช้ไปให้โปรแกรมหรือโปรเซสอื่น และมีการเรียกใช้อินเตอร์รัพท์หมายเลขนี้อีก จะทำให้เกิดการกระโดดไปทำงานยังพื้นที่ที่ไม่ใช่อินเตอร์รัพท์ฟังก์ชัน (เพราะได้นำข้อมูลใหม่ไปใส่แทนแล้ว) จะทำให้ระบบเสียหายได้ แต่เราอาจไม่ต้องคืนค่าแก่ตารางอินเตอร์รัพท์ก็ได้ ถ้าโปรแกรมเป็นโปรแกรมแบบฝังตัว (resident program)

การเขียนโปรแกรม

เราใช้ส่วนโปรแกรมอินเตอร์รัพท์นี้เพื่อให้ขณะที่ใช้งานโทรศัพท์สามารถใช้งานคอมพิวเตอร์อื่นได้อีก ซึ่งในที่นี้เราใช้ฮาร์ดแวร์อินเตอร์รัพท์โดยใช้การ์ดอินเทอรัพท์ผ่านสล็อตคอมพิวเตอร์ ฮาร์ดแวร์อินเตอร์รัพท์จากการ์ดมี 2 อินเตอร์รัพท์ คือ อินเตอร์รัพท์จากการยกหูหรือวางหู และ อินเตอร์รัพท์จากการกดหมายเลข ของโทรศัพท์ทั้ง 4 เครื่อง

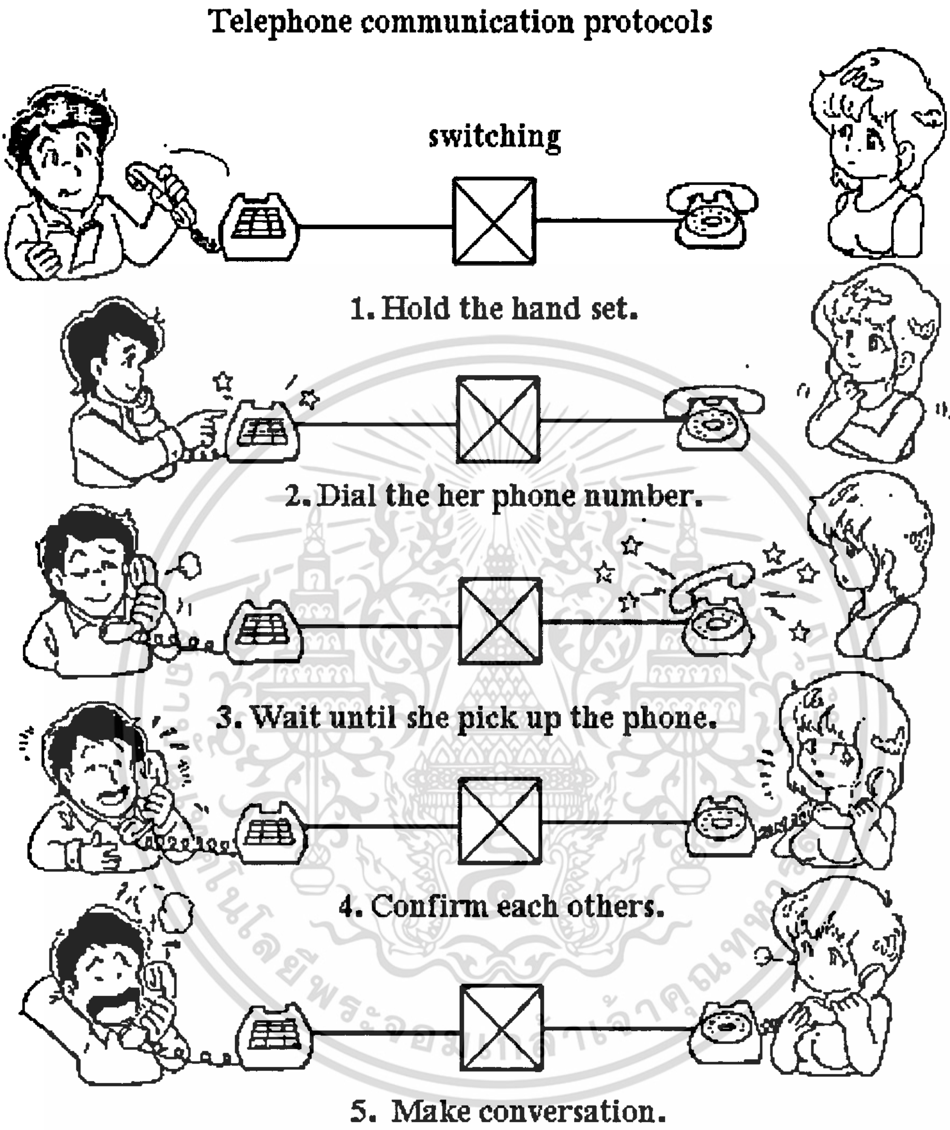
การเลือกหมายเลขอินเตอร์รัพท์ฮาร์ดแวร์ที่ใช้ทำโดยผู้ใช้สามารถเลือกได้ตามที่คอมพิวเตอร์นั้นมีหมายเลขใดที่ว่าง โดยปกติฮาร์ดแวร์อินเตอร์รัพท์ของคอมพิวเตอร์ที่ปรากฏที่สล็อตคอมพิวเตอร์มีหมายเลข 0 - 7 (IRQ0 - IRQ7) ถ้าไม่ว่างก็สามารถเลือกใช้ได้โดยต้องเก็บค่าการชี้ของอินเตอร์รัพท์ฟังก์ชันเดิมไว้ แล้วทำการกำหนดตารางเวคเตอร์อินเตอร์รัพท์ใหม่ให้ไปโปรแกรมกระโดดไปทำที่โปรแกรมบริการอินเตอร์รัพท์ (Interrupt Service Routine) ที่เราได้เขียนไว้ สำหรับโปรแกรมบริการอินเตอร์รัพท์จึงมี 2 โปรแกรมคือ โปรแกรมบริการอินเตอร์รัพท์การยกหูหรือวางหู และ โปรแกรมบริการอินเตอร์รัพท์การกดหมายเลข ซึ่งจะกล่าวถึงตัวโปรแกรมนี้ต่อไป

5.2 ขั้นตอนการใช้โทรศัพท์ในการติดต่อสื่อสาร

ขั้นตอนและกรณีต่าง ๆ ในการโทรศัพท์

- ยกหู
- ได้รับ สัญญาณให้หมุน
- กดหมายเลขที่ต้องการจะติดต่อ
- หยุดส่งสัญญาณให้หมุน
- รับหมายเลขจนครบ
- ตรวจสอบหมายเลขว่าที่ถูกต้องหรือไม่

- หมายเลขผิด
  - ได้รับสัญญาณไม่ว่าง
  - วางหู
  - หยุดส่งสัญญาณไม่ว่าง
- หมายเลขถูกต้อง
  - ตรวจสอบปลายทาง ว่าว่างหรือไม่
    - ไม่ว่าง
      - ได้รับสัญญาณไม่ว่าง
      - วางหู
      - หยุดส่งสัญญาณไม่ว่าง
    - ว่าง
      - ได้รับสัญญาณเรียกกลับ ฝ่ายตรงกันข้ามได้รับสัญญาณเรียก
        - ฝ่ายถูกเรียกยกหู
          - หยุดส่งสัญญาณเรียกและเรียกกลับ
          - ทำการต่อสาย
            - ฝ่ายใดฝ่ายหนึ่งวางหู (เมื่อโทรเสร็จ)
            - ตัดการต่อสาย
            - ฝ่ายตรงกันข้ามได้รับสัญญาณ ไม่ว่าง
            - วางหู
            - หยุดส่งสัญญาณ ไม่ว่าง
        - ฝ่ายถูกเรียกไม่ยกหู
          - วางหู
          - หยุดส่งสัญญาณเรียกและเรียกกลับ



รูปที่ 5.1 แสดงขั้นตอนต่าง ๆ ในการโทรศัพท์

กรณีต่าง ๆ เกี่ยวกับการยกหู,วางหูและการกดหมายเลขและการบวนการที่ต้องทำ

- ยกหู
  - เพื่อจะโทร
    - ส่งสัญญาณให้หมุน
  - เพื่อจะรับสาย
    - หยุดส่งสัญญาณเรียกและเรียกกลับ

- สร้างช่องทางในการติดต่อ

- วางหู
    - โทรไม่ติดและไม่ว่าง
        - หยุดส่งสัญญาณไม่ว่าง
    - โทรแล้วไม่มีคนรับสาย
        - หยุดส่งสัญญาณเรียกและเรียกกลับ
    - โทรเสร็จแล้ว (จบการสนทนา)
        - ตัดการช่องสัญญาณการติดต่อ
        - ส่งสัญญาณไม่วางให้ฝ่ายตรงกัน
    - หลังฝ่ายตรงกันข้ามวาง (โทรเสร็จ)
        - หยุดการส่งสัญญาณ ไม่ว่าง
    - กด key ไม่ครบ (กดผิด)
        - ลบหมายเลขที่เก็บไว้
- กด key
    - หยุดส่งสัญญาณให้หมุน
    - รับหมายเลข
    - ถ้ารับหมายเลขตรวจสอบความถูกต้องและปลายทางว่างหรือไม่

      ผิดหรือไม่ว่าง ส่งสัญญาณไม่ว่าง

      ถูกและว่าง ส่งสัญญาณเรียกและเรียกกลับ

บทที่ 6

การทดลอง

6.1 ขั้นตอนการทดลอง

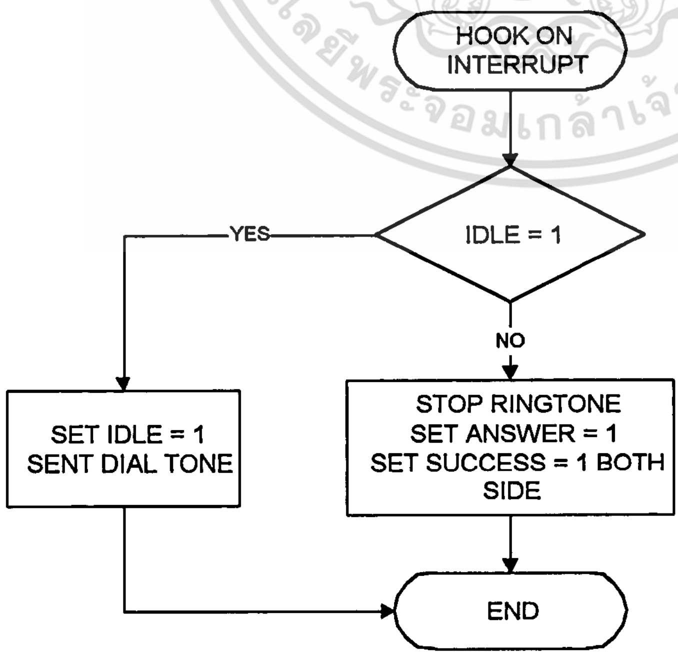
1. ศึกษาการทำงานของ IPX Protocol และ SPX Protocol และเขียนโปรแกรมรับส่งข้อมูล จากนั้นนำไปทดสอบการทำงานของโปรแกรมรับส่งข้อมูลตามต้องการ
2. ศึกษารายละเอียดการทำงานของการ์ดที่ทำให้ให้สามารถเข้ากันได้กับซอฟต์แวร์ และเขียนโปรแกรมควบคุมการทำงาน แล้วไปทดสอบและแก้ไข
3. ศึกษารายละเอียดของคอมพิวเตอร์บางส่วน ได้แก่ ส่วนสล็อตที่เสียบการ์ด
4. ศึกษาการเขียนโปรแกรมอินเตอร์รัพท์ ซึ่งเกี่ยวกับการเปลี่ยนแปลงอินเตอร์รัพท์เวกเตอร์
5. ทดสอบการทำงานการควบคุมทีละส่วน
6. นำโปรแกรมทั้งหมดในแต่ละส่วนมารวมกันเพื่อเป็นโปรแกรมหลักโปรแกรมเดียว

6.2 ลักษณะการทำงานของโครงงานนี้

สามารถแบ่งการบริการการอินเตอร์รัพท์เป็น 3 อย่างคือ

1. โปรแกรมบริการการอินเตอร์รัพท์การยกหู

สามารถออกแบบขั้นตอนการทำงานตามกรณีต่าง ๆ ในการยกหู ตามโฟลว์ชาร์ทดังนี้

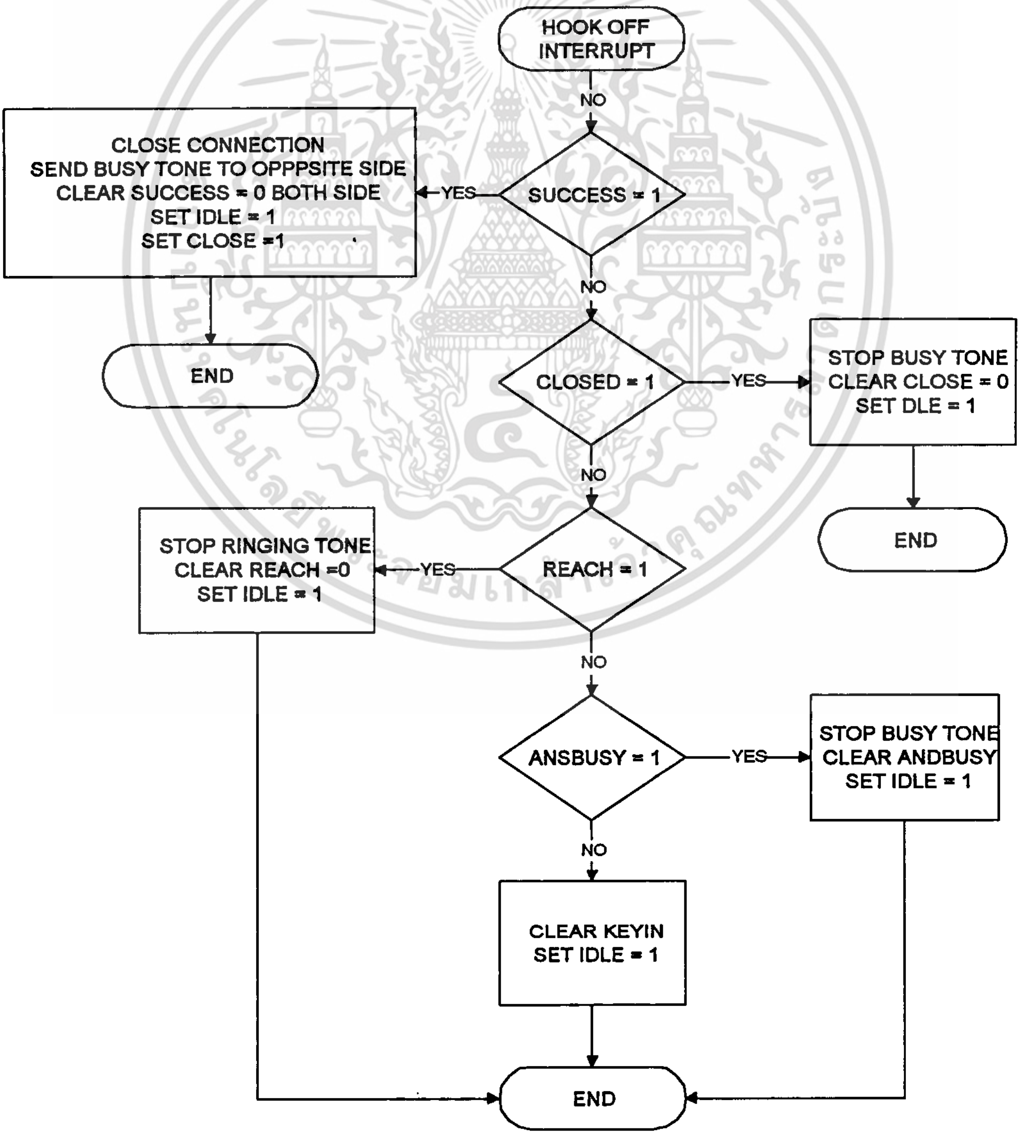


รูปที่ 6.1 โฟลว์ชาร์ทแสดงการบริการการอินเตอร์รัพท์การยกหู

จากโฟลว์ชาร์ทแสดงการบริการการอินเตอร์รัพท์การยกหูจะใช้แฟลก (Flag) ในการบอกสถานะคือ

IDLE FLAG	เท่ากับ	1 หมายถึง โทรศัพท์ว่างไม่มีการใช้งาน
		0 หมายถึง อยู่ในสถานะที่ใช้งานอยู่หรือมีผู้อื่นเรียกเข้ามา
ANSWER FLAG	เท่ากับ	1 หมายถึง มีสถานะเป็นผู้ถูกเรียก
		0 หมายถึง เป็นผู้เรียก
SUCCESS FLAG	เท่ากับ	1 หมายถึง การสร้างช่องทางการติดต่อสำเร็จแล้ว
		0 หมายถึง ไม่มีการสร้างช่องทางการติดต่อ

2. โปรแกรมบริการการอินเตอร์รัพท์การยกวางหู



รูปที่ 6.2 โฟลว์ชาร์ทแสดงการบริการการอินเตอร์รัพท์การวางหู

จากโฟลว์ชาร์ทแสดงการบริการการอินเตอร์รัพท์การยกหูจะใช้แฟลก (Flag) ในการบอกสถานะคือ

CLOSE FLAG เท่ากับ 1 หมายถึง สถานะการยกเลิกช่องทางการติดต่อหลังจากที่ ฝ่ายตรงกันข้ามวางหูก่อน

เท่ากับ 0 ไม่ได้กำหนดใช้

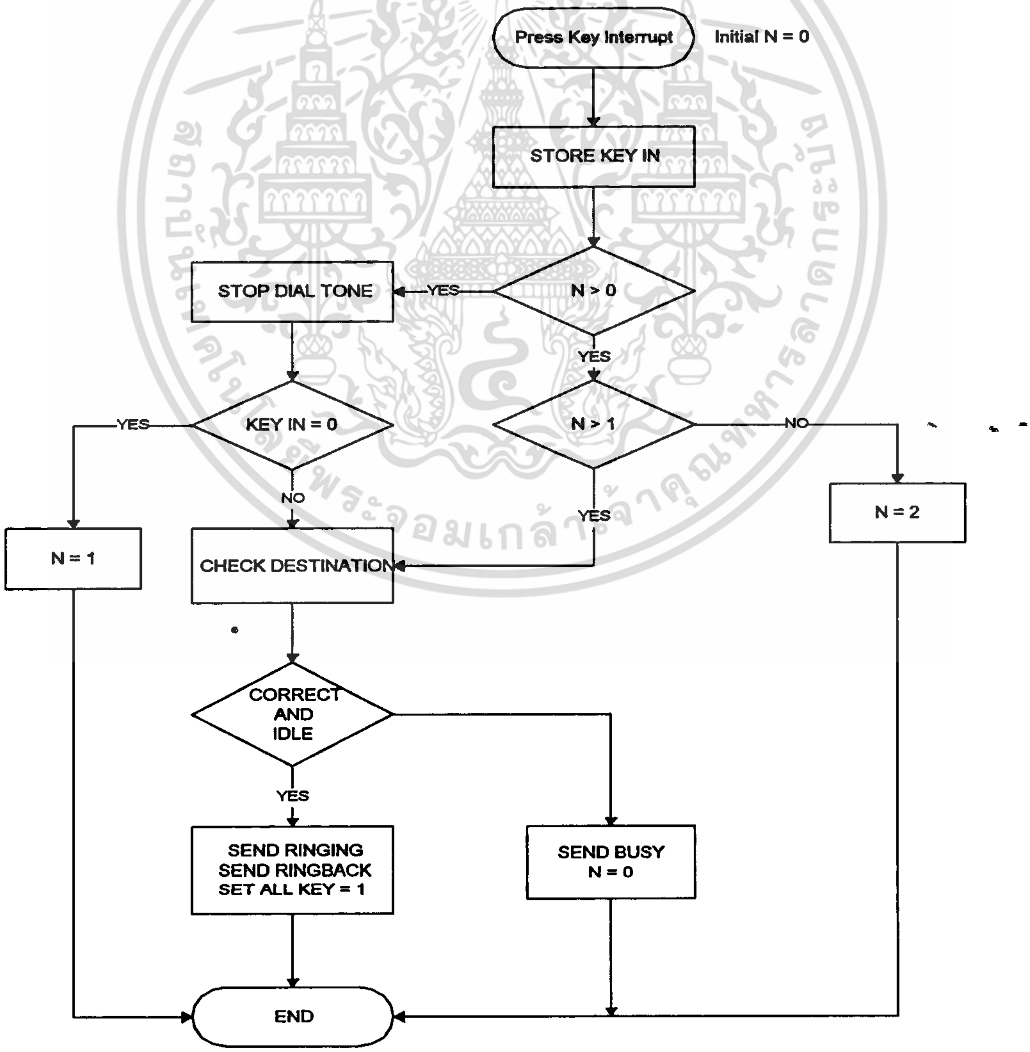
REACH FLAG เท่ากับ 1 หมายถึง สถานะที่รอฝ่ายตรงข้ามยกหู

0 ไม่ได้กำหนดใช้

ANSBUSY FLAG เท่ากับ 1 หมายถึง สถานะที่ฝ่ายถูกเรียกไม่ว่างหรือหมายเลขไม่ถูกต้อง

0 ไม่ได้กำหนดใช้

3.โปรแกรมบริการการอินเตอร์รัพท์การกดหมายเลข



รูปที่ 6.3 โฟลว์ชาร์ทแสดงขั้นตอนการทำงานบริการอินเตอร์การกดหมายเลข

จากโฟลว์ชาร์ทแสดงขั้นตอนการทำงานการบริการการกดหมายเลขมีแฟลกที่ใช้คือ

ALLKEY FLAG เท่ากับ 1 หมายถึง สถานะที่ผู้โทรศัพท์กดหมายเลขครบด้วย

0 หมายถึง การกดหมายเลขยังไม่ครบหรือยังไม่ได้กด

บทที่ 7
สรุปและวิจารณ์ผลการทดลอง

จากการทดลองใช้โปรแกรมเพื่อควบคุมการทำงานของโทรศัพท์แต่ละส่วนปรากฏว่าได้ผลดีมีข้อต้องปรับปรุงแก้ไขอยู่บ้างแต่ก็ไม่มากนัก ส่วนการทดลองส่งข้อมูลผ่านระบบเครือข่ายคอมพิวเตอร์ซึ่งได้ทำการทดลองโดยการส่งตัวอักษร (Characters) ก็สามารถทำงานได้ตามต้องการ ส่วนการควบคุมการทำงานทั้งระบบ ยังมีเสถียรภาพที่ยังไม่ดีพอ เนื่องจากในบางกรณียังเกิดข้อผิดพลาดขึ้นซึ่งต้องทำการทดลองและแก้ไขต่อไป

ปัญหาที่เกิดขึ้นมีดังนี้คือ เนื่องจากเราเปลี่ยนข้อมูลจากเสียงซึ่งเป็นสัญญาณอนาล็อก ให้เป็นสัญญาณดิจิตอล ซึ่งใช้แบบ PCM (Pulse Close Modulation) ซึ่งเราใช้อัตราการสุ่มข้อมูลเท่ากับ 8 กิโลเฮร์ต ขนาดข้อมูลเท่ากับ 8 บิต เพราะฉะนั้นข้อมูลที่ต้องส่งทั้งหมด 8 กิโลไบต์ต่อวินาที ซึ่งการใช้โทรศัพท์มีลักษณะเป็น Real Time ฉะนั้นอาจมีปัญหาในการส่งข้อมูลไม่ทันเนื่องจากในระบบเครือข่ายอาจจะมีการส่งข้อมูลที่หนาแน่น ซึ่งอาจจะแก้ปัญหานี้โดยการบีบอัดข้อมูล หรือการเพิ่มศักยภาพของเครือข่ายคอมพิวเตอร์

อีกปัญหาหนึ่งก็คือการใช้โทรศัพท์เป็นชุมสายระหว่างที่มีการใช้โทรศัพท์ต้องมีการย้ายข้อมูลตลอดเวลา ซึ่ง อาจจะไม่สามารถทำงานอย่างอื่นได้ตามที่เรากำหนดเป้าหมายเอาไว้ แนวทางการแก้ปัญหานี้อาจจะทำได้ 2 วิธี วิธีแรกเราจะตัดปัญหาโดยการให้คอมพิวเตอร์บางเครื่องทำงานเป็นชุมสายอย่างเดียว หรืออีกวิธีหนึ่งก็คือใช้คอมพิวเตอร์ที่มีประสิทธิ์ภาพสูง และ ซอฟต์แวร์ต้องพัฒนาให้ดีด้วย คือต้องสนับสนุนการทำงานหลาย ๆ งานพร้อมกันได้ (Multitask) และควรเป็นโปรแกรมลักษณะโปรแกรมฝังตัว (Resident) จะทำให้มีประสิทธิภาพสูงสุด

แนวทางการพัฒนาเนื่องการส่งข้อมูลมีลักษณะเป็นดิจิตอล ซึ่งได้เปรียบเนื่องจากปัจจุบันมีระบบอินเตอร์เน็ตซึ่งสามารถส่งข้อมูลได้อย่างง่ายดายไปทุกแห่งทั่วโลก เราอาจพัฒนาโครงงานนี้ให้เข้าได้กับมาตรฐานการส่งข้อมูลบนอินเตอร์เน็ต ซึ่งจะทำให้เราสามารถโทรศัพท์ไปได้ทั่วโลกโดยระบบอินเตอร์เน็ต อีกแนวทางหนึ่งก็คือ เราอาจพัฒนาให้เป็นการประชุมทางไกลได้โดยการส่งข้อมูลที่มีการแปลงให้เป็นสัญญาณดิจิตอลแล้วส่งไปด้วย กับข้อมูลเสียง

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้ประสบความสำเร็จลุล่วงไปได้ด้วยดีนั้น ต้องขอขอบคุณ อาจารย์สุรสิทธิ์ วรรณไกรโรจน์ ในฐานะอาจารย์ที่ปรึกษา และขอขอบคุณ อาจารย์ อัครินทร์ คุณกิตติ ที่ให้คำแนะนำ จัดหาอุปกรณ์เครื่องมือต่าง ๆ รวมทั้งห้องปฏิบัติการในการทำงานครั้งนี้ และขอขอบคุณ บริษัท 401 AUTOMATION ที่ให้ความอนุเคราะห์ ให้ไอซีบางตัวมาใช้งาน

หนังสืออ้างอิง

1. Network Programming in C,Barry Nance,QUE Corporation.,1990
2. Netware C Interface-Dos , Novell Incorporated .
3. Programmer's Guide to Netware, Charles G.Rose , Mcgraw-Hill Inc.,1990
4. ระบบเครือข่ายคอมพิวเตอร์ LAN, อัครเสน สุนทรผ่อง, จักร พิชัยสรทัต, บริษัท ซีเอ็ดยูเคชั่น จำกัด.
5. คู่มือการใช้ Netware , ต้น ตัณฑ์สิทธิวงศ์, สุพจน์ ปุณณชัยยะ.
6. การเขียนภาษาซี, ธันวา ศรีประมง.

```
#include <dos.h>
#include <stdio.h>
#define CtrlSwitch          0x284
#define DtmfKey 1
#define MaxR 1
#define MaxN 1
#define MaxC 1
#define MaxT 4
#define TCK 0x0B
#define ctrl1 0x283
#define ctrl2 0x287
#define CRS   0x286
#define Std_H_int
#define Tel_1add 0x288
#define Tel_2add 0x289
#define Tel_3add 0x28a
#define Tel_4add 0x28b
#define LINT_ARGS           /* To allow for MSC protype inclusion    */
#define SOCKET   0x4141
#define SOCKET1   0x4242
#define SOCKET2   0x4343
#define SOCKET3   0x4444
#define SOCKET4   0x4545
#define NUM_ECBS 6
#define ESCAPE_KEY 0x1B
#define RETURN_KEY 0x0D
#define MAX_RECEIVE_ECBs 6
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include <dos.h>
#ifdef LC60
#include <dos.h>
#else
#include <conio.h>
#endif
#ifdef __TURBOC__
#include <alloc.h>
#endif
#ifndef LC60
#ifndef __TURBOC__
#include <malloc.h>
```

```
#endif
#endif
#include <nxt.h>            /* API headers */
#include <nit.h>

/**********************************************************************
     function prototype
**********************************************************************/
void ini(void);
void interrupt ISR(void);
void interrupt(*oldintfunction)(void);
void Trancievedata(void);
void Std_Hint (char std_h_flg);
void IsrDTMF(char IdTel,char Number);
void Busy(void);
void Tone(char a,char b,char c);
void ISRHook(char IdTel);
void Make_Connect(char IdTel);
void Clear_Connect(char IdTel);
void Ringing(char IdTel,char Type);
void PreConnect(char MyIdTel,char AnsIdTel);
void Init_LAN(char);
void Send(char,char[3],char[3],char[3],char[3]);//new
void SendData1(char,char[5][100]);
void SendData2(char,char[5][100]);
void SendData3(char,char[5][100]);
void SendData4(char,char[5][100]);
void Requesting(char);
void NowSuccess(char);
void IsrRequest(char[10]); //new
void IsrAcknowIdle(char[10]);//new
void IsrAcknowBusy(char[10]);//new
void IsrLAN(char);//new
void IsrSuccess(char[10]);// ***New***
void IsrClose(char[10]);
void ending(void);
void Initialize(void) ;
void Usage( void ) ;
void MakeConnection(WORD *) ;
void DoListen( WORD * ) ;
void CreateReceiveECBsandPost(void) ;
void CreateConnectionECBandHeader(void) ;
void CreateSendECBandHeader( void ) ;
void CreateReceiveECBandHeader( void ) ;
void CreateListenECBandHeader( void ) ;
```

```
void SendReceiveChar(WORD,WORD,WORD,WORD,WORD) ;
void ReceiveChar( void ) ;
void Terminate( WORD );
void Initialize1(void) ;
void MakeConnection1(WORD * ) ;
void DoListen1( WORD * ) ;
void CreateReceiveECBsandPost1( void ) ;
void CreateConnectionECBandHeader1(void);
void CreateSendECBandHeader1( void ) ;
void CreateReceiveECBandHeader1( void ) ;
void CreateListenECBandHeader1( void ) ;
void Terminate1( WORD ) ;
void Initialize2(void) ;
void MakeConnection2(WORD * ) ;
void DoListen2( WORD * ) ;
void CreateReceiveECBsandPost2( void ) ;
void CreateConnectionECBandHeader2(void);
void CreateSendECBandHeader2( void ) ;
void CreateReceiveECBandHeader2( void ) ;
void CreateListenECBandHeader2( void ) ;
void Terminate2( WORD ) ;
void Initialize3(void) ;
void MakeConnection3(WORD * ) ;
void DoListen3( WORD * ) ;
void CreateReceiveECBsandPost3( void ) ;
void CreateConnectionECBandHeader3(void);
void CreateSendECBandHeader3( void ) ;
void CreateReceiveECBandHeader3( void ) ;
void CreateListenECBandHeader3( void ) ;
void Terminate3( WORD ) ;
void Initialize4(void) ;
void MakeConnection4(WORD * ) ;
void DoListen4( WORD * ) ;
void CreateReceiveECBsandPost4( void ) ;
void CreateConnectionECBandHeader4(void);
void CreateSendECBandHeader4( void ) ;
void CreateReceiveECBandHeader4( void ) ;
void CreateListenECBandHeader4( void ) ;
void Terminate4( WORD ) ;
void far cdecl ReceiveESRHandler( ) ;
void cdecl ProcessReceivedData( ECB * ) ;
void far cdecl ReceiveESRHandler1( ) ;
void cdecl ProcessReceivedData1( ECB * ) ;
void far cdecl ReceiveESRHandler2( ) ;
void cdecl ProcessReceivedData2( ECB * ) ;
```

```
void far cdecl ReceiveESRHandler3( ) ;
void cdecl ProcessReceivedData3( ECB * ) ;
void far cdecl ReceiveESRHandler4( ) ;
void cdecl ProcessReceivedData4( ECB * ) ;
volatile char errorint;

/*********************************************************************
        Global variable
*********************************************************************/
typedef struct {
                char Hook;
                char STD ;
                char Idle;
                char Success;
                char Reach;
                char AnsBusy;
                char Close;
                char AllKey;
                char count;
                char CallNum[5];
                char Local;
                char Outbroad;
                char EndKey;
                char RingFlag;
                } TelStruct;
TelStruct Tel[5];
typedef struct {
                char Message;
                char IdTel;
                char Status;
                }S_Hflg;
volatile S_Hflg  StdP , HookP;

char HookMassage;
char IntLANFlag;
char StdMassage;
struct {
        char IdTel;
        char Req;
        char Even;
        char Time;
    }Timer;
struct {
        char Flg;
        char IdTel;
```

```
      } ConReq;
char DataInTel[5][100],DataOutTel[5][100];
int count[5];
char *Pin1,*Pin2,*Pin3,*Pin4,*Pout[5];
char IntClock;
ECB *sendECB, *receiveECB, *connectionECB, *listenECB ;
SPXHeader *receiveHeader, *sendHeader  ;
ECB *sendECB1, *receiveECB1, *connectionECB1, *listenECB1 ;
SPXHeader *receiveHeader1, *sendHeader1  ;
ECB *sendECB2, *receiveECB2, *connectionECB2, *listenECB2 ;
SPXHeader *receiveHeader2, *sendHeader2  ;
ECB *sendECB3, *receiveECB3, *connectionECB3, *listenECB3 ;
SPXHeader *receiveHeader3, *sendHeader3  ;
ECB *sendECB4, *receiveECB4, *connectionECB4, *listenECB4 ;
SPXHeader *receiveHeader4, *sendHeader4  ;
WORD fSocket ;          /* Fileserver socket number.                */
int  socket;          /* Socket number used by this program.         */
int  socket1;
int  socket2;
int  socket3;
int  socket4;
char command[9] ;          /* Char. buffer for sent and received data.    */
char data1[500];
char data2[500];
char data3[500];
char data4[500];
BYTE done ;            /* Loop flag to break loop.                  */
          int connectionNumber ;
          WORD spxConnectionID ;
          int connectionNumber1 ;
          WORD spxConnectionID1 ;
          int connectionNumber2 ;
          WORD spxConnectionID2 ;
          int connectionNumber3 ;
          WORD spxConnectionID3 ;
          int connectionNumber4 ;
          WORD spxConnectionID4 ;
```

```
#include <dos.h>
#include <stdio.h>
#include "project.c"
#include <conio.h>

void main(void)
{
        char n ;
        char run=1;
        ini();      //INITIAL 8255 PORT
        IntLANFlag=0;
        for (n=1;n<5;n++)
        {
                Tel[n].Hook=0;          // Hook off = 0
                Tel[n].STD=0;           // Get First Key = 0
                Tel[n].Idle=1;          // No Operation = 1
                Tel[n].Success=0;       // Connect Success = 0
                Tel[n].Reach=0;         // Send Ringing&Get ringback = 0
                Tel[n].AnsBusy=0;       // Answer side is busy = 0
                Tel[n].Close=0;         // Hook on After = 0
                Tel[n].AllKey=1;        // Get all Key = 1
                Tel[n].EndKey=0;
                Tel[n].count=0;         // Count of Number = 0
                Tel[n].CallNum[0]=9;
                Tel[n].CallNum[1]=0;
                Tel[n].CallNum[2]=0;
                Tel[n].CallNum[3]=0;
                Tel[n].CallNum[4]=0;
                Tel[n].Local=1;
                Tel[n].RingFlag=0;

        }
        IntClock=0;
        Pin1=(DataInTel[1]);
        Pin2=(DataInTel[2]);
        Pin3=(DataInTel[3]);
        Pin4=(DataInTel[4]);
        oldintfunction = getvect(TCK);
        setvect(TCK,ISR);
        asm {STI}
        asm {
            IN  AL,21H
            AND AL,11110111B
            OUT 21H,AL
        }
```

```
	outportb(0x282,0x00);   //Reset Hook/Std interrupt
	outportb(0x282,0x01);   //Set Hook/Std interrupt
	Init_LAN(0x4c); // 4c=L  , 53=S

	while (run)
	{
		if(StdP.Message)
		{
			printf("Now Tel %x you press a key %x\n",StdP.IdTel,StdP.Status);
			if(Tel[StdP.IdTel].AllKey&!Tel[StdP.IdTel].Success)
			IsrDTMF(StdP.IdTel,StdP.Status);
			else printf("Your key is error plese on hook\n");
			printf("Ok Tel %x your Key are
%x.%x.%x.%x.%x\n",StdP.IdTel,Tel[StdP.IdTel].CallNum[0],Tel[StdP.IdTel].CallNum[1],Tel[StdP.IdTel].Call
Num[2],Tel[StdP.IdTel].CallNum[3],Tel[StdP.IdTel].CallNum[4]);
			StdP.Message=0;
		}
		if(HookP.Message)
		{
			if(!Tel[HookP.IdTel].Hook)
				printf("Hello Tel %x\n",HookP.IdTel);
			else
				printf("Bye.. Tel %x\n",HookP.IdTel);
			HookP.Message=0;
			ISRHook(HookP.IdTel);

		}

		if(IntLANFlag)
		{
			IsrLAN(IntLANFlag);
			IntLANFlag=0;
		}
		if(kbhit())
		{
			run = 0;
		}
	}
}
```

```c
#include <stdio.h>
#include <dos.h>
#include "pabx1.h"
#define Request 0x11   //new
#define AcknowIdle 0x22//new
#define AcknowBusy 0x33//new
#define Successed 0x44 //new
#define Closed 0x55    //new

void ini(void)
{  asm { // initial 8255
          mov al,0x9a //10011000B set pa:in,pb:out,pclow:in,pchigh:out
          mov dx,ctrl1
          out dx,al
          mov al,0x80 //10000000B set pa:out,pb:out,pc:out
          mov dx,ctrl2
          out dx,al
          mov al,0xff // initial STD and Ringing
          mov dx,CRS
          out dx,al
     }
   outportb(0x286,0xff);
}

void interrupt ISR(void)
{
          char  clockflg=0 ;
          char std_h_flg=0;
          //printf("You get int \n");
          clockflg=inportb(0x280)&0x10; //check clock interrupt
          std_h_flg=inportb(0x281);     //check std/hook interrupt
          //printf("flag is %x\n",std_h_flg);
          if (clockflg&&std_h_flg)
          {
                    Trancievedata();
                    Std_Hint(std_h_flg);
          }
          else
                    if (clockflg)
                      Trancievedata();
                    else
                              if (std_h_flg)
                                Std_Hint(std_h_flg);
                              else errorint = 1;
```

```
        asm { MOV AL,20H      //enable new interrupt
              OUT 20H,AL
            }
        outportb(0x282,0x00);
        if (IntClock)
        {
                outportb(0x282,0x03);
        }
        else
        outportb(0x282,0x01);
}

void Std_Hint (char std_h_flg)
{
        char std_0,std_1,std_2,std_3,std_4;
        char h_int0,h_int1,h_int2,h_int3,h_int4,hst;
        std_0=std_h_flg&0xf0;
        h_int0=std_h_flg&0x0f;
        if(std_0)
        {
                StdP.Message=1;
                std_1=std_0&0x80;
                std_2=std_0&0x40;
                std_3=std_0&0x20;
                std_4=std_0&0x10;
                if(std_1)
                {
                        StdP.IdTel=1;
                        outportb(0x285,0x10);
                        StdP.Status=inportb(0x280)&0x0f;
                }
                else    if(std_2)
                        {
                                StdP.IdTel=2;
                                outportb(0x285,0x20);
                                StdP.Status=inportb(0x280)&0x0f;
                        }
                        else    if(std_3)
                                {
                                        StdP.IdTel=3;
                                        outportb(0x285,0x40);
                                        StdP.Status=inportb(0x280)&0x0f;
                                }
                                else    if(std_4)
                                        {
```

```
                    StdP.IdTel=4;
                    outportb(0x285,0x80);
                    StdP.Status=inportb(0x280)&0x0f;
                }
                else
                {
                    StdP.IdTel=0;
                    StdP.Message=0;
                    StdP.Status=1;
                }
}
if(h_int0)
{
    HookP.Message=1;
    hst=inportb(0x282)&0xf0;
    h_int1=h_int0&0x01;
    h_int2=h_int0&0x02;
    h_int3=h_int0&0x04;
    h_int4=h_int0&0x08;
    if(h_int1)
    {
        HookP.IdTel=1;
        Tel[1].Hook=(hst&0x10)>>4;
    }
    else    if(h_int2)
            {
                HookP.IdTel=2;
                Tel[2].Hook=(hst&0x20)>>5;
            }
            else    if(h_int3)
                    {
                        HookP.IdTel=3;
                        Tel[3].Hook=(hst&0x40)>>6;
                    }
                    else    if(h_int4)
                            {
                                HookP.IdTel=4;
                                Tel[4].Hook=(hst&0x80)>>7;
                            }
                            else
                            {
                                HookP.IdTel=0;
                                HookP.Message=0;
                                HookP.Status=1;
                            }
```

```
        }
}

void Busy(void)
{
        printf("Your Key is error!\n");
}

void Tone(char a,char b,char c)
{
        char data,data1;
        switch(c)
        {
                case 0: switch(b)
                        {
                                case 1:switch(a)
                                {
                                        case 1:{data=0x20; data1=0x00; printf("Tel 1 not get dial
tone now\n");break;}
                                        case 2:{data=0x24; data1=0x04;printf("Tel 2 not get dial
tone now\n");break; }
                                        case 3:{data=0x28; data1=0x08; printf("Tel 3 not get dial
tone now\n ");break; }
                                        case 4:{data=0x2c; data1=0x0c; printf("Tel 4 not get dial
tone now\n ");break;}
                                };break;
                                case 2: switch(a)
                                {
                                        case 1:{data=0x21;data1=0x01;printf("Tel 1 not get busy
tone now\n");break;}
                                        case 2:{data=0x25;data1=0x05;printf("Tel 2 not get busy
tone now\n");break;}
                                        case 3:{data=0x29;data1=0x09;printf("Tel 3 not get busy
tone now\n");break;}
                                        case 4:{data=0x2d;data1=0x0d;printf("Tel 4 not get busy
tone now\n");break;}
                                };break;
                                case 3: switch(a)
                                {
                                        case 1:{data=0x22;data1=0x02;printf("Tel 1 not get
ringback tone now\n");break;}
                                        case 2:{data=0x26;data1=0x06;printf("Tel 2 not get
ringback tone now\n");break;}
                                        case 3:{data=0x2a;data1=0x0a;printf("Tel 3 not get
ringback tone now\n");break;}
```

```
                                        case 4:{data=0x2e;data1=0x0e;printf("Tel 4 not get
ringback tone now\n");break;}
                                };break;
                        };break;
                case 1: switch(b)
                        {
                                case 1:switch(a)
                                {
                                        case 1:{data=0x30;data1=0x10;printf("Tel 1 get dial tone
now\n");break;}
                                        case 2:{data=0x34;data1=0x14;printf("Tel 2 get dial tone
now\n");break;}
                                        case 3:{data=0x38;data1=0x18;printf("Tel 3 get dial tone
now\n");break;}
                                        case 4:{data=0x3c;data1=0x1c;printf("Tel 4 get dial tone
now\n");break;}
                                };break;
                                case 2: switch(a)
                                {
                                        case 1:{data=0x31;data1=0x11;printf("Tel 1 get busy
tone now\n");break;}
                                        case 2:{data=0x35;data1=0x15;printf("Tel 2 get busy
tone now\n");break;}
                                        case 3:{data=0x39;data1=0x19;printf("Tel 3 get busy
tone now\n");break;}
                                        case 4:{data=0x3d;data1=0x1d;printf("Tel 4 get busy tone
now\n");break;}
                                };break;
                                case 3: switch(a)
                                {
                                        case 1:{data=0x32;data1=0x12;printf("Tel 1 get ringback
tone now\n");break;}
                                        case 2:{data=0x36;data1=0x16;printf("Tel 2 get ringback
tone now\n");break;}
                                        case 3:{data=0x3a;data1=0x1a;printf("Tel 3 get ringback
tone now\n");break;}
                                        case 4:{data=0x3e;data1=0x1e;printf("Tel 4 get ringback
tone now\n");break;}
                                };break;
                        };break;
        }
                outportb(CtrlSwitch,data);
                outportb(CtrlSwitch,data1);
}
```

```
void IsrDTMF(char IdTel,char Number)
{
        if (!Tel[IdTel].Idle)
        {
                if (!Tel[IdTel].count)          //First Key
                {
                        Tone(IdTel,1,0);
                        Tel[IdTel].STD=1;
                        if (Number!=0xa)        //First Key not '0'
                        {
                                Tel[IdTel].count=3;
                                Tel[IdTel].CallNum[Tel[IdTel].count]=Number;
                        }
                        else
                        {
                        Tel[IdTel].CallNum[0]=Number;   //First Key is'0'
                        Tel[IdTel].Local=0;
                        }
                }

                else Tel[IdTel].CallNum[Tel[IdTel].count]=Number;
                //printf("count of Tel%x is %x\n",IdTel,Tel[IdTel].count);
                Tel[IdTel].count++;
                Timer.Req=1;
                Timer.Time=10;
                Timer.IdTel=IdTel;
                Timer.Even=DtmfKey;
                switch (Tel[IdTel].count)
                {
                        case 2 :{               if((Tel[IdTel].CallNum[1] >
MaxR) | | (Tel[IdTel].CallNum[1]==0))
                                                {
                                                        printf("Tel %x your 2nd key is error please hook
on",IdTel);
                                                        Tone(IdTel,2,1);
                                                        Tel[IdTel].AllKey=0;
                                                }
                                                break;
                                }
                        case 3 :{               if((Tel[IdTel].CallNum[2] >
MaxN) | | (Tel[IdTel].CallNum[2]==0))
                                                {
                                                        printf("Tel %x your 3rd key is error please hook
on",IdTel);
                                                        Tone(IdTel,2,1);
```

```
                        Tel[IdTel].AllKey=0;
                    }
                    break;
                }
                case 4 :{       if((Tel[IdTel].CallNum[3] >
MaxC) | | (Tel[IdTel].CallNum[3]==0))

                    {
                        printf("Tel %x your 4th key is error please hook
on",IdTel);

                        Tone(IdTel,2,1);
                        Tel[IdTel].AllKey=0;
                    }
                    break;
                }
                case 5 :{       if((Tel[IdTel].CallNum[4] >
MaxT) | | (Tel[IdTel].CallNum[4]==0))

                    {
                        printf("Tel %x your 5th key is error please hook
on",IdTel);

                        Tone(IdTel,2,1);
                    }
                    else
                    {
                        ConReq.Flg=1;
                        ConReq.IdTel=IdTel;
                        if(Tel[IdTel].Local)
                        PreConnect(IdTel,Tel[IdTel].CallNum[4]);
                        else
                        {
                        printf(" Now call Request\n ");
                        Requesting(IdTel);
                        }
                    }
                    Timer.Req=1;
                    Timer.Time=0;
                    Tel[IdTel].count=0;
                    Tel[IdTel].AllKey=0;
                    Tel[IdTel].EndKey=1;
                }
        }

    }
}
```



```
void ISRHook ( char IdTel )
```

```
{
    if(!Tel [IdTel].Hook)                   //Hook off
    {
        if (Tel[IdTel].Idle)
        {
            Tone (IdTel,1,1);
            Tel[IdTel].Idle=0;
        }
        else
        {
            if(Tel[IdTel].Local)
            {
            printf("You are Ans \n" );
            Tone(Tel[IdTel].CallNum[4],3,0);
            Ringing (IdTel,0);
            }
            else
            {
            printf("You are Ans \n");
            NowSuccess(IdTel);
            }
            Make_Connect (IdTel);

        }
    }
    else
    {                                       //Hook on
        if (Tel[IdTel].AnsBusy)
        {

            Tone(IdTel,2,0);  //stop Busy
            Tel[IdTel].AnsBusy=0;
        }
        else
        {
            if(Tel[IdTel].Success)
            {
                Clear_Connect (IdTel);
            }
            else
            {
                if(Tel[IdTel].Close)
                {
                    Tone(IdTel,2,0);
                    Tel[IdTel].Close=0;
```

```
                        }
                        else
                        {
                                if(Tel[IdTel].Reach)
                                {
                                        printf("You on Hook");
                                        Tel[Tel[IdTel].CallNum[4]].Idle=1;
                                        Tel[IdTel].Reach=0;
                                        Tel[Tel[IdTel].CallNum[4]].RingFlag=0;
                                        Ringing(Tel[IdTel].CallNum[4],0); //stop Ringing
                                        Tone(IdTel,3,0); //stop Ringback
                                }
                                else
                                {
                                        if (!Tel[IdTel].STD)
                                        {
                                                Tone(IdTel,1,0);
                                        }
                                        else if(!Tel[IdTel].EndKey)
                                            {
                                                Tone(IdTel,2,0) ;
                                            }
                                }
                        }
                }
            }
        Tel[IdTel].Idle=1;
        Tel[IdTel].CallNum[0]=9;
        Tel[IdTel].CallNum[1]=0;
        Tel[IdTel].CallNum[2]=0;
        Tel[IdTel].CallNum[3]=0;
        Tel[IdTel].CallNum[4]=0;
        Tel[IdTel].AllKey=1;
        Tel[IdTel].count=0;
        Tel[IdTel].Reach=0;
        Tel[IdTel].STD=0;
        Tel[IdTel].EndKey=0;
        Tel[IdTel].Local=1;
        }
}

void Ringing (char IdTel,char Type)
{
        int data,t1,t2,t3,t4;
        Tel[IdTel].RingFlag=Type;
```

```
	if (Tel[1].RingFlag)
		t1=0xef;
	else t1=0xff;
	//printf("RingFlag of Call 1 is %x\n",Tel[1].RingFlag);
	if (Tel[2].RingFlag)
		t2=0xdf;
	else t2=0xff;
	//printf("RingFlag of call 2 is %x\n",Tel[2].RingFlag);
	if (Tel[3].RingFlag)
		t3=0xbf;
	else t3=0xff;
	//printf("RingFlag of call 3 is %x\n",Tel[3].RingFlag);
	if (Tel[4].RingFlag)
		t4=0x7f;
	else t4=0xff;
	//printf("RingFlag of Call 4 is %x\n",Tel[4].RingFlag);
	data=(t1&t2&t3&t4);
	//printf("Ringing is %x\n",data);
	outportb(CRS,data);
	if (Type)
		printf("Tel %x you have Ringing\n",IdTel);
	else
		printf("Tel %x you  don't have Ringing\n",IdTel);
}

void Clear_Connect (char IdTel)
{
	printf("Tel %x you have disconnect\n",IdTel);
	Tone(Tel[IdTel].CallNum[4],2,1);
	Tel[Tel[IdTel].CallNum[4]].Close=1;
	Tel[Tel[IdTel].CallNum[4]].Success=0;
	Tel[IdTel].Success=0;
	if (!(Tel[1].Success||Tel[2].Success||Tel[3].Success||Tel[4].Success))
	    {
		IntClock=0;
		printf("Chennal close");
	    }
}

void PreConnect(char MyIdTel,char AnsIdTel)
{
	if (Tel[MyIdTel].Local)
	{
		if (Tel[AnsIdTel].Idle)
		{
```

```
                    Tone(MyIdTel,3,1);     // Ans OK
                    Tel[MyIdTel].Reach=1;
                    Tel[AnsIdTel].Idle=0;
                    Tel[AnsIdTel].CallNum[4]=MyIdTel;
                    Ringing(AnsIdTel,1);
            }
            else
            {
                    Tone(MyIdTel,2,1);        // Ans Busy
                    Tel[MyIdTel].AnsBusy=1;
            }
        }
        else
        {
         // about network
        }
}

void Make_Connect(char IdTel)
{
        printf("Tel %x you have connect\n",IdTel);
        Tel[IdTel].Success=1;
        Tel[Tel[IdTel].CallNum[4]].Success=1;
        Tel[Tel[IdTel].CallNum[4]].Reach=0;
        printf("Tel%xSucess",IdTel);
        if(!Tel[IdTel].Local)
        {
                Pout[IdTel]=DataOutTel[IdTel];
                count[IdTel]=100;
        }
        else
        {
                Pout[IdTel]=DataInTel[Tel[IdTel].CallNum[4]];
                Pout[Tel[IdTel].CallNum[4]]=DataInTel[IdTel];


        }
        if(!IntClock)
        {
                printf("open clock int.\n");
                outportb(0x282,0x03);
                IntClock=1;
        }
}
```



```
void Trancievedata()
```

```
{
        if(Tel[1].Success)
        {
                *Pin1=inportb(Tel_1add);
                outportb(Tel_1add,*Pout[1]);
                if (!Tel[1].Local)
                {
                        if(count[1])
                        {
                                Pin1++;
                                Pout[1]++;
                                count[1]--;
                        }
                        else
                        {
                                SendData1(1,DataInTel[1]);
                                count[1]=100;
                        }
                }
        }
        if(Tel[2].Success)
        {
                *Pin2=inportb(Tel_2add);
                outportb(Tel_2add,*Pout[2]);
                if (!Tel[2].Local)
                {
                        if(count[2])
                        {
                                Pin2++;
                                Pout[2]++;
                                count[2]--;
                        }
                        else
                        {
                                SendData2(2,DataInTel[2]);
                                printf("send data\n");
                                count[2]=100;
                        }
                }
        }
        if(Tel[3].Success)
        {
                *Pin3=inportb(Tel_3add);
                outportb(Tel_3add,*Pout[3]);
                if (!Tel[3].Local)
```

```
        {
                if(count[3])
                {
                        Pin3++;
                        Pout[3]++;
                        count[3]--;
                }
                else
                {
                        SendData3(3,DataInTel[3]);
                        count[3]=100;
                }
        }
    }
    if(Tel[4].Success)
    {
        *Pin4=inportb(Tel_4add);
        outportb(Tel_4add,*Pout[4]);
        if (!Tel[4].Local)
        {
                if(count[4])
                {
                        Pin4++;
                        Pout[4]++;
                        count[4]--;
                }
                else
                {
                        SendData4(4,DataInTel[4]);
                        count[4]=100;
                }
        }
    }
}

void IsrLAN(char IntLANFlag)
{
switch (IntLANFlag)
{
case 0x11:IsrRequest(command);    break;
case 0x22:IsrAcknowIdle(command); break;
case 0x33:IsrAcknowBusy(command); break;
case 0x44:IsrSuccess(command);    break;
case 0x55:IsrClose(command);      break;
}
```

```
}

void Requesting(char Id)
{
 char Sstation[3],Stel[3],Dstation[3],Dtel[3];
 Stel[1]=1;// number card
 Stel[2]=Id;
 Dtel[1]=1;// number card
 Dtel[2]=Tel[Id].CallNum[4];
 Send(Request,Sstation,Stel,Dstation,Dtel);
}

void Send(char output,char Sstation[3],char Stel[3],char Dstation[3],char Dtel[3])
{
 command[0]=output; // 11 command to request
 command[1]=Sstation[1];
 command[2]=Sstation[2];
 command[3]=Stel[1];
 command[4]=Stel[2];
 command[5]=Dstation[1];
 command[6]=Dstation[2];
 command[7]=Dtel[1];
 command[8]=Dtel[2];
 SPXSendSequencedPacket(spxConnectionID,sendECB);
}

void SendData1(char Id,char DataInTel[1][100])
{
 SPXSendSequencedPacket(spxConnectionID1,sendECB1);
}

void SendData2(char Id,char DataInTel[2][100])
{
 SPXSendSequencedPacket(spxConnectionID2,sendECB2);
}

void SendData3(char Id,char DataInTel[3][100])
{
 SPXSendSequencedPacket(spxConnectionID3,sendECB3);
}

void SendData4(char Id,char DataInTel[4][100])
{
 SPXSendSequencedPacket(spxConnectionID4,sendECB4);
}
```

```
void IsrRequest(char buffer[10])
{
char Sstation[3],Stel[3],Dstation[3],Dtel[3];
printf("Now I have request from stel-%02x%02x, dtel-%02x%02x\n",
    buffer[3],buffer[4],buffer[7],buffer[8]);
if (Tel[buffer[8]].Idle==1)
  {
  Tel[buffer[8]].Idle=0;// set to busy not idle
  Tel[buffer[8]].Local=0;// set to not local called
  Tel[buffer[8]].CallNum[3]=1;
  Tel[buffer[8]].CallNum[4]=buffer[4];
  printf("requesting from %02x",buffer[4]);
  Tel[buffer[8]].RingFlag=1;
  Ringing(buffer[8],1);//send ringing to itseft
  Stel[1]=buffer[7];
  Stel[2]=buffer[8];
  Dtel[1]=buffer[3];
  Dtel[2]=buffer[4];
  Send(AcknowIdle,Sstation,Stel,Dstation,Dtel);// acknow and Idle back to calling
  }
else
  {
  Send(AcknowBusy,Dstation,Dtel,Sstation,Stel);// acknow and Busy back to calling
  }
}

void IsrAcknowIdle(char buffer[10])
{
 printf("now I have AcknowIdle\n");

 Tone(buffer[8],3,1);
 Tel[buffer[8]].Reach=1;
}

void IsrAcknowBusy(char buffer[10])
{
 Tone(buffer[8],2,1);
 Tel[buffer[8]].AnsBusy=1;
}

void NowSuccess(char Id)
{
 char Sstation[3],Stel[3],Dstation[3],Dtel[3];
```

```
 Ringing(Id,0);
 Stel[1]=1;
 Stel[2]=Id;
 Dtel[1]=1;
 Dtel[2]=Tel[Id].CallNum[4];
 printf("Tel[Id].CallNum[4]=%02x",Dtel[2]);
 Send(Successed,Sstation,Stel,Dstation,Dtel);
}

void IsrSuccess(char buffer[10])
{
 printf("I have Success signal from Tel-%02x to Tel-%02x\n",buffer[4],buffer[8]);
 Tel[buffer[8]].Reach=0;
 Tone(buffer[8],3,0); //stop ringback
 Make_Connect(buffer[8]);
}

void Closeconnect(Id)
{
 char Sstation[3],Stel[3],Dstation[3],Dtel[3];
// stop send receive
// close connection
 Stel[1]=1;
 Stel[2]=Id;
 Dtel[1]=1;
 Dtel[2]=Tel[Id].CallNum[4];
 Send(Closed,Sstation,Stel,Dstation,Dtel);
}

void IsrClose(char buffer[10])
{
//   stop send receive
//   close connection
   Tone(buffer[8],2,1);
}

void Init_LAN(char temp)
{
          Initialize() ;
          Initialize1() ;
          Initialize2() ;
          Initialize3() ;
          Initialize4() ;
          switch(temp) {
            case 'L':
```

```
            DoListen( &spxConnectionID ) ;
            DoListen1( &spxConnectionID1 ) ;
            DoListen2( &spxConnectionID2 ) ;
            DoListen3( &spxConnectionID3 ) ;
            DoListen4( &spxConnectionID4 ) ;
            break;
          case 'S':
            MakeConnection( &spxConnectionID ) ;
            delay(1000);
            MakeConnection1( &spxConnectionID1 ) ;
            delay(1000);
            MakeConnection2( &spxConnectionID2 ) ;
            delay(1000);
            MakeConnection3( &spxConnectionID3 ) ;
            delay(1000);
            MakeConnection4( &spxConnectionID4 ) ;
            delay(1000);
            break;
        }
}

void Initialize(void)
{
    BYTE maj, min;
    WORD max, avail;
    done = 0 ;
    SPXInitialize((BYTE *)&maj, (BYTE *)&min, (WORD *)&max, (WORD *)&avail) ;
    socket = IntSwap(SOCKET) ;
    IPXOpenSocket((BYTE *) &socket, (BYTE) SHORT_LIVED) ;
    CreateReceiveECBsandPost() ;
    CreateSendECBandHeader( ) ;
}

void Initialize1(void)
{
    BYTE maj, min;
    WORD max, avail;
    done = 0 ;
    SPXInitialize((BYTE *)&maj, (BYTE *)&min, (WORD *)&max, (WORD *)&avail) ;
    socket1 = IntSwap(SOCKET1) ;
    IPXOpenSocket((BYTE *) &socket1, (BYTE) SHORT_LIVED) ;
    CreateReceiveECBsandPost1() ;
    CreateSendECBandHeader1( ) ;
}
```

```
void Initialize2(void)
{
    BYTE maj, min;
    WORD max, avail;
    done = 0 ;
    SPXInitialize((BYTE *)&maj, (BYTE *)&min, (WORD *)&max, (WORD *)&avail) ;
    socket2 = IntSwap(SOCKET2)  ;
    IPXOpenSocket((BYTE *) &socket2, (BYTE) SHORT_LIVED) ;
    CreateReceiveECBsandPost2() ;
    CreateSendECBandHeader2( ) ;
}

void Initialize3(void)
{
    BYTE maj, min;
    WORD max, avail;
    done = 0 ;
    SPXInitialize((BYTE *)&maj, (BYTE *)&min, (WORD *)&max, (WORD *)&avail) ;
    socket3 = IntSwap(SOCKET3)  ;
    IPXOpenSocket((BYTE *) &socket3, (BYTE) SHORT_LIVED) ;
    CreateReceiveECBsandPost3() ;
    CreateSendECBandHeader3( ) ;
}

void Initialize4(void)
{
    BYTE maj, min;
    WORD max, avail;
    done = 0 ;
    SPXInitialize((BYTE *)&maj, (BYTE *)&min, (WORD *)&max, (WORD *)&avail) ;
    socket4 = IntSwap(SOCKET4)  ;
    IPXOpenSocket((BYTE *) &socket4, (BYTE) SHORT_LIVED) ;
    CreateReceiveECBsandPost4() ;
    CreateSendECBandHeader4( ) ;
}

void CreateReceiveECBsandPost( void )
{
    char i ;
    for(i = 0; i < MAX_RECEIVE_ECBs; i++) {
              CreateReceiveECBandHeader() ;
       SPXListenForSequencedPacket(receiveECB) ;
    }
}
```

```
void CreateReceiveECBsandPost1( void )
{
    char i ;
    for(i = 0; i < MAX_RECEIVE_ECBs; i++) {
            CreateReceiveECBandHeader1() ;
      SPXListenForSequencedPacket(receiveECB1) ;
    }
}

void CreateReceiveECBsandPost2( void )
{
    char i ;
    for(i = 0; i < MAX_RECEIVE_ECBs; i++) {
            CreateReceiveECBandHeader2() ;
      SPXListenForSequencedPacket(receiveECB2) ;
    }
}

void CreateReceiveECBsandPost3( void )
{
    char i ;
    for(i = 0; i < MAX_RECEIVE_ECBs; i++) {
            CreateReceiveECBandHeader3() ;
      SPXListenForSequencedPacket(receiveECB3) ;
    }
}

void CreateReceiveECBsandPost4( void )
{
    char i ;
    for(i = 0; i < MAX_RECEIVE_ECBs; i++) {
            CreateReceiveECBandHeader4() ;
      SPXListenForSequencedPacket(receiveECB4) ;
    }
}

void Usage( void )
{
    printf("SPXCHAT <s or l> \n");
    printf("      s - run as a sender.\n");
    printf("      l - run as a listener.\n");
}

void MakeConnection( WORD *spxConnectionID)
{
```

```
    CreateConnectionECBandHeader() ;
    SPXEstablishConnection((BYTE)3, (BYTE) 0xFF, spxConnectionID,
                                                                connectionECB);
    while (connectionECB->inUseFlag)
        kbhit() ;
    if (connectionECB->completionCode == 0)
        printf("Connected to foreign host, Hit ESC to kill.\n");
    else
        printf("Could not connect to foreign host [%d]\n",
                                            connectionECB->completionCode);
    return ;
}

void MakeConnection1( WORD *spxConnectionID1)
{
    CreateConnectionECBandHeader1() ;
    SPXEstablishConnection((BYTE)3, (BYTE) 0xFF, spxConnectionID1,
                                                                connectionECB1);

    while (connectionECB1->inUseFlag)
        kbhit() ;
    if (connectionECB1->completionCode == 0)
        printf("Connected to foreign host, Hit ESC to kill.\n");
    else
        printf("Could not connect to foreign host [%d]\n",
                                            connectionECB1->completionCode);
    return ;
}

void MakeConnection2( WORD *spxConnectionID2)
{
    CreateConnectionECBandHeader2() ;
    SPXEstablishConnection((BYTE)3, (BYTE) 0xFF, spxConnectionID2,
                                                                connectionECB2);

    while (connectionECB2->inUseFlag)
        kbhit() ;
    if (connectionECB2->completionCode == 0)
        printf("Connected to foreign host, Hit ESC to kill.\n");
    else
        printf("Could not connect to foreign host [%d]\n",
                                            connectionECB2->completionCode);
    return ;
}
```

```
void MakeConnection3( WORD *spxConnectionID3)
{
    CreateConnectionECBandHeader3() ;
    SPXEstablishConnection((BYTE)3, (BYTE) 0xFF, spxConnectionID3,
                                                            connectionECB3);

    while (connectionECB3->inUseFlag)
          kbhit() ;
    if (connectionECB3->completionCode == 0)
          printf("Connected to foreign host, Hit ESC to kill.\n");
    else
          printf("Could not connect to foreign host [%d]\n",
                                             connectionECB3->completionCode);
    return ;
}

void MakeConnection4( WORD *spxConnectionID4)
{
    CreateConnectionECBandHeader4() ;
    SPXEstablishConnection((BYTE)3, (BYTE) 0xFF, spxConnectionID4,
                                                            connectionECB4);

    while (connectionECB4->inUseFlag)
          kbhit() ;
    if (connectionECB4->completionCode == 0)
          printf("Connected to foreign host, Hit ESC to kill.\n");
    else
          printf("Could not connect to foreign host [%d]\n",
                                             connectionECB4->completionCode);
    return ;
}

void DoListen( WORD *spxConnectionID )
{
    time_t interim;
    WORD *destinationID ;

    system("cls") ;
    printf("Waiting for connection!\n") ;

    CreateListenECBandHeader() ;
    SPXListenForConnection((BYTE)3, (BYTE)0, listenECB);

    time(&interim);
    while (listenECB->inUseFlag) {
```

```
            IPXRelinquishControl();
        if (time((time_t *)0) > (interim+3)) {
                putchar('.');
            time(&interim);
        }
    }
    destinationID = (WORD *) listenECB->IPXWorkspace ;
    *spxConnectionID = *destinationID ;
    printf("\nConnected with send side!\n") ;
}

void DoListen1( WORD *spxConnectionID1 )
{
    time_t interim;
    WORD *destinationID ;

    system("cls") ;
    printf("Waiting for connection!\n") ;

    CreateListenECBandHeader1() ;
    SPXListenForConnection((BYTE)3, (BYTE)0, listenECB1);

    time(&interim);
    while (listenECB1->inUseFlag) {
            IPXRelinquishControl();
        if (time((time_t *)0) > (interim+3)) {
                putchar('.');
            time(&interim);
        }
    }
    destinationID = (WORD *) listenECB1->IPXWorkspace ;
    *spxConnectionID1 = *destinationID ;
    printf("\nConnected with send side!\n") ;
}

void DoListen2( WORD *spxConnectionID2 )
{
    time_t interim;
    WORD *destinationID ;

    system("cls") ;
    printf("Waiting for connection!\n") ;

    CreateListenECBandHeader2() ;
    SPXListenForConnection((BYTE)3, (BYTE)0, listenECB2);
```

```
    time(&interim);
    while (listenECB2->inUseFlag) {
            IPXRelinquishControl();
         if (time((time_t *)0) > (interim+3)) {
                 putchar('.');
            time(&interim);
         }
    }
    destinationID = (WORD *) listenECB2->IPXWorkspace ;
    *spxConnectionID2 = *destinationID ;
    printf("\nConnected with send side!\n") ;
}

void DoListen3( WORD *spxConnectionID3 )
{
    time_t interim;
    WORD *destinationID ;

    system("cls") ;
    printf("Waiting for connection!\n") ;

    CreateListenECBandHeader3() ;
    SPXListenForConnection((BYTE)3, (BYTE)0, listenECB3);

    time(&interim);
    while (listenECB3->inUseFlag) {
            IPXRelinquishControl();
         if (time((time_t *)0) > (interim+3)) {
                 putchar('.');
            time(&interim);
         }
    }
    destinationID = (WORD *) listenECB3->IPXWorkspace ;
    *spxConnectionID3 = *destinationID ;
    printf("\nConnected with send side!\n") ;
}

void DoListen4( WORD *spxConnectionID4 )
{
    time_t interim;
    WORD *destinationID ;

    system("cls") ;
    printf("Waiting for connection!\n") ;
```

```
    CreateListenECBandHeader4() ;
    SPXListenForConnection((BYTE)3, (BYTE)0, listenECB4);

    time(&interim);
    while (listenECB4->inUseFlag) {
            IPXRelinquishControl();
         if (time((time_t *)0) > (interim+3)) {
                 putchar('.');
            time(&interim);
         }
    }
    destinationID = (WORD *) listenECB4->IPXWorkspace ;
    *spxConnectionID4 = *destinationID ;
    printf("\nConnected with send side!\n") ;
}


void CreateConnectionECBandHeader(void)
{
    connectionECB = (ECB *) calloc(1, sizeof(ECB)) ;
    sendHeader = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    IPXGetDataAddress((BYTE *) sendHeader,
                      (WORD *) &connectionECB->fragmentDescriptor[0].address) ;
    connectionECB->fragmentDescriptor[0].size = sizeof(SPXHeader);
    connectionECB->fragmentCount = 1 ;
    connectionECB->ESRAddress = (void (far *) () ) 0 ;
    connectionECB->socketNumber = socket ;
         sendHeader->destination.network[0]=0x0;
         sendHeader->destination.network[1]=0x0;
         sendHeader->destination.network[2]=0x0;
         sendHeader->destination.network[3]=0x0;
         sendHeader->destination.node[0]=0x00;
         sendHeader->destination.node[1]=0x80;
         sendHeader->destination.node[2]=0x48;
         sendHeader->destination.node[3]=0x91;
         sendHeader->destination.node[4]=0xe8;
         sendHeader->destination.node[5]=0xa2;
         *(WORD *) sendHeader->destination.socket = socket ;
}

void CreateConnectionECBandHeader1(void)
{
    connectionECB1 = (ECB *) calloc(1, sizeof(ECB)) ;
```

```
    sendHeader1 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    IPXGetDataAddress((BYTE *) sendHeader1,
                      (WORD *) &connectionECB1->fragmentDescriptor[0].address) ;
    connectionECB1->fragmentDescriptor[0].size = sizeof(SPXHeader);
    connectionECB1->fragmentCount = 1 ;
    connectionECB1->ESRAddress = (void (far *) () ) 0 ;
    connectionECB1->socketNumber = socket1 ;
        sendHeader1->destination.network[0]=0x0;
        sendHeader1->destination.network[1]=0x0;
        sendHeader1->destination.network[2]=0x0;
        sendHeader1->destination.network[3]=0x0;
        sendHeader1->destination.node[0]=0x00;
        sendHeader1->destination.node[1]=0x80;
        sendHeader1->destination.node[2]=0x48;
        sendHeader1->destination.node[3]=0x91;
        sendHeader1->destination.node[4]=0xe8;
        sendHeader1->destination.node[5]=0xa2;
        *(WORD *) sendHeader1->destination.socket = socket1 ;
}

void CreateConnectionECBandHeader2(void)
{
    connectionECB2 = (ECB *) calloc(1, sizeof(ECB)) ;
    sendHeader2 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    IPXGetDataAddress((BYTE *) sendHeader2,
                      (WORD *) &connectionECB2->fragmentDescriptor[0].address) ;
    connectionECB2->fragmentDescriptor[0].size = sizeof(SPXHeader);
    connectionECB2->fragmentCount = 1 ;
    connectionECB2->ESRAddress = (void (far *) () ) 0 ;
    connectionECB2->socketNumber = socket2 ;
        sendHeader2->destination.network[0]=0x0;
        sendHeader2->destination.network[1]=0x0;
        sendHeader2->destination.network[2]=0x0;
        sendHeader2->destination.network[3]=0x0;
        sendHeader2->destination.node[0]=0x00;
        sendHeader2->destination.node[1]=0x80;
        sendHeader2->destination.node[2]=0x48;
        sendHeader2->destination.node[3]=0x91;
        sendHeader2->destination.node[4]=0xe8;
        sendHeader2->destination.node[5]=0xa2;
        *(WORD *) sendHeader2->destination.socket = socket2 ;
}
```

```
void CreateConnectionECBandHeader3(void)
{
   connectionECB3 = (ECB *) calloc(1, sizeof(ECB)) ;
   sendHeader3 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

   IPXGetDataAddress((BYTE *) sendHeader3,
                     (WORD *) &connectionECB3->fragmentDescriptor[0].address) ;
   connectionECB3->fragmentDescriptor[0].size = sizeof(SPXHeader);
   connectionECB3->fragmentCount = 1 ;
   connectionECB3->ESRAddress = (void (far *) () ) 0 ;
   connectionECB3->socketNumber = socket3 ;
       sendHeader3->destination.network[0]=0x0;
       sendHeader3->destination.network[1]=0x0;
       sendHeader3->destination.network[2]=0x0;
       sendHeader3->destination.network[3]=0x0;
       sendHeader3->destination.node[0]=0x00;
       sendHeader3->destination.node[1]=0x80;
       sendHeader3->destination.node[2]=0x48;
       sendHeader3->destination.node[3]=0x91;
       sendHeader3->destination.node[4]=0xe8;
       sendHeader3->destination.node[5]=0xa2;
       *(WORD *) sendHeader3->destination.socket = socket3 ;
}

void CreateConnectionECBandHeader4(void)
{
   connectionECB4 = (ECB *) calloc(1, sizeof(ECB)) ;
   sendHeader4 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

   IPXGetDataAddress((BYTE *) sendHeader4,
                     (WORD *) &connectionECB4->fragmentDescriptor[0].address) ;
   connectionECB4->fragmentDescriptor[0].size = sizeof(SPXHeader);
   connectionECB4->fragmentCount = 1 ;
   connectionECB4->ESRAddress = (void (far *) () ) 0 ;
   connectionECB4->socketNumber = socket4 ;
       sendHeader4->destination.network[0]=0x0;
       sendHeader4->destination.network[1]=0x0;
       sendHeader4->destination.network[2]=0x0;
       sendHeader4->destination.network[3]=0x0;
       sendHeader4->destination.node[0]=0x00;
       sendHeader4->destination.node[1]=0x80;
       sendHeader4->destination.node[2]=0x48;
       sendHeader4->destination.node[3]=0x91;
       sendHeader4->destination.node[4]=0xe8;
       sendHeader4->destination.node[5]=0xa2;
```

```
        *(WORD *) sendHeader4->destination.socket = socket4 ;
}

void CreateSendECBandHeader( void )
{
   sendECB = (ECB *) calloc(1, sizeof(ECB)) ;
   sendHeader = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

   IPXGetDataAddress((BYTE *) sendHeader,
                     (WORD *) &sendECB->fragmentDescriptor[0].address) ;
   sendECB->fragmentDescriptor[0].size = sizeof(SPXHeader);
   IPXGetDataAddress((BYTE *) &command,
                     (WORD *) &sendECB->fragmentDescriptor[1].address) ;
   sendECB->fragmentDescriptor[1].size = sizeof(command) ;
   sendECB->fragmentCount = 2 ;
   sendECB->ESRAddress = (void (far *) ()) 0 ;
   sendECB->socketNumber = socket ;
}

void CreateSendECBandHeader1( void )
{
   sendECB1 = (ECB *) calloc(1, sizeof(ECB)) ;
   sendHeader1 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

   IPXGetDataAddress((BYTE *) sendHeader1,
                     (WORD *) &sendECB1->fragmentDescriptor[0].address) ;
   sendECB1->fragmentDescriptor[0].size = sizeof(SPXHeader);
   IPXGetDataAddress((BYTE *) DataInTel[1],
                     (WORD *) &sendECB1->fragmentDescriptor[1].address) ;
   sendECB1->fragmentDescriptor[1].size = sizeof(DataInTel[1]) ;
   sendECB1->fragmentCount = 2 ;
   sendECB1->ESRAddress = (void (far *) ()) 0 ;
   sendECB1->socketNumber = socket1 ;
}

void CreateSendECBandHeader2( void )
{
   sendECB2 = (ECB *) calloc(1, sizeof(ECB)) ;
   sendHeader2 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

   IPXGetDataAddress((BYTE *) sendHeader2,
                     (WORD *) &sendECB2->fragmentDescriptor[0].address) ;
   sendECB2->fragmentDescriptor[0].size = sizeof(SPXHeader);
   IPXGetDataAddress((BYTE *) DataInTel[2],
                     (WORD *) &sendECB2->fragmentDescriptor[1].address) ;
```

```
    sendECB2->fragmentDescriptor[1].size = sizeof(DataInTel[2]) ;
    sendECB2->fragmentCount = 2 ;
    sendECB2->ESRAddress = (void (far *) ()) 0 ;
    sendECB2->socketNumber = socket2 ;
}

void CreateSendECBandHeader3( void )
{
    sendECB3 = (ECB *) calloc(1, sizeof(ECB)) ;
    sendHeader3 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    IPXGetDataAddress((BYTE *) sendHeader3,
                      (WORD *) &sendECB3->fragmentDescriptor[0].address) ;
    sendECB3->fragmentDescriptor[0].size = sizeof(SPXHeader);
    IPXGetDataAddress((BYTE *) DataInTel[3],
                      (WORD *) &sendECB3->fragmentDescriptor[1].address) ;
    sendECB3->fragmentDescriptor[1].size = sizeof(DataInTel[3]) ;
    sendECB3->fragmentCount = 2 ;
    sendECB3->ESRAddress = (void (far *) ()) 0 ;
    sendECB3->socketNumber = socket3 ;
}

void CreateSendECBandHeader4( void )
{
    sendECB4 = (ECB *) calloc(1, sizeof(ECB)) ;
    sendHeader4 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    IPXGetDataAddress((BYTE *) sendHeader4,
                      (WORD *) &sendECB4->fragmentDescriptor[0].address) ;
    sendECB4->fragmentDescriptor[0].size = sizeof(SPXHeader);
    IPXGetDataAddress((BYTE *) DataInTel[4],
                      (WORD *) &sendECB4->fragmentDescriptor[1].address) ;
    sendECB4->fragmentDescriptor[1].size = sizeof(DataInTel[4]) ;
    sendECB4->fragmentCount = 2 ;
    sendECB4->ESRAddress = (void (far *) ()) 0 ;
    sendECB4->socketNumber = socket4 ;
}


void CreateReceiveECBandHeader( void )
{
    receiveECB = (ECB *) calloc(1, sizeof(ECB)) ;
    receiveHeader = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    receiveECB->ESRAddress = (void (far *) () ) ReceiveESRHandler ;
```

```
    receiveECB->socketNumber = socket ;
    receiveECB->fragmentCount = 2 ;
    IPXGetDataAddress((BYTE *) receiveHeader,
                      (WORD *) &receiveECB->fragmentDescriptor[0].address) ;
    receiveECB->fragmentDescriptor[0].size = sizeof(SPXHeader) ;
    IPXGetDataAddress((BYTE *) &command,
                      (WORD *) &receiveECB->fragmentDescriptor[1].address) ;
    receiveECB->fragmentDescriptor[1].size = sizeof(command) ;
}

void CreateReceiveECBandHeader1( void )
{
    receiveECB1 = (ECB *) calloc(1, sizeof(ECB)) ;
    receiveHeader1 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    receiveECB1->ESRAddress = (void (far *) () ) ReceiveESRHandler1 ;
    receiveECB1->socketNumber = socket1 ;
    receiveECB1->fragmentCount = 2 ;
    IPXGetDataAddress((BYTE *) receiveHeader1,
                      (WORD *) &receiveECB1->fragmentDescriptor[0].address) ;
    receiveECB1->fragmentDescriptor[0].size = sizeof(SPXHeader) ;
    IPXGetDataAddress((BYTE *) DataOutTel[1],
                      (WORD *) &receiveECB1->fragmentDescriptor[1].address) ;
    receiveECB1->fragmentDescriptor[1].size = sizeof(DataOutTel[1]) ;
}

void CreateReceiveECBandHeader2( void )
{
    receiveECB2 = (ECB *) calloc(1, sizeof(ECB)) ;
    receiveHeader2 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    receiveECB2->ESRAddress = (void (far *) () ) ReceiveESRHandler2 ;
    receiveECB2->socketNumber = socket2 ;
    receiveECB2->fragmentCount = 2 ;
    IPXGetDataAddress((BYTE *) receiveHeader2,
                      (WORD *) &receiveECB2->fragmentDescriptor[0].address) ;
    receiveECB2->fragmentDescriptor[0].size = sizeof(SPXHeader) ;
    IPXGetDataAddress((BYTE *) DataOutTel[2],
                      (WORD *) &receiveECB2->fragmentDescriptor[1].address) ;
    receiveECB2->fragmentDescriptor[1].size = sizeof(DataOutTel[2]) ;
}

void CreateReceiveECBandHeader3( void )
{
    receiveECB3 = (ECB *) calloc(1, sizeof(ECB)) ;
```

```
    receiveHeader3 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    receiveECB3->ESRAddress = (void (far *) () ) ReceiveESRHandler3 ;
    receiveECB3->socketNumber = socket3 ;
    receiveECB3->fragmentCount = 2 ;
    IPXGetDataAddress((BYTE *) receiveHeader3,
                        (WORD *) &receiveECB3->fragmentDescriptor[0].address) ;
    receiveECB3->fragmentDescriptor[0].size = sizeof(SPXHeader) ;
    IPXGetDataAddress((BYTE *) DataOutTel[3],
                        (WORD *) &receiveECB3->fragmentDescriptor[1].address) ;
    receiveECB3->fragmentDescriptor[1].size = sizeof(DataOutTel[3]) ;
}

void CreateReceiveECBandHeader4( void )
{
    receiveECB4 = (ECB *) calloc(1, sizeof(ECB)) ;
    receiveHeader4 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    receiveECB4->ESRAddress = (void (far *) () ) ReceiveESRHandler4 ;
    receiveECB4->socketNumber = socket4 ;
    receiveECB4->fragmentCount = 2 ;
    IPXGetDataAddress((BYTE *) receiveHeader4,
                        (WORD *) &receiveECB4->fragmentDescriptor[0].address) ;
    receiveECB4->fragmentDescriptor[0].size = sizeof(SPXHeader) ;
    IPXGetDataAddress((BYTE *) DataOutTel[4],
                        (WORD *) &receiveECB4->fragmentDescriptor[1].address) ;
    receiveECB4->fragmentDescriptor[1].size = sizeof(DataOutTel[4]) ;
}


void CreateListenECBandHeader( void )
{
    listenECB = (ECB *) calloc(1, sizeof(ECB)) ;
    receiveHeader = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    listenECB->ESRAddress = (void (far *) ()) 0 ;
    listenECB->socketNumber = socket ;
    listenECB->fragmentCount = 1 ;
    IPXGetDataAddress((BYTE *) receiveHeader,
                        (WORD *) &listenECB->fragmentDescriptor[0].address) ;
    listenECB->fragmentDescriptor[0].size = sizeof(SPXHeader) ;
}

void CreateListenECBandHeader1( void )
{
```

```
    listenECB1 = (ECB *) calloc(1, sizeof(ECB)) ;
    receiveHeader1 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    listenECB1->ESRAddress = (void (far *) ()) 0 ;
    listenECB1->socketNumber = socket1 ;
    listenECB1->fragmentCount = 1 ;
    IPXGetDataAddress((BYTE *) receiveHeader1,
                      (WORD *) &listenECB1->fragmentDescriptor[0].address) ;
    listenECB1->fragmentDescriptor[0].size = sizeof(SPXHeader) ;
}

void CreateListenECBandHeader2( void )
{
    listenECB2 = (ECB *) calloc(1, sizeof(ECB)) ;
    receiveHeader2 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    listenECB2->ESRAddress = (void (far *) ()) 0 ;
    listenECB2->socketNumber = socket2 ;
    listenECB2->fragmentCount = 1 ;
    IPXGetDataAddress((BYTE *) receiveHeader2,
                      (WORD *) &listenECB2->fragmentDescriptor[0].address) ;
    listenECB2->fragmentDescriptor[0].size = sizeof(SPXHeader) ;
}

void CreateListenECBandHeader3( void )
{
    listenECB3 = (ECB *) calloc(1, sizeof(ECB)) ;
    receiveHeader3 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    listenECB3->ESRAddress = (void (far *) ()) 0 ;
    listenECB3->socketNumber = socket3 ;
    listenECB3->fragmentCount = 1 ;
    IPXGetDataAddress((BYTE *) receiveHeader3,
                      (WORD *) &listenECB3->fragmentDescriptor[0].address) ;
    listenECB3->fragmentDescriptor[0].size = sizeof(SPXHeader) ;
}

void CreateListenECBandHeader4( void )
{
    listenECB4 = (ECB *) calloc(1, sizeof(ECB)) ;
    receiveHeader4 = (SPXHeader *) calloc(1, sizeof(SPXHeader)) ;

    listenECB4->ESRAddress = (void (far *) ()) 0 ;
    listenECB4->socketNumber = socket4 ;
    listenECB4->fragmentCount = 1 ;
```

```
    IPXGetDataAddress((BYTE *) receiveHeader4,
                      (WORD *) &listenECB4->fragmentDescriptor[0].address) ;
    listenECB4->fragmentDescriptor[0].size = sizeof(SPXHeader) ;
}


void cdecl ProcessReceivedData(ECB *usedECB)
   {
   IntLANFlag=command[0];
   SPXListenForSequencedPacket(usedECB) ;
   }

void cdecl ProcessReceivedData1(ECB *usedECB)
   {
   SPXListenForSequencedPacket(usedECB) ;
   }

void cdecl ProcessReceivedData2(ECB *usedECB)
   {
   SPXListenForSequencedPacket(usedECB) ;
   }

void cdecl ProcessReceivedData3(ECB *usedECB)
   {
   SPXListenForSequencedPacket(usedECB) ;
   }

void cdecl ProcessReceivedData4(ECB *usedECB)
   {
   SPXListenForSequencedPacket(usedECB) ;
   }


void Terminate(WORD spxConnectionID)
{
   sendECB->fragmentCount = 1 ;
   SPXTerminateConnection( spxConnectionID, sendECB) ;
   while(sendECB->inUseFlag)   /* Don't close socket till done */
      IPXRelinquishControl();
   IPXCloseSocket(socket) ;
}

void Terminate1(WORD spxConnectionID1)
{
   sendECB1->fragmentCount = 1 ;
```

```
    SPXTerminateConnection( spxConnectionID1, sendECB1) ;
    while(sendECB1->inUseFlag)   /* Don't close socket till done */
       IPXRelinquishControl();
    IPXCloseSocket(socket1) ;
}

void Terminate2(WORD spxConnectionID2)
{
    sendECB2->fragmentCount = 1 ;
    SPXTerminateConnection( spxConnectionID2, sendECB2) ;
    while(sendECB2->inUseFlag)   /* Don't close socket till done */
       IPXRelinquishControl();
    IPXCloseSocket(socket2) ;
}

void Terminate3(WORD spxConnectionID3)
{
    sendECB3->fragmentCount = 1 ;
    SPXTerminateConnection( spxConnectionID3, sendECB3) ;
    while(sendECB3->inUseFlag)   /* Don't close socket till done */
       IPXRelinquishControl();
    IPXCloseSocket(socket3) ;
}

void Terminate4(WORD spxConnectionID4)
{
    sendECB4->fragmentCount = 1 ;
    SPXTerminateConnection( spxConnectionID4, sendECB4) ;
    while(sendECB4->inUseFlag)   /* Don't close socket till done */
       IPXRelinquishControl();
    IPXCloseSocket(socket4) ;
}

void ending(void)
{
        Terminate( spxConnectionID ) ;
        Terminate1( spxConnectionID1 ) ;
        Terminate2( spxConnectionID2 ) ;
        Terminate3( spxConnectionID3 ) ;
        Terminate4( spxConnectionID4 ) ;
    while(sendECB->inUseFlag)   /* Hang till clear, ESC sent? */
       IPXRelinquishControl();
    while(sendECB1->inUseFlag)   /* Hang till clear, ESC sent? */
       IPXRelinquishControl();
    while(sendECB2->inUseFlag)   /* Hang till clear, ESC sent? */
```

```
    IPXRelinquishControl();
  while(sendECB3->inUseFlag)   /* Hang till clear, ESC sent? */
    IPXRelinquishControl();
  while(sendECB4->inUseFlag)   /* Hang till clear, ESC sent? */
    IPXRelinquishControl();
}
```