การติดต่อระหว่าง PC ในระบบ LAN โดยเสียง

Voice Communication in Local Area Network by IPX Protocol

โดย

นาย ชัยวัฒน์ ชัยวันเพ็ญ 34102088

นาย อภิชาติ สิริเกียรติกุล 34109484

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขา วิศวกรรมอิเล็กทรอนิคส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชาอิเล็คทรอนิคส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

เรื่อง การติดต่อระหว่าง PC ในระบบ LAN โดยเสียง

ผู้จัดทำ

นาย ชัยวัฒน์ ชัยวันเพ็ญ 34102088

นาย อภิชาติ สิริเกียรติกุล 34109484

อาจารย์ที่ปรึกษา .....

(ดร.มนัส สังวรศิลป์)

การติดต่อระหว่าง PC ในระบบ LAN โดยเสียง

นาย ชัยวัฒน์ ชัยวันเพ็ญ
นาย อภิชาติ สิริเกียรติกุล

อาจารย์ที่ปรึกษา
ดร.มนัส สังวรศิลป์

ปีการศึกษา 2537

บทคัดย่อ

ในปัจจุบัน การติดต่อสื่อสารระหว่าง PC ในระบบ LAN เป็นสิ่งจำเป็นและ มีการใช้งานอย่างกว้างขวาง แต่เป็นการติดต่อโดยใช้ข้อความภาษาอังกฤษเป็นส่วนใหญ่ โครงงานนี้จึงมีความคิดที่จะให้การติดต่อมีความสะดวกขึ้น โดยการใช้เสียงในการติดต่อ ซึ่งจะต้องทำการศึกษา การทำงานของระบบ LAN เพื่อทำการเขียนโปรแกรมควบคุมการรับส่งข้อมูลและการทำงานอื่นๆ นอกจากนี้ยังต้องศึกษาการทำงานของ Sound card เพื่อเขียนโปรแกรมในการบันทึกเสียงลงในหน่วยความจำ เป็นข้อมูลที่สามารถใช้ในการส่ง และแปลงข้อมูลนั้นกลับไปเป็นเสียงอีกทีเมื่อถึงยังปลายทางซึ่งเป็นฝ่ายรับ

Voice Communication in Local Area Network by IPX Protocol

Chaiwat Chaiwanpen

Apichart Sirikiatkul

Advisor

Dr.Manus Sungwornsilp

Abstract

In the present, the communication between PC and LAN is very important and it uses widely. But it almost communicates with English language. So in this project will has an idea for easy communication by voice. So we must be study how LAN and Sound-card works for programming. We use sound card for sound recording to memory at primary station and playback data to sound again at secondary station.

คำนำ

ในปัจจุบันการติดต่อสื่อสารระหว่าง PC ในระบบ LAN เป็นสิ่งจำเป็นอย่างหนึ่งไม่ว่าจะเป็นการแลกเปลี่ยนข้อมูลระหว่างกัน หรือใช้ในการติดต่อระหว่าง user หนึ่งไปยังอีก user หนึ่ง เป็นต้น และมาตรฐานของ PC ในปัจจุบันมีขีดความสามารถสูงขึ้น โดยแต่ละเครื่องจะสามารถทำหน้าที่ได้มากขึ้นกว่าแต่ก่อน ความสามารถที่เพิ่มขึ้นนี้ทำให้ผู้ใช้ได้รับความสะดวกสบายมากขึ้น เช่นในการใช้งานหลายๆสื่อพร้อมกัน หรือที่เรียกว่า "Multimedia" ความสามารถในด้านนี้กำลังจะเป็นมาตรฐานโดยทั่วไปในอนาคต

ปริญญานิพนธ์นี้ทำการศึกษา และทดลองการเขียนโปรแกรมที่ใช้รับส่งข้อมูลเสียง ระหว่างคอมพิวเตอร์ที่ต่อกันอยู่ภายในระบบ LAN โดยอาศัย Sound Blaster ซึ่งเป็นอุปกรณ์ที่ใช้ในงาน Multimedia เกี่ยวกับเสียง เป็นเครื่องมือที่ใช้ในการบันทึกข้อมูลเสียงและเล่นกลับ และทำการศึกษา IPX Protocol เพื่อใช้ในการเขียนโปรแกรมรับส่งข้อมูล

เนื่องจากว่าปริญญานิพนธ์นี้ ยังไม่ได้เคยมีการศึกษามาก่อน และการหาข้อมูลทำได้ค่อนข้างลำบาก ดังนั้นหากปริญญานิพนธ์นี้มีข้อผิดพลาดประการใด ทางผู้ศึกษาขออภัยมา ณ ที่นี้ด้วย

ด้วยความเคารพอย่างสูง
นาย ชัยวัฒน์ ชัยวันเพ็ญ
นาย อภิชาติ สิริเกียรติกุล

สารบัญ

บทนำ

แนวความคิดของโครงงาน

ในปัจจุบัน PC มีความสำคัญในการทำงานต่างๆมากกว่าแต่ก่อนมาก เนื่องจากความสามารถในการทำงานของ PC ถูกพัฒนาขึ้นอยู่ตลอดเวลา งานที่มีจำนวนมากและซับซ้อนซึ่งต้องใช้คอมพิวเตอร์ขนาดใหญ่ในสมัยก่อน ถูกทำแทนโดย PC ในปัจจุบันได้อย่างรวดเร็วและถูกต้อง ความสามารถที่เพิ่มขึ้นของ PC ในหลายๆด้านนี้ รวมถึงความสามารถในการนำเสนอภาพและเสียงที่ดีขึ้นแก่ผู้ใช้ ซึ่งทำให้ผู้ใช้เครื่องมีความรู้สึกใกล้เคียงธรรมชาติมากขึ้นขณะทำงาน และยังทำให้ผู้ใช้เครื่องสามารถทำงานกับข้อมูลที่ไม่ใช่เพียงตัวอักษรเท่านั้น ความสามารถที่เพิ่มขึ้นดังกล่าวนี้กำลังจะกลายเป็นมาตรฐานการทำงานโดยทั่วไปของ PC ในอนาคต ซึ่งรู้จักกันในศัพท์ที่เรียกว่า "Multimedia" อุปกรณ์สำคัญของการทำงานในลักษณะนี้คือ video card และ sound card

แนวความคิดของโครงงานนี้ ต้องการให้มีการติดต่อกันได้ระหว่างคอมพิวเตอร์ผ่านระบบ LAN โดยใช้เสียง โครงงานนี้จึงประยุกต์ ใช้ความสามารถบางประการของ sound card และอุปกรณ์ร่วม เพื่อทำหน้าที่ในการแปลงเสียง ให้เป็นข้อมูลที่จะสามารถใช้ส่งไปได้ใน LAN ไปยังเครื่องคอมพิวเตอร์เป้าหมาย และทำหน้าที่แปลงข้อมูลที่ได้รับจากเครื่องคอมพิวเตอร์ต้นทาง กลับเป็นสัญญาณเสียง จึงต้องมีการศึกษาทฤษฎีและการทำงานของ sound card เพื่อใช้ในการเขียนโปรแกรมควบคุมการทำงานของ sound card ให้ได้ตามต้องการ

นอกจากนั้นการรับและส่งข้อมูลระหว่าง PC ในระบบ LAN จำเป็นต้องเขียนโปรแกรมเพื่อควบคุมการทำงานโดยเฉพาะ จึงต้องศึกษาถึงการทำงานของระบบ LAN และ การเรียกใช้ฟังก์ชันต่างๆเพื่อให้การทำงานเป็นไปตามต้องการ

จุดประสงค์ของโครงงาน

1.เพื่อเพิ่มความสะดวกในการติดต่อระหว่างเครื่อง PC ในระบบ LAN จากเดิมที่ติดต่อกันโดยข้อความภาษาอังกฤษ เปลี่ยนมาเป็นเสียงพูดโดยตรง

2.อาจมีการพัฒนาการทำงานต่อไป สู่ระดับที่สามารถฝากข่าวสารทางเสียงในระบบ LAN โดยไม่ต้องส่งไปทันที แต่เก็บข้อมูลไว้ใน file server (user ที่ต้องการติดต่อด้วยไม่จำเป็นต้องกำลังใช้งาน PC อยู่)ได้

บทที่ 1 ความรู้ทั่วไปเกี่ยวกับระบบเครือข่ายท้องถิ่น

ระบบเครือข่ายคอมพิวเตอร์ (computer network) ประกอบด้วยคอมพิวเตอร์หลายตัวที่สามารถติดต่อเพื่อแลกเปลี่ยนข้อมูลกันได้ การติดต่อจะผ่านทางช่องทางการสื่อสารต่างๆ สามารถแบ่งประเภทของระบบเครือข่ายได้ดังนี้

1.Wide Area Network (WAN) เป็นระบบเครือข่ายที่ติดตั้งใช้งานอยู่ในบริเวณกว้าง เช่น Internet มีโมเด็มเป็นอุปกรณ์ในการสื่อสาร

2.Local Area Network (LAN) เป็นระบบเครือข่ายที่ใช้อยู่ในบริเวณไม่กว้างมากนัก เช่น ภายในมหาวิทยาลัย การส่งข้อมูลทำได้ด้วยความเร็วสูงผิดพลาดน้อย ระบบเครือข่ายท้องถิ่นออกแบบมาเพื่อลดต้นทุนและเพิ่มประสิทธิภาพการใช้งานอุปกรณ์ต่างๆร่วมกัน

3.Metropolitan Area Network(MAN)เป็นระบบเครือข่ายที่มีขนาดอยู่ระหว่าง LAN กับ WAN ใช้ในตัวเมืองหรือจังหวัดเท่านั้น

1.1 ความสามารถของระบบเครือข่ายท้องถิ่น

LAN มีข้อดีเหนือมินิคอมพิวเตอร์หรือเมนเฟรมคือ มีการประมวลผลแบบกระจายงาน (distributed processing) ความรวดเร็วในการติดต่อสื่อสารการติดต่อระหว่างสถานีผู้ใช้งาน (interconnected stations) ใช้โปรแกรมและข้อมูลร่วมกันได้ และใช้ฮาร์ดแวร์และ ทรัพยากรที่มีอยู่ร่วมกันได้

การประมวลผลแบบกระจายงาน ในการประมวลผลแบบกระจายเมื่อสถานีของ ผู้ใช้บริการขอโปรแกรม และข้อมูลจาก file server โปรแกรมคำสั่งจะถูกสำเนาจากศูนย์บริการข้อมูลไปยังหน่วยความจำที่สถานีของผู้ใช้ เมื่อเสร็จสิ้นการใช้งานโปรแกรมและไฟล์ข้อมูลแล้ว สถานีผู้ใช้จะส่งข้อมูลกลับไปเก็บยัง file server เพื่อใช้งานต่อไป

การประมวลผลแบบกระจายงานจะช่วยให้สถานีหลายๆ สถานีใช้งานร่วมกันในระบบได้ โดยไม่ปลดความสามารถในการประมวลผลข้อมูลของแต่ละสถานี ในเครื่องคอมพิวเตอร์แบบเมนเฟรม ความสามารถในการประมวลผลจะถูกแบ่งออกไปในแต่ละสถานี ถ้ายิ่งมีสถานีมากก็จะทำให้ประสิทธิภาพการทำงานลดลง ในการกระจายการประมวลผลข้อมูลจะทำให้เพิ่มความเร็วและประสิทธิภาพของระบบได้

ความรวดเร็วในการติดต่อสื่อสาร LAN มีสื่อใช้ในการส่งข้อมูลเป็นของตนเอง ทำให้การติดต่อสื่อสารมีความเร็วสูงมากกว่า 1 ล้านบิตต่อวินาที ช่วยทำให้ข้อมูลทันต่อเหตุการณ์ ลดความซ้ำซ้อนของข้อมูล และเพิ่มความถูกต้องแม่นยำ

การติดต่อระหว่างสถานีผู้ใช้งาน แต่ละสถานีถูกเชื่อมต่อเข้าด้วยกัน ทำให้ผู้ใช้ติดต่อหรือส่งข้อมูลระหว่างกันได้ เช่น E-mail

การใช้โปรแกรมร่วมกัน ในระบบ LAN ผู้ใช้สามารถจะใช้ข้อมูลและซอฟท์แวร์ร่วมกันได้ เพื่อลดความซ้ำซ้อนของข้อมูล สื่อบันทึกข้อมูล ค้นหาตรวจสอบข้อมูลได้รวดเร็วถูกต้อง

การใช้ทรัพยากรร่วมกัน ในระบบ LAN อนุญาติให้ผู้ใช้ระบบเครือข่ายใช้อุปกรณ์ร่วมกัน ช่วยลดความซ้ำซ้อนของอุปกรณ์ทำให้ประหยัดค่าใช้จ่าย

ระบบเครือข่ายท้องถิ่นได้พัฒนาขึ้นมาหลายแบบ แต่ที่นิยมใช้กันกว้างขวาง เป็นระบบเครือข่ายท้องถิ่นแบบ ISO-OSI(Open System Interconnection)ซึ่งมีคุณสมบัติต่างๆดังนี้

1.มีความยืดหยุ่นสูง ง่ายต่อการเปลี่ยนแปลงโครงสร้าง ในการเพิ่มหรือลดอุปกรณ์ในเครือข่าย

2.เป็นโครงสร้างที่สามารถขยายสาขาพิเศษออกไปได้เมื่อต้องการ

3.ตัวกลางในการส่งข้อมูล และระบบควบคุมสามารถขยายออกไปได้

4.เป็นระบบที่ความเชื่อถือได้สูง ถ้าสายส่งข้อมูลหรือเครื่องใดเสียหายจะไม่ทำให้การรับส่งข้อมูลของเครื่องอื่นเสียหายไปด้วย

5.เป็นระบบที่กระจายการควบคุม เพื่อเพิ่มความเชื่อถือได้ของข้อมูล

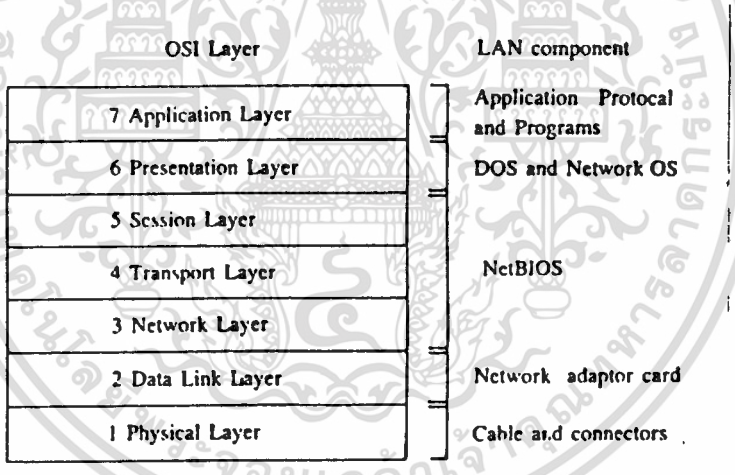
1.2 มาตรฐานของระบบเครือข่ายท้องถิ่น

ระบบเครือข่ายท้องถิ่นมีการใช้งานอย่างกว้างขวาง จึงมีการกำหนดมาตรฐานของระบบขึ้น เพื่อให้ระบบเครือข่ายของแต่ละบริษัท ที่มีความแตกต่างกันในด้าน Topology และชนิดของสายส่งข้อมูล สามารถเชื่อมต่อกันได้

สถาบันที่สำคัญในการกำหนดมาตรฐานก็เช่น IEEE(Institute of Electrical and Electronic Engineers)และ ANSI(American National Standards Institute) ทั้ง 2 สถาบันได้กำหนดมาตรฐานฮาร์ดแวร์ของระบบเครือข่ายท้องถิ่นขึ้นมา รวมทั้งคำจำกัดความอย่างเป็นทางการของระบบเครือข่าย เช่น Ethernet ของบริษัท Xerox , IBM Token-Ring ของ บริษัท IBM เป็นต้น

1.3 OSI Model

Open System Interconnection(OSI) เป็นโมเดลที่พัฒนาขึ้นโดย Inter national Standards Organization(ISO) เพื่อเป็นมาตรฐานในการออกแบบ Protocol ในการติดต่อสื่อสารระหว่างคอมพิวเตอร์ และเป็นโครงสร้างที่ใช้กันอย่างแพร่หลายสำหรับเปรียบเทียบระบบเครือข่ายทางทฤษฎี โมเดล OSI แบ่งออกเป็น 7 ชั้น ซึ่งแต่ละชั้นจะประกอบกันเป็นพื้นฐานเพื่อเป็นแนวความคิดในการออกแบบและการนำระบบเครือข่ายไปใช้งาน โมเดล OSI จะบรรยายกระบวนการติดต่อสื่อสารตามลำดับขั้นของชั้น แต่ละชั้นจะกำหนดการติดต่อกับชั้นที่ติดกัน ซึ่งการติดต่อเหล่านี้มีความยืดหยุ่นพอที่จะให้ผู้ออกแบบระบบใช้ Protocol ได้หลายแบบ โดยที่ระบบยังอยู่ในมาตรฐาน



รูปที่ 1.1 โมเดล OSI เทียบกับองค์ประกอบในระบบเครือข่ายท้องถิ่น

ชั้นที่ 1 Physical Layer

- Voltage และ Electrical pulse
- สื่อ(media) และตัวติดต่อกับสื่อ(medis interface)
- การกำหนดพิน (pin assignments)
- โทโพโลยีของระบบเครือข่าย

ในชั้นนี้เป็นการกำหนดกลไก สัญญาณไฟฟ้า การติดต่อกับระบบย่อย (subnet) และกำหนดโทโพโลยีของระบบเครือข่าย

ชั้นที่ 2 Data Link Layer

- Framing
- Error Control
- Data transperency

กำหนด Protocol ที่เครื่องแม่จะต้องใช้ในการส่งและรับข้อมูลบนระบบเครือข่าย Data Link Layer จะประกอบข้อมูลเป็น Physical-layer-service-data-units หรือ frame นอกจากนี้ชั้นนี้ยังใช้ในการแก้ไขข้อผิดพลาดทางกายภาพ(physical error) ของระบบเครือข่าย แต่ไม่มีหน้าที่แก้ไขข้อผิดพลาดของ Protocol

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

รับผิดชอบในการติดต่อสื่อสารระหว่างกระบวนการของเครื่องแม่โดยไม่มีข้อผิดพลาด อาจทำโดยผ่าน circuit service หรือ datagram service ก็ได้ แบบ circuit จะเหมือนกับการพูดโทรศัพท์ ส่วนแบบ datagram จะเหมือนการส่งจดหมาย

ชั้นที่ 5 Session Layer

- การเริ่มเชื่อมต่อ การเลิกติดต่อ
- Dialogue discipline
- การ synchronize ระหว่าง task ของผู้ใช้(end-user)

ทำงานเหมือนกับการติดต่อระหว่างผู้ใช้กับระบบเครือข่าย ให้การบริการโดยตรงที่ให้อุปกรณ์ต่างๆ อ้างอิงด้วยชื่อแทนที่จะอ้างถึงโดยตำแหน่งของมัน และมีหน้าที่แก้ไขข้อผิดพลาดที่เกิดจากการการส่งข้อมูลที่ขาดความเชื่อถือ

ชั้นที่ 6 Presentation Layer

- การกำหนดรูปแบบของข้อมูล(syntax)
- การลดขนาดของข้อมูล(Data compression)
- การเข้ารหัสข้อมูล(Data encryption)

ชั้นนี้จะกำหนดว่าโปรแกรมประยุกต์จะเข้าสู่ระบบเครือข่ายได้อย่างไร และทำงานบางอย่าง เช่น การลดขนาดข้อมูล

ชั้นที่ 7 Application

- การติดต่อสื่อสารระหว่างกระบวนการกับกระบวนการในโปรแกรม
- ตัวอย่างของ Protocol เช่น
  - * File Transfer , Access and Management(FTAM)
  - * Virtual Terminal(VT)
  - * Message Handling System(MHS)
  - * Transaction Processing(TP)
  - * Job Transfer and Manipulation(JTM)
  - * Remote Database Access(RDA)

กำหนดชนิดของโปแกรมประยุกต์ของระบบเครือข่าย โปรแกรม Application ต่างๆ จะอยู่ในชั้นนี้

1.4 โมเดล OSI กับระบบเครือข่ายท้องถิ่น

ความสัมพันธ์ขององค์ประกอบในระบบเครือข่ายท้องถิ่นโมเดล OSI มีดังนี้

ชั้นที่ 1 รวมถึงสื่อที่ใช้ในการส่งข้อมูล (สายโทรศัพท์ สายโคแอกเชียล หรือเส้นใยนำแสง) รวมถึงตัวเชื่อมต่อ ตัวขยายสัญญาณ และอุปกรณ์อื่นๆในการรับส่งบิตข้อมูล

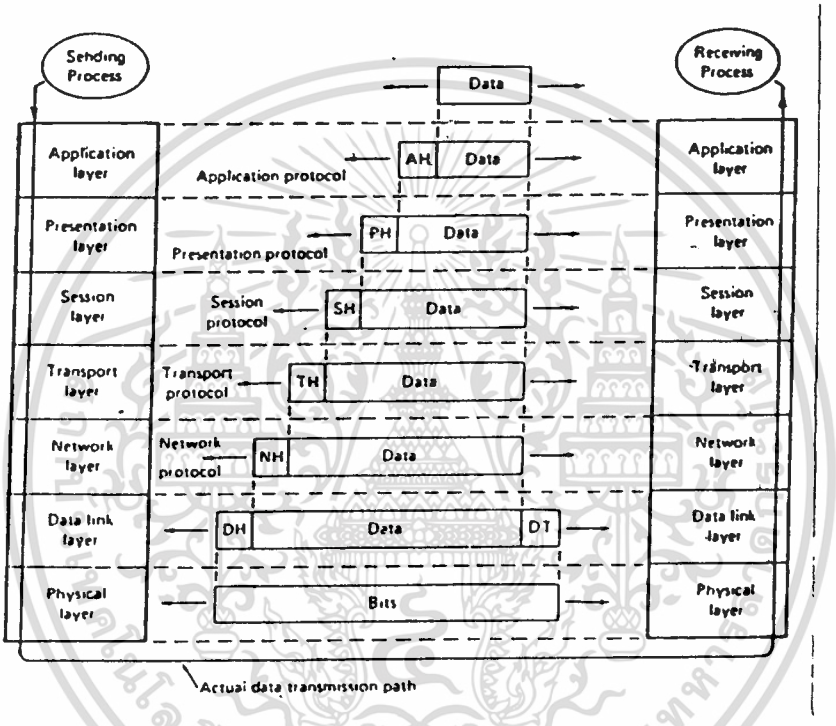
ชั้นที่ 2 นำไปใช้งานบน Network Interface Card (NIC) กำหนดข้อมูลจากชั้นที่สูงขึ้นไปให้อยู่ในลำดับที่กำหนด Ethernet,Token ring,starLAN,ARCNET และรูปแบบเฟรมอื่นๆจะแตกต่างกันออกไป ซึ่งแต่ละเฟรมจะเป็นข้อมูลพื้นฐานของระบบเครือข่ายท้องถิ่น การทำงานอื่นในชั้นนี้ก็คือ กำหนดตำแหน่งของต้นทางและปลายทางเพื่อระบุผู้ส่งและผู้รับของเฟรม รวมทั้ง check error 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 หรือ จำลองอยู่ในซอฟท์แวร์ที่สถานีผู้ใช้และศูนย์บริการข้อมูล

ชั้นที่ 6 นำไปใช้งานในระบบปฏิบัติการของระบบเครือข่าย(NOS) และการจำลองใน DOS ที่สัมพันธ์กัน

ชั้นที่ 7 โปรแกรม Protocol และโปรแกรมประยุกต์



รูปที่ 1.2 ลักษณะของข้อมูลที่ถูกส่งผ่านในแต่ละ LAYER

1.5 องค์ประกอบของ LAN

ระบบ LAN มีองค์ประกอบหลักๆ 3 ส่วนดังนี้

* อุปกรณ์ด้านฮาร์ดแวร์ (Hardware)

* โปรแกรมควบคุมระบบ หรือ ซอฟท์แวร์ (Software)

* ตัวกลางนำข้อมูล (Media)

ฮาร์ดแวร์ที่ใช้เชื่อมต่อระบบ LAN ส่วนใหญ่จะออกแบบมาเป็นการ์ด หรือแผงวงจรที่ใส่ลงในช่องสล็อต(slot) ของไมโครคอมพิวเตอร์ ซึ่งการ์ดเหล่านี้จะมีช่องอยู่ที่ด้านหลังของเครื่องไมโครคอมพิวเตอร์ เพื่อให้ต่อเข้ากับสายที่ใช้เชื่อมโยงเครือข่าย หรือที่เรียกว่าตัวกลางนำข้อมูล(Media) ซึ่งโดยทั่วไปจะใช้สายได้หลายประเภท เช่น สายโทรศัพท์ สายโคแอกเชียล สายนำสัญญาณด้วยแสง เป็นต้น ขึ้นอยู่กับ Network card ที่ใช้ แต่ทั่วๆไปมักใช้สายโคแอกเชียลเป็นส่วนใหญ่

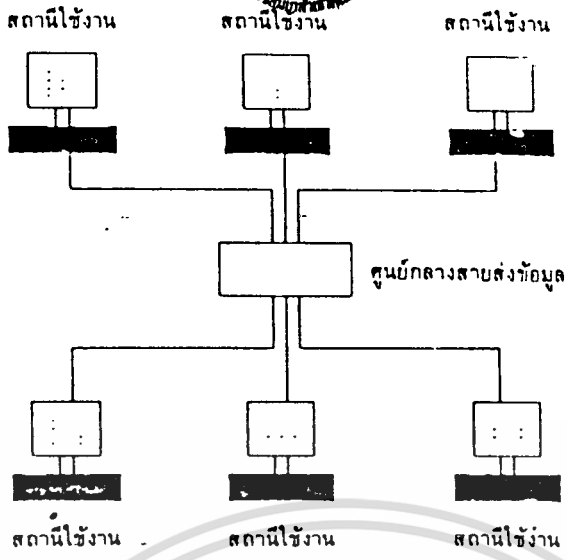
นอกจากอุปกรณ์สองส่วนที่กล่าวมาแล้ว โปรแกรมควบคุมระบบ Network หรือที่เรียกว่า Network Operating System (NOS) ก็เป็นอีกส่วนหนึ่งที่มีความสำคัญมาก เพราะนอกจากฮาร์ดแวร์แล้วการที่ระบบจะรู้ว่าข้อมูลที่ใช้งานอยู่ในเครื่องคอมพิวเตอร์ของตนเอง หรือ เครื่องอื่นในระบบนั้น โปรแกรมควบคุมระบบ Network จะเป็นผู้จัดการ ดังนั้นโปรแกรมควบคุมระบบที่ดี จะต้องทำให้ผู้ใช้ไม่เห็นความแตกต่างของการใช้งาน ไม่ว่าจะอยู่ในเครื่องคอมพิวเตอร์ของตนเอง หรือทำงานโดยใช้ข้อมูลจากเครื่องคอมพิวเตอร์อื่นใน Network ซึ่งโปรแกรมควบคุมระบบ Lan ที่ใช้ทั่วไปแบ่งการทำงานออกเป็น 2 ประเภท ประเภทแรก คือโปรแกรมควบคุมระบบ LAN ที่ทำงานภายใต้ระบบปฏิบัติการอื่นเช่น OS/2 LAN Manager อีกประเภทหนึ่งคือโปรแกรมควบคุมระบบ LAN ที่สร้างระบบปฏิบัติการของตนเองขึ้นมา เช่น NetWare เป็นต้น

1.6 ลักษณะการต่อสาย LAN (Topology)

มีอยู่ 3 แบบใหญ่ๆคือ

1.Star Topology

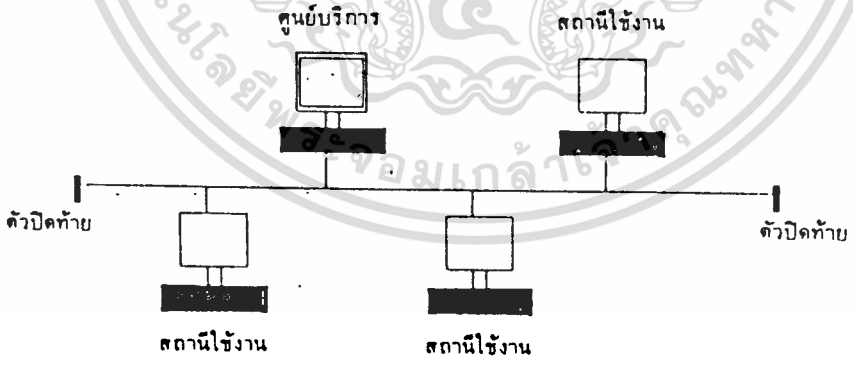
User terminal จะต่ออยู่กับศูนย์กลางสายส่งข้อมูลเพียงตัวเดียว เมื่อ terminal ใดต้องการติดต่อกับเครื่องอื่นๆ ก็ต้องผ่านตัวกลางนี้ก่อน



รูปที่ 1.3 โครงสร้างการเชื่อมต่อแบบดาว

2.Bus Toplogy

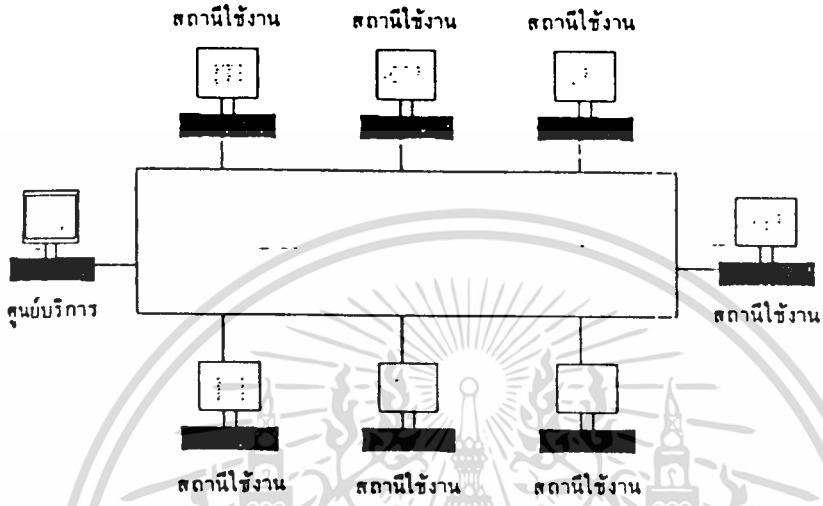
มีสายส่งข้อมูลอันเดียวเรียกว่า bus หรือ trunck ทุกสถานีต่ออยู่กับ bus และที่ปลายทั้งสองด้านปิดด้วย terminator



รูปที่ 1.4 โครงสร้างการเชื่อมต่อแบบบัส

3.Ring Toplogy

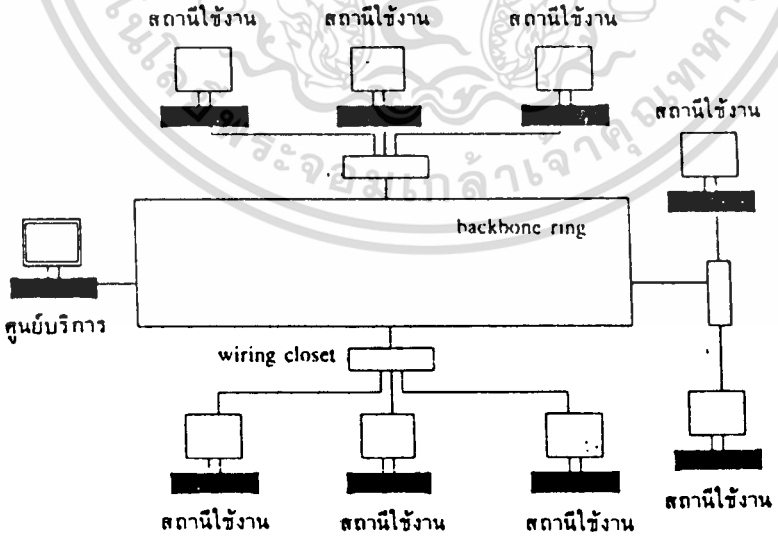
มีสถานีในระบบต่อกันเป็น Loop โดยสถานีสุดท้ายจะต่อกับสถานีแรก ทำให้รูปแบบของระบบเป็นวงแหวน ข้อมูลที่ส่งในระบบนี้จะผ่านทุกสถานี โดยอุปกรณ์หรือสถานีที่ส่งจะต้องคอยดูจากกลุ่มข้อมูลสื่อสาร(packet) ที่มันส่งเพื่อที่จะไม่ส่งข้อมูลสื่อสารนั้นซ้ำอีก



รูปที่ 1.5 โครงสร้างการเชื่อมต่อแบบวงแหวน

4.Star-ring Topology

จะต่อกันทางกายภาพแบบดาวที่ต่อออกจากวงแหวนของ Hub ดังนั้นแต่ละสถานีจะถูกมองเหมือนว่าต่อเป็นแบบวงแหวน โดยมี Hub เป็นตัวช่วยในการเชื่อมต่อกับสถานีที่ใกล้สุด



รูปที่ 1.6 โครงสร้างการเชื่อมต่อแบบวงแหวนดาว

1.7 ลักษณะการแบ่งการใช้สาย (Media Access Control)

โดยทั่วไปมี 2 แบบคือ

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

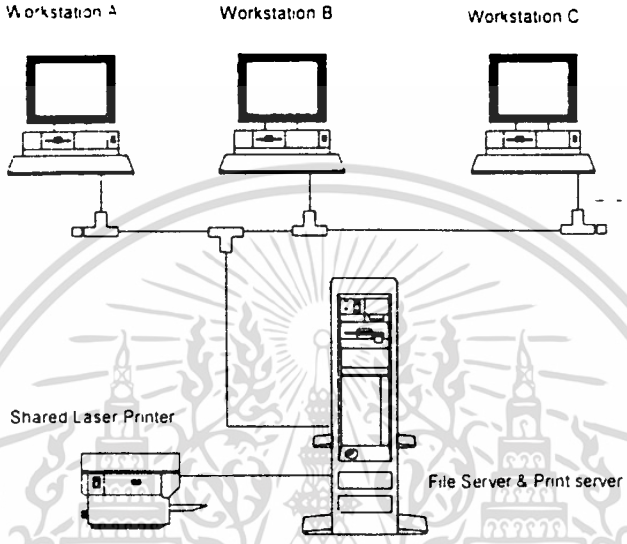
วิธีนี้ใช้ในกรณีทุกเครื่องต่ออยู่กับสายชุดเดียวกัน คือ Bus Topology โดยที่ขณะใดขณะหนึ่งคอมพิวเตอร์แต่ละเครื่องจะคอย "ฟัง"ว่าสายว่างหรือไม่(carrier sense) ถ้าว่างก็จะเริ่มส่งสัญญาณออกมา ถ้าสายว่างข้อมูลก็จะไปถึงผู้รับได้เลย แต่การเริ่มส่งสัญญาณนี้อาจเกิดขึ้นจากหลายๆสถานีพร้อมๆกันได้ ผลก็คือ สัญญาณที่ได้จะมีการชนกันในสายทำให้ข้อมูลใช้ไม่ได้ เพราะฉะนั้นแต่ละเครื่องต้องสามารถตรวจสอบการชนกัน หรือ collision detect ได้ จากนั้นแต่ละเครื่องก็จะหยุดส่งและรอ โดยนับถอยหลังในเวลาที่สุ่มขึ้นมาให้แตกต่างกันระหว่างแต่ละเครื่อง เมื่อครบเวลาที่นับของแต่ละเครื่องแล้วก็ค่อยส่งข้อมูลออกมาใหม่ ซึ่งเนื่องจากเวลาที่ใช้รอแตกต่างกัน การส่งครั้งใหม่จึงไม่มีโอกาสชนกันระหว่างคู่เดิมอีก

2.Token-passing

วิธีนี้ใช้กับ Topology ได้หลายแบบ ไม่ว่าจะเป็น bus,star หรือ ring โดยในขณะใดขณะหนึ่งจะมีคอมพิวเตอร์เพียงเครื่องเดียวใน LAN ที่มีสิทธิในการส่งข้อมูลโดยมีรหัสที่เรียกว่า token เก็บไว้ เมื่อส่งข้อมูลออกไปเสร็จแล้วก็จะส่งรหัส token นี้ออกไปให้เครื่องอื่นๆตามลำดับที่กำหนดไว้ล่วงหน้า เครื่องอื่นๆเมื่อได้รับรหัสแล้ว ถ้าเครื่องไหนยังไม่ต้องการส่งข้อมูลก็จะส่งรหัสต่อไปเลย แต่ถ้าต้องการส่งข้อมูลก็ให้ส่งข้อมูลออกมาก่อน แล้วค่อยส่งรหัสตามออกไปให้เครื่องอื่นตามลำดับ ด้วยวิธีนี้ทุกเครื่องจะได้รับสิทธิในการส่งข้อมูล 1 ครั้งภายใน 1 รอบการทำงาน ทำให้สามารถจำกัดเวลาได้ว่าจะส่งข้อมูลออกไปภายในเวลาไม่เกินกี่ ms

1.8 คอมพิวเตอร์ในเน็ตเวิร์ค

คอมพิวเตอร์ใน Network สามารถแบ่งเป็น 2 ประเภท ตามหน้าที่การทำงาน คือ Server กับ Workstation



รูปที่ 1.7 เซอร์เวอร์ กับ เวิร์คสเตชั่น

Server

เป็นคอมพิวเตอร์ที่ทำหน้าที่ให้บริการแก่เครื่องอื่นซึ่งเป็นลูกข่าย โดยทั่วไปมีหน้าที่ 3 ประเภท คือ บริการในการจัดเก็บข้อมูลเรียกว่า "File Server" บริการในการควบคุมเครื่องพิมพ์เรียกว่า "Print Server" บริการในการควบคุมอุปกรณ์การสื่อสารเรียกว่า "Communication Server" ในระบบปฏิบัติการ NetWare Server จะเป็นทั้ง File Server และ Print Server แต่ในกรณีของ Communication Server จะทำหน้าที่ง่ายๆไม่ซับซ้อน

สำหรับ Server ของ NetWare โดยทั่วไปจะทำงานในลักษณะของไมโครคอมพิวเตอร์ไม่ได้ ต้องเปิดเครื่องเพื่อรอให้บริการแก่คอมพิวเตอร์อื่นๆ เรียกว่าเป็น "Dedicated File Server" แต่บางรุ่นก็สามารถทำงานเป็นไมโครคอมพิวเตอร์ธรรมดาได้ เรียกว่าเป็น "Non-Dedicated File Server"

Workstation

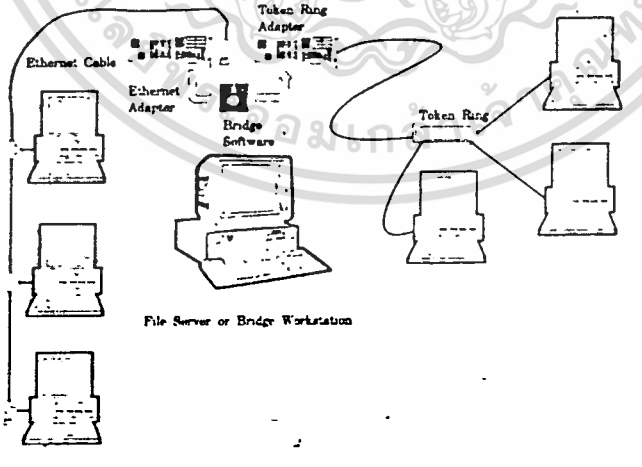
ไมโครคอมพิวเตอร์เครื่องอื่นในระบบ Network ที่ผู้ใช้สามารถเข้าไปใช้งานโปรแกรมต่างๆ ติดต่อกับ Server ได้เรียกว่า "workstation" จะเป็นไมโครคอมพิวเตอร์ที่ใช้ระบบปฏิบัติการต่างๆเช่น DOS OS/2 เป็นต้น โดยก่อนจะเริ่มติดต่อกับ Server ต้องทำการ Load โปรแกรมที่เรียกว่า "Network Shell" ก่อน Network Shell จะทำงานภายใต้ DOS ถ้าหาก workstation เป็นระบบอื่น Network Shell ก็จะต่างกันออกไป โปรแกรม Network Shell จะเกิดขึ้น ในขั้นตอนที่เรียกว่า Generate Shell ซึ่งจะต้องกำหนดประเภทของการ์ด หน่วยความจำ และ interrupt ต่างๆของโปรแกรม Shell ให้ตรงกับ Hardware

Multiple Server

ภายใต้ระบบ LAN ของ NetWare สามารถติดตั้ง Sever ได้มากกว่า 1 เครื่อง โดยทำการติดตั้งระบบปฏิบัติการ NetWare ลงในเครื่องอื่นๆ และกำหนดชื่อ Server ให้แตกต่างกัน ก็จะทำให้สามารถมี Server ไว้ใช้งานได้หลาย Server

Bridge

หมายถึง สะพานที่เชื่อมต่อระหว่างระบบ Network ต่างชุดกัน ทำได้โดยการใช้คอมพิวเตอร์เครื่องหนึ่ง ทำหน้าที่เชื่อม 2 ระบบเข้าด้วยกัน อาจเป็นเครื่องพิเศษที่ใช้สำหรับเป็น Bridgeโดยเฉพาะ หรือเป็นเครื่องธรรมดาที่ใช้โปรแกรมให้ทำงานเป็น Bridge ก็ได้



รูปที่ 1.8 Bridge ในระบบของ NetWare

Router

คืออุปกรณ์ที่ทำหน้าที่เชื่อมต่อ LAN หลายๆส่วนเข้าด้วยกัน คล้ายกับ Bridge แต่มีการทำงานที่ซับซ้อนกว่า โดยมันจะเก็บสภาวะของ Network ทั้งหมดเอาไว้ เพื่อให้ Workststion ของแต่ละส่วนใน LAN ติดต่อส่งข้อมูลหรือเรียกใช้ข้อมูลจาก Server ที่อยู่ใน LAN ส่วนอื่นได้ถูกต้อง Router อาจเป็ยเพียงต่อ LAN สองส่วนเข้าด้วยกัน หรือเชื่อมต่อ LAN หลายส่วนเข้าเป็นระบบเดียวกันก็ได้

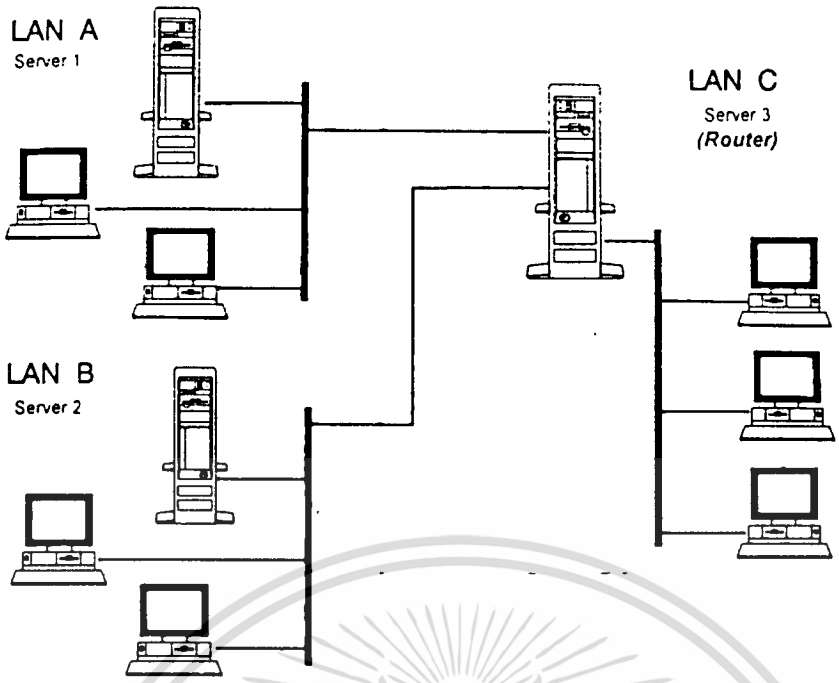
เมื่อเราต่อ LAN หลายส่วนๆเข้าเป็นระบบใหญ่ระบบเดียว Router จะเก็บสภาวะของ Network และ Server ที่กำลังทำงานอยู่เอาไว้ในตารางที่เรียกว่า Routing Table ซึ่งจะไม่เหมือนกันใน Server แต่ละตัว เมื่อ Workstation จะติดต่อไปยัง Server ตัวอื่น Routing Table จะหาเส้นทางที่เหมาะสมที่สุดที่จะติดต่อไปยัง Server นั้นให้

เมื่อถึงช่วงเวลาที่ตั้งเอาไว้ Server แต่ละตัว จะทำการส่ง Routing Table ของมันไปให้ Server ตัวอื่นๆที่ทำงานอยู่ เพื่อให้ Server ทุกตัวรู้ว่าขณะนี้ LAN ทั้งระบบมี Server ตัวไหนต่ออยู่บ้างและใครต่อกับใคร โดยส่วนที่ส่งข้อมูลนี้เรียกว่า Routing Information Protocol (RIP) ถ้าในระบบมี Server ตัวอื่นต่อเพิ่มขึ้นมาหรือมี Server บางตัวหยุดการทำงานลง หรือจุดเชื่อมต่อระหว่าง Network บางจุดเกิดปัญหา Routing Table จะถูกปรับตามสภาวะของ Network ที่เป็นจริง ทำให้การใช้งานไม่ผิดพลาด และเราไม่ต้องคอยตรวจสอบว่ามี Server อยู่กี่ตัวขณะนี้และสถานะของระบบเปลี่ยนแปลงไปจากเดิมมากน้อยแค่ไหน

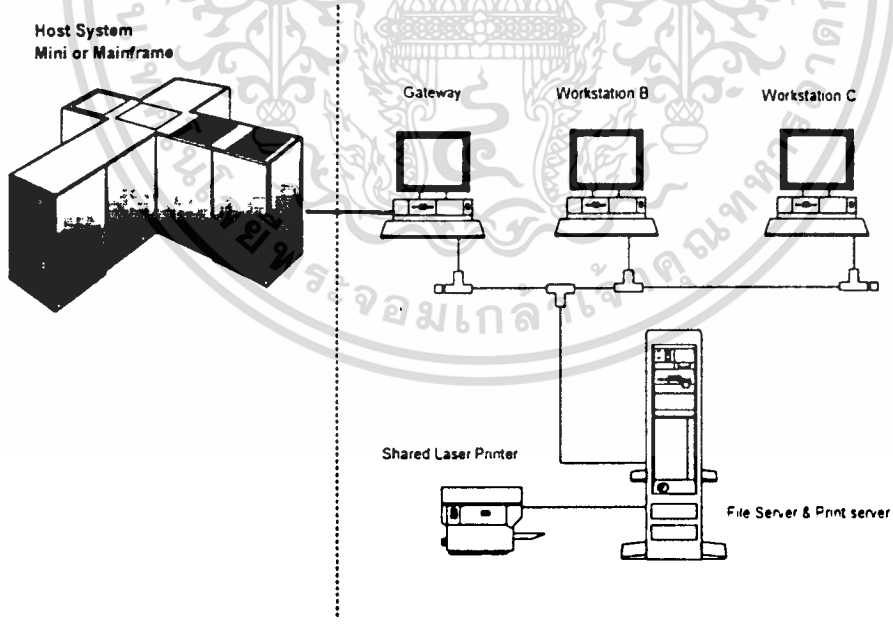
Router แบ่งได้เป็น 2 ชนิด คือ Internel Router และ External Router Internal Router จะใช้ตัว Server ทำหน้าที่นี้ ส่วนใน External Router จะใช้ Workstation ตัวหนึ่งทำหน้าที่เป็น Router อย่างเดียว

Gateway

คืออุปกรณ์ที่ใช้เชื่อมการติดต่อระหว่าง Network ที่มีสถาปัตยกรรมแตกต่างกัน หรือเชื่อมต่อกับมินิคอมพิวเตอร์ เมนเฟรมคอมพิวเตอร์ อาจทำโดยใช้เครื่องคอมพิวเตอร์เครื่องหนึ่งในการเชื่อมต่อ



รูปที่ 1.9 Router ที่ใช้เชื่อม LAN หลายๆระบบเข้าด้วยกัน



รูปที่ 1.10 Gateway

บทที่ 2 ความรู้ทั่วไปเกี่ยวกับ NetWare

NetWare เป็นโปรแกรมควบคุมระบบ Network หรือ "Network Operating System" โปรแกรมหนึ่งที่พัฒนาขึ้นโดยบริษัท Novell ระบบปฏิบัติการ NetWare จะถูกติดตั้ง ในเครื่องคอมพิวเตอร์เครื่องใดเครื่องหนึ่งในระบบ Network หลังจากที่ติดตั้งเรียบร้อยแล้ว คอมพิวเตอร์เครื่องนั้นจะถูกเรียกว่า "Server"โดยทำหน้าที่บริการแก่เครื่องคอมพิวเตอร์ลูกข่าย โดยให้บริการฮาร์ดดิสค์ และ พรินเตอร์ เครื่องอื่นๆที่มาใช้บริการจะเรียกว่า "Workstation" โดยทำงานภายใต้ระบบปฏิบัติการต่างๆ เช่น DOS OS/2 เป็นต้น และจะเข้าไปใช้อุปกรณ์ต่างๆ ของ Server ได้ ต้องโหลดโปรแกรม "LAN Shell" ซึ่งเป็นโปรแกรมฝังตัวขึ้นมาก่อน

โปรแกรม NetWare ผู้ใช้สามารถกำหนดได้ว่า จะใช้งานร่วมกับการ์ดประเภทไหน ดังนั้น NetWare จึงใช้งานได้กับการ์ดหลายแบบ และการเชื่อมต่อในแบบต่างๆกัน และสามารถใช้งานได้กับผลิตภัณฑ์ของผู้ผลิตต่างๆกัน โดยผู้ใช้จะต้องทำการระบุขั้นตอนที่ทำการติดตั้งโปรแกรม NetWare ร่วมกับฮาร์ดแวร์ต่างชนิดกัน อาจมีผลนำให้ความเร็วในการทำงานต่างกันบ้าง ขึ้นอยู่กับความเร็วในการส่งข้อมูลประเภทนั้น และไดรเวอร์ของ NetWare ที่ใช้งานร่วมกับการ์ดนั้นๆ ซึ่งความเร็วของการใช้งานจะยิ่งเห็นได้ชัดเจนเมื่อมีจำนวน Workstation เพิ่มมากขึ้นในระบบ

2.1 ประเภทต่างๆของ NetWare แบ่งเป็น 4 ระดับ คือ

1.ELS NetWare ELS ย่อมาจากคำว่า "Entry Level System" ซึ่งเป็น version ที่เล็กที่สุดของ NetWare มี 2 ระดับ คือ ใช้งาน 4 คน กับ ใช้งาน 8 คน

2.Advanced NetWare 286 เป็น NetWare มาตรฐานที่ใช้ Server ที่มี CPU 80286

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

4.NetWare 386 มีประสิทธิภาพสูงสุด โดยใช้เครื่องคอมพิวเตอร์ที่มี CPU 80386 หรือ 80486 เป็น Server

2.2 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
ขนาดของระบบปฏิบัติการใน(Harddisk)	5 MB	9 MB
ขนาดของฮาร์ดดิสก์มากที่สุด	2 GB	2048 GB
จำนวน VOLUMES 1 server	32	64
จำนวนดิสก์ไดรฟ์ 1 server	32	1024
จำนวน VOLUMES สูงสุด	255 MB	2048 MB
ขนาดไฟล์สูงสุด	255 MB	2048 MB
ใช้ไฟล์ได้พร้อมกันสูงสุด (ต่อ server)	1,000	100,000

2.3 ส่วนประกอบของระบบเครือข่าย NetWare

ระบบเครือข่าย NetWare จะประกอบด้วย ศูนย์บริการข้อมูล สถานีผู้ใช้และอุปกรณ์ อื่นๆ ที่ต่อเชื่อมกับศูนย์บริการข้อมูล

ศูนย์บริการข้อมูล เป็นหัวใจของระบบเครือข่าย NetWare ทำหน้าที่จัดการเกี่ยวกับการใช้ file ร่วมกัน ระบบรักษาความปลอดภัยของข้อมูล การทำงานติดต่อกันระหว่างสถานีการใช้ดิสค์ร่วมกัน ผู้ใช้ระบบเครือข่ายอาจตรวจสอบและควบคุมการทำงานของศูนย์บริการข้อมูล ได้จากจอภาพของศูนย์บริการข้อมูล ผู้ใช้สามารถเริ่มและหยุดศูนย์บริการข้อมูล กำหนดเวลาการใช้งาน ควบคุมการทำงานอื่นๆได้

เครื่องพิมพ์ของระบบเครือข่าย ศูนย์บริการข้อมูลแต่ละตัวจะต่อเครื่องพิมพ์ได้ไม่เกิน 5 ตัว และจะเป็นเครื่องพิมพ์ของระบบ ผู้ใช้ใดๆสามารถใช้งานได้โดยเรียกผ่านศูนย์บริการข้อมูล

ดิสค์ของระบบเครือข่าย ศูนย์บริการแต่ละตัวอาจมี Harddisk ตั้งแต่ 1 ตัว หรือมากกว่าก็ได้ เรียก Harddisk ชนิดนี้ว่า Network disks

ดิสค์ของระบบจะแบ่งเป็น volume,directory,file

- volume Server แต่ละตัวจะต้องมีที่เก็บข้อมูลอย่างน้อย 1 volume คือ SYS : VOLUME ซึ่งถูกสร้างขึ้นเมื่อทำการติดตั้ง Server

- directory แต่ละ volume จะแบ่งเป็นหน่วยทาง logic ที่เรียกว่า directory ใน directory หนึ่งๆอาจประกอบด้วย directory ย่อยอีกก็ได้ แต่ละระดับของ directory ใน volume ของระบบเครือข่าย จะประกอบเป็นโครงสร้างตามลำดับขั้น หรือ "hierarchical directory structure" ผู้ติดตั้งระบบจะเป็นผู้กำหนดจำนวนสูงสุดของ directory ที่มีในแต่ละ volume ในระหว่างการติดตั้งระบบ และหลังจากการติดตั้งระบบแล้ว ผู้ควบคุมระบบจะเป็นคนกำหนดสิทธิในการใช้ directory ให้กับผู้ใช้แต่ละคน

- file เป็นชุดข้อมูลที่ถูกเก็บไว้รวมกัน

สถานีของผู้ใช้ ผู้ใช้แต่ละคนสามารถสร้างเก็บและเรียกใช้งานไฟล์ที่อยู่ใน Server ได้จากสถานีของผู้ใช้ โดยปกติสถานีของผู้ใช้จะประกอบด้วย จอภาพ แป้นพิมพ์ ดิสค์ที่สถานีผู้ใช้ และหน่วยประมวลผล (CPU) ติดต่อกับระบบปฏิบัติการของ Server โดย Network Shell

ดิสค์ประจำสถานีผู้ใช้ ที่สถานีของผู้ใช้งานจะประกอบด้วย local disk ตั้งแต่ 1 ตัวขึ้นไป ซึ่งดิสค์เหล่านี้ไม่ได้ใช้สำหรับการชี้ไปยังไดรฟ์ของระบบเครือข่ายหรือ directory ของระบบเครือข่าย

ไดรฟ์ของระบบเครือข่าย ในระบบเครือข่ายสามารถกำหนดอักษรหมายเลขไดรฟ์ ให้ชี้ไปยัง directory ของระบบเครือข่ายได้ ซึ่งไดรฟ์ดังกล่าวจะเรียกว่า network drive ระบบเครือข่าย NetWare แต่ละสถานีสามารถกำหนดอักษรกำกับไดรฟ์ได้ 26 ไดรฟ์

2.4 ผู้ใช้ต่างๆใน NetWare แบ่งออกเป็น 5 ประเภท ใหญ่ๆคือ

1.ผู้ใช้ทั่วไป(Regular User) เป็นผู้ใช้ทั่วไปของระบบ โดยจะมีสิทธิและขอบเขตการทำงานตามที่กำหนดไว้ล่วงหน้า ไม่มีสิทธิพิเศษอื่นๆ

2.โอเปอเรเตอร์(Operator) ทำหน้าที่ควบคุมการทำงานของอุปกรณ์ต่างๆ ภายใน NetWare ประกอบด้วย

- File Server Console Operator
- Print Queue Operator
- Print Server Operator

3.ผู้จัดการ(Manager) มีหน้าที่ควบคุมดูแลผู้ใช้บางกลุ่ม แบ่งเป็น 2 ประเภท

- Account Manager ทำหน้าที่ดูแลรายละเอียด ของผู้ใช้ทั่วไปเฉพาะกลุ่มที่อยู่ในความดูแลของตน ซึ่งจะกำหนด Acoount Manager ได้ด้วยคำสั่ง syscon ในคำสั่ง User Information/Managed Users and Groups

- Workgroup Manager ทำหน้าที่เสมือน ผู้ควบคุมระบบ(Supervisor) แต่จะควบคุมเฉพาะกลุ่มผู้ใช้บางกลุ่ม ซึ่งจะสามารถเพิ่มผู้ใช้ใหม่ แก้ไขรายละเอียดของผู้ใช้ รวมทั้งยกเลิกผู้ใช้ในกลุ่มของตนได้ ผู้ที่จะเป็น Workgroup Manager จะต้องถูกกำหนดให้เป็น Account Managers มาก่อน

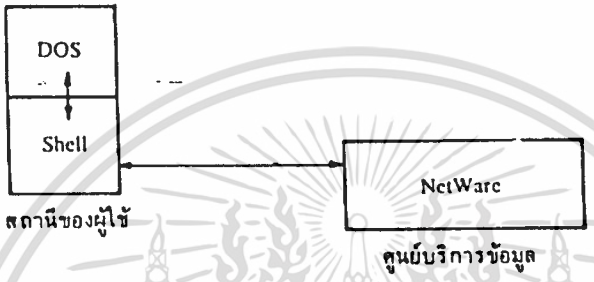
4.ผู้ควบคุมระบบ(Supervisor) มีสิทธิสูงสุดใน Server สามารทำงานได้ทุกอย่างใน Server ของตนเองโดยไม่มีขีดจำกัด หลังจากที่ทำการติดตั้งระบบเสร็จแล้ว NetWare จะสร้างชื่อผู้ใช้ชื่อ Supervisor ให้โดยอัตโนมัติซึ่งจะกำหนอหน้าที่เป็นผู้ควบคุมระบบ

5.ADMIN มีเฉพาะใน NetWare version 4.xx ซึ่งมีความสามารถของ NetWare Directory Services (NDS) โดยที่ ADMIN นี้จะเป็นผู้ควบคุมระบบ และจัดสรรทรัพยากรต่างๆภายใน Network ซึ่งประกอบด้วย Server หลาย Server ดังนั้น ADMIN จะต่างจาก Supervisor ในกรณีที่ Supervisor มีสิทธิภายใน Server เครื่องหนึ่งเท่านั้น

2.5 การทำงานของ NetWare

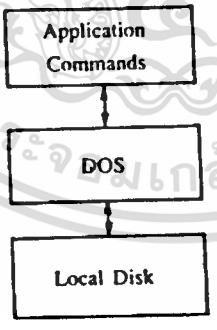
ระบบปฏิบัติการ NetWare จะสนับสนุนการใช้งานเครื่องไมโครคอมพิวเตอร์ที่ต่างกัน หรือมีระบบปฏิบัติการที่ต่างกัน โดยไม่ต้องมีการแบ่งเนื้อที่บนดิสค์ ไฟล์ทั้งหมดและสถานีผู้ใช้ทุกสถานีสามารถใช้งาน directory ร่วมกันได้

NetWare จะอยู่ที่ Server และ DOS จะอยู่ที่สถานีของผู้ใช้ NetWare Shellจะอยู่ที่สถานีของผู้ใช้ เพื่อให้ DOS ติดต่อกับ NetWare ได้



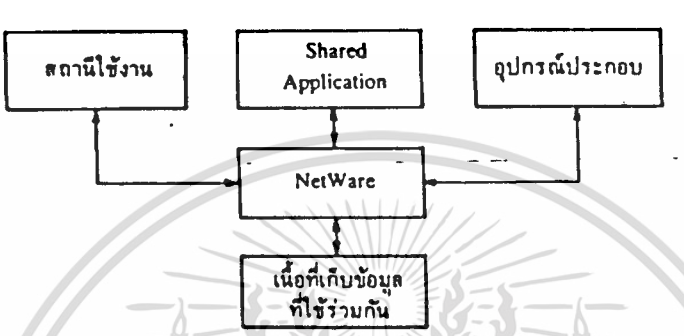
รูปที่ 2.1 Shell ของ NetWare ทำหน้าที่ติดต่อระหว่างคอสของสถานีผู้ใช้งานกับเซอร์เวอร์

DOS,NetWare และ Shell จะทำงานร่วมกันเพื่อช่วยในการใช้งานข้อมูล และฮาร์ดแวร์ร่วมกัน DOS ออกแบบให้ประมวลผลโปรแกรมที่สถานีผู้ใช้ในสภาพแวดล้อมแบบผู้ใช้คนเดียว



รูปที่ 2.2 การทำงานของคอส(สภาวะแวดล้อมแบบมีผู้ใช้คนเดียว)

NetWare ออกแบบมาให้ประมวลผลในสภาพแวดล้อมที่มีผู้ใช้หลายคน ซึ่งจะช่วยในการใช้งานอุปกรณ์ต่างๆร่วมกัน ใช้งานโปรแกรมร่วมกัน ควบคุม directory ที่ซับซ้อน และ file allocation tables(FAT) ที่ Server



รูปที่ 2.3 การทำงานของ NetWare (สภาวะแวดล้อมแบบมีผู้ใช้หลายคน)

การรักษาความปลอดภัยและการจัดการไฟล์ที่ DOS ไม่ได้จัดการให้ จะถูกจัดการโดย NetWare ที่ Server โดย NetWare กำหนดบุคคลที่มีคุณสมบัติดังนี้

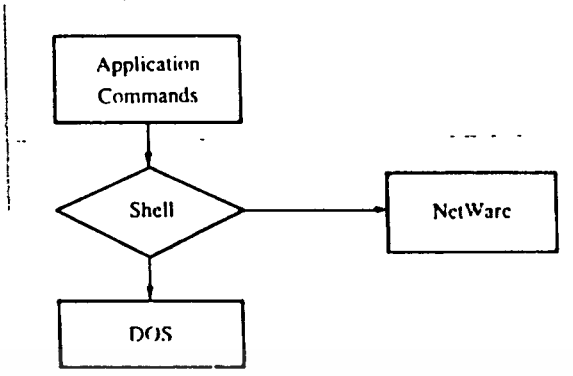
- ผู้ที่สามารถใช้งาน Server ได้
- ผู้ที่สามารถใช้งานไฟล์ได้
- ผู้ที่จะสามารถใช้งานไฟล์ร่วมกันได้

NetWare นั้นไม่ใช้ DOS ในสภาพแวดล้อม ของระบบเครือข่าย แต่ DOS จะถูกใช้ที่สถานีผู้ใช้เท่านั้น เพื่อทำงานบนไดรฟ์ที่เป็นของตัวเอง

NetWare ควบคุมการสร้าง และจัดการข้อมูลบน Server ในขณะที่ Network Shell จะช่วยให้ DOS และ NetWare ติดต่อกันได้

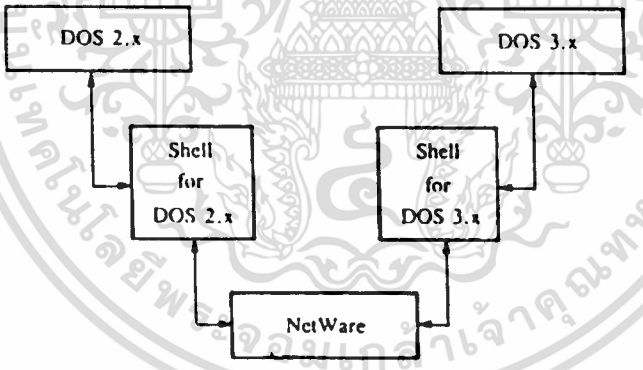
NetWare Shell ซึ่งมีขนาดประมาณ 40 Kbytes จะอยู่ที่สถานีของผู้ใช้ และดักการเรียกใช้อินเทอร์รัปต์ 21H (การขอข้อมูลหรือคำสั่งของ NetWare ที่อยู่บน Server)

DOS จะเปรียบเทียบได้กับ ผู้ที่พูดและเข้าใจ เฉพาะภาษาอังกฤษ NetWare เป็นผู้ที่พูดและเข้าใจเฉพาะภาษาไทย shell จะเป็นล่ามระหว่าง DOS และ NetWare



รูปที่ 2.4 การทำงานของ Shell ภายใต้ NetWare

Shell จะทำให้ NetWare ต่างไปจากระบบปฏิบัติการของระบบเครือข่ายอื่นๆ เพราะไม่จำกัดว่าสถานีของผู้ใช้ใช้ DOS รุ่นไหน NetWare จะสนับสนุน DOS ตั้งแต่ 2.xx ขึ้นไป



รูปที่ 2.5 การสนับสนุนการใช้คอสภายใต้ NetWare

บทที่ 3 การวิเคราะห์ NetWare เปรียบเทียบกับ OSI Model

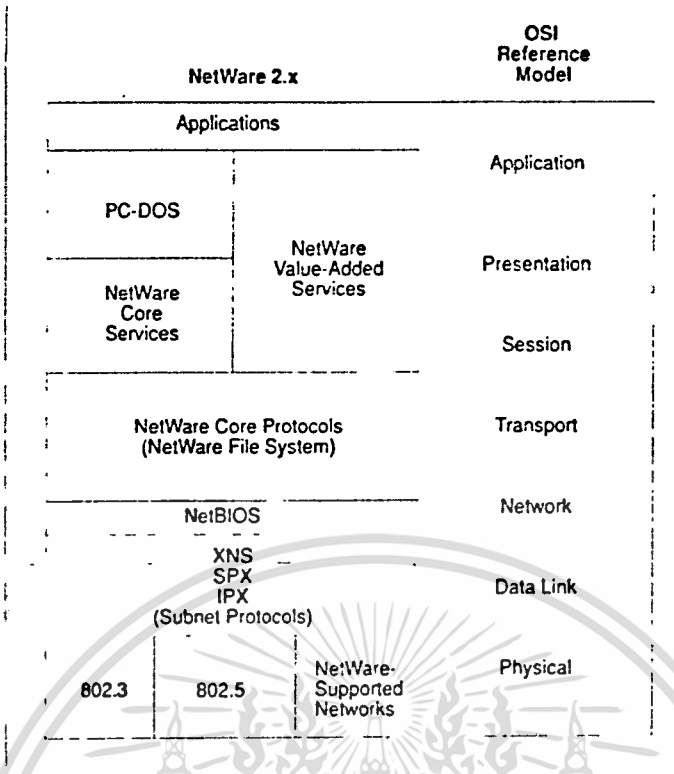
Novell's Netware คือ Network Operating System(NOS) มีพื้นฐานจาก Xerox Network System(XNS) Protocol ซึ่งพัฒนาโดย Xerox Corporation Palo Alto Research Center XNS มีลักษณะสถาปัตยกรรมที่เหมาะสมสำหรับการจัดฟังก์ชันในแต่ละ layer XNSใช้วิธีตรงๆในการ Addressing ใน Network และ Internetwork XNS จึงเหมาะที่จะนำมาประยุกต์ใช้กับ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

ข้อดีของ Netware ที่เป็นที่ยอมรับนิยมกับผู้ใช้คือ มีหลาย version ให้เลือกเช่น EIS (Entry Level System) Level 1,2 ได้ถูกออกแบบสำหรับผู้ใช้จำนวนน้อยๆ ประมาณ 4 ถึง 8 workstations Network ที่มีขนาดใหญ่ขึ้นอาจใช้ Netware ver.2.15 ซึ่ง support ถึง 100 users และสามารถ interface กับ Netware ที่ใช้กับ Macintosh และ Netware ที่ใช้กับ OS/2 Workstation Netware 386 version 3.0 ถูกออกแบบมาเพื่อใช้กับ Microprocessor 80386 และสามารถใช้ได้กับ 250 users

3.1 Netware กับ OSI Model

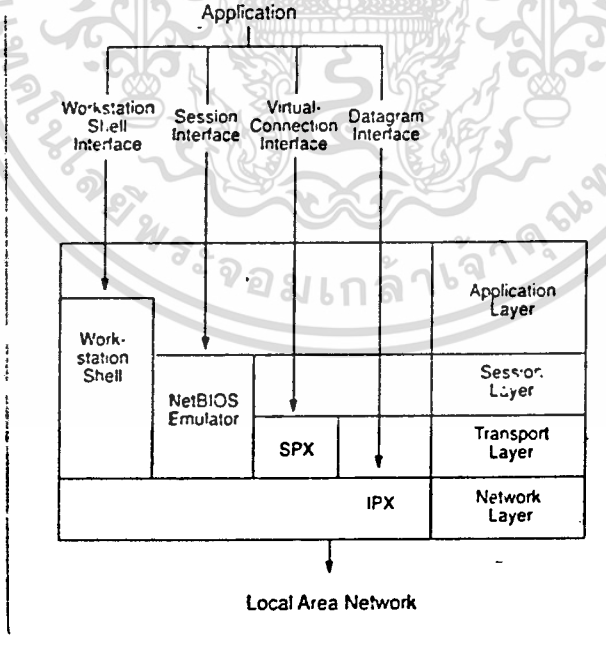
โครงสร้างของ Netware ได้ถูกออกแบบมาอิงกับมาตรฐาน OSI Model โดยสามารถเปรียบเทียบได้ดังนี้

- Physical and Data Link Layers Netwareจะอิงกับ Ethernet,IEEE 802.3,IEEE 80-2.5 เช่นเดียวกับ NOS ของบริษัทอื่นเช่น ARCNET
- Network and Transport Layers ซึ่งจะทำงานโดยใช้ Internetwork Packet Exchange (IPX) และ Sequened Packet Exchange (SPX) Protocols โดย Protocols ทั้งสองนี้เป็นส่วนหนึ่งของ XNS
- Session Layer จะใช้ Network Basic Input Output (NetBIOS) interface ซึ่งถูกพัฒนาโดย IBM และ Sytek
- Presentation and Aplication Layers จะประกอบไปด้วย Netware Core Protocols (NCP), Netware Core and Value-Added Services, DOS และ User Applications



รูปที่ 3.1 ความสัมพันธ์ระหว่างโครงสร้างของ NetWare กับ Osi Model

โปรแกรม Application ต่างๆสามารถใช้ Netware protocols ได้ 4 วิธี



รูปที่ 3.2 วิธี interface กับ NetWare

- การ Interface กับ Network Layer คือ IPX ซึ่งเป็นการติดต่อแบบ Connectionless (Datagram) ระหว่าง Workstations ซึ่ง packet ที่ถูกส่งจะไม่รับประกันการส่ง

- การ Interface กับ Transport Layer คือ SPX ซึ่งเป็นการติดต่อแบบ Connection-Oriented ระหว่าง Workstations ซึ่งเส้นทางติดต่อต้องสร้างขึ้นก่อนการส่งข้อมูล โดย SPX จะรับประกันการส่ง packets โดยปราศจาก error โดยการ Interface กับ Layer นี้

- การ Interface กับ Session Layer คือ ใช้ NetBios emulator ดังนั้น โปรแกรม Application ซึ่งใช้ Protocols NetBios สามารถใช้งานบน Netware ได้เช่นกัน

- การ Interface กับ Application Layer ทำโดยผ่าน Workstation Shell โดยทำการติดต่อผ่านฟังก์ชันของ DOS โดย Shell จะทำหน้าที่ติดต่อระหว่าง DOS กับ IPX Protocol ซึ่งใช้ในการรับส่งข้อมูลใน Network โปรแกรม Application จะใช้ Interface ใน Layer นี้ เพื่อติดต่อระหว่าง Workstations กับ Server เพื่อขอใช้ฟังก์ชันต่างๆใน Network เช่น การขอใช้ไฟล์ข้อมูลใน Server หรือการพิมพ์ผ่าน printer ของระบบ การ Interface ในLayerนี้เรียกว่า Workstation Shell Interface

3.2 Network Architecture

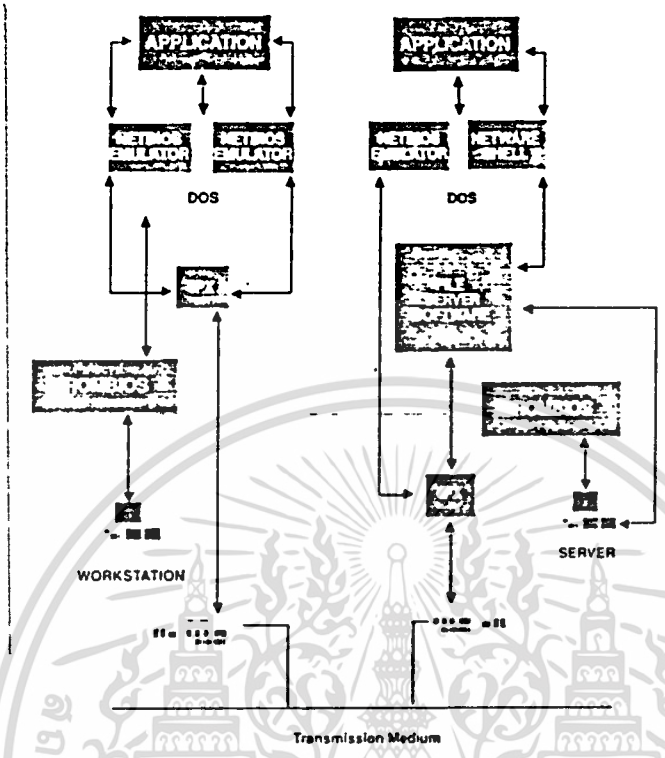
การติดต่อระหว่าง Workstation กับ Workstation หรือ Server ต่างๆ บนระบบ Netware สามารถพิจารณาได้ดังนี้

- Physical Layer ทำการติดต่อผ่าน LAN card(NIC,Network Interface Card) ที่ติดตั้งในแต่ละ Node บน Network ผ่าน Transmission medium โดยโปรแกรมที่ใช้นั้นขึ้นอยู่กับชนิดของการ์ดที่ใช้ ซึ่งมีการใช้กันอยู่หลายบริษัทเช่น NE1000,NE2000 เป็นต้น โดยโปรแกรม Driver นี้จะทำหน้าที่ Interface โดยตรง ระหว่าง IPX Protocol กับ Hardware เพื่อติดต่อระหว่างกัน

- โปรแกรม Application สามารถใช้ IPX ได้ 2 วิธี

1.ใช้ผ่าน NetBios emulator ซึ่งโดยมากใช้กับ โปรแกรมที่เขียนโดยใช้ Protocol NetBios

2.ผ่าน NetworkShell โดยใช้เรียกผ่าน DOS Interrupt โดย Local Dos calls ถูกผ่านไปที่ ROM Bios และ Local Hardware Network calls จะถูกผ่านไปให้ IPX , Network Hardware และโปรแกรมของ file server โดยโครงสร้างการติดต่อนี้ สามารถแสดงได้ดังภาพ



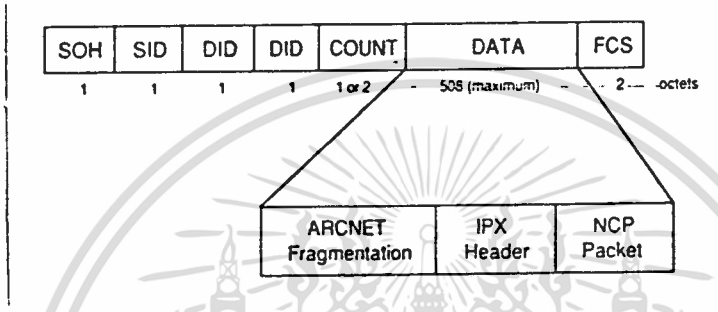
รูปที่ 3.3 ลักษณะการทำงานของ เวิร์คสเตชั่นและเซอร์เวอร์

3.3 Network Data Link Layer Protocols

เฟรมข้อมูลที่ถูกใช้ใน Network ระหว่าง Workstation กับ Server ใช้ระดับ Data Link Layer

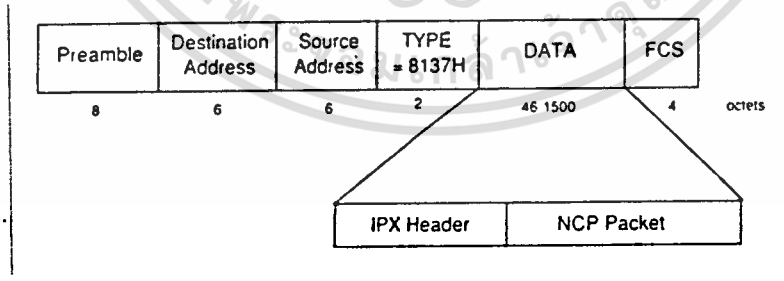
การส่งข้อมูลของ IPX จริงๆนั้น เนื่องจาก IPX เป็น Protocol ที่อยู่ในชั้น Network Layer เมื่อทำการส่งข้อมูลผ่าน LAN card ซึ่งจะทำหน้าที่ในชั้น Data Link Layer และ Physical Layer LAN card จะต้องเพิ่ม Header ในชั้น Data Link Layer เข้าไปเพิ่มขึ้นตามลักษณะที่ได้อธิบายมาแล้วในเรื่อง OSI Model โดยลักษณะของ Header ในชั้น Data Link Layer ที่ต้องเพิ่มเข้ามาโดย LAN card นี้ จะขึ้นอยู่กับว่าการ์ดอิงกับมาตรฐานใด เช่น ARCNET,IEEE802.3,IEEE802.5 เป็นต้น

-ARCNET frame จะมีรูปแบบดังรูปข้างล่าง ซึ่งจะมีส่วนของฟิลด์ข้อมูลของ Netware ประกอบอยู่ดังรูปโดย fragmentation header จะประกอบด้วยหมายเลขของข้อมูล ส่วนนี้มีความสำคัญมาก เนื่องจาก ARCNET frame จะมี Data ได้สูงสุด 508 Octet ในขณะที่ IPX Packet จะมีความยาวสูงสุด 576 Octet ดังนั้นเมื่อต้องการส่ง IPX ที่มีความยาวสูงสุดผ่าน LAN card ที่ใช้มาตรฐาน ARCNET จะใช้เฟรม ARCNET 2 เฟรมในการส่ง โดยแบ่งเป็น ARCNET เฟรมแรกจะแบ่ง IPX Protocol 508 Octet แรก และ ส่วนที่เหลืออีก 68 Octet ใส่ใน ARCNET เฟรมที่ 2



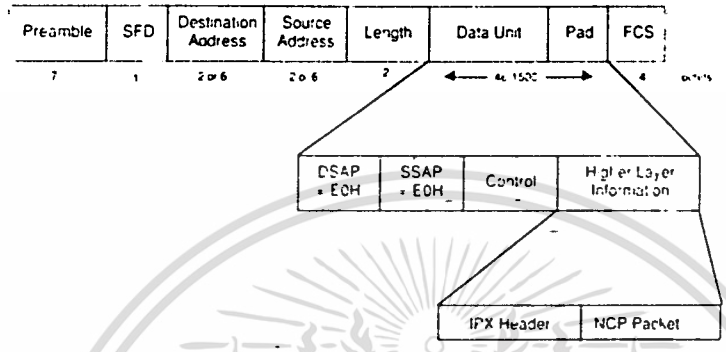
รูปที่ 3.4 Novell's NetWare core protocol packet encapsulated within an ARCNET frame

- Ethernet frame มีรูปแบบดังรูปข้างล่าง จะมีฟิลด์ Ethertype เป็น 8137H ซึ่งส่วนของข้อมูล Netware จะอยู่ในส่วนฟิลด์ Data ของเฟรม จะเห็นได้ว่า IPX Packet สามารถใส่ในเฟรม Ethernet นี้ได้โดยตรงไม่ต้องแบ่งเป็นส่วนๆ เนื่องจาก Data field สามารถมีได้ถึง 1500 Octet

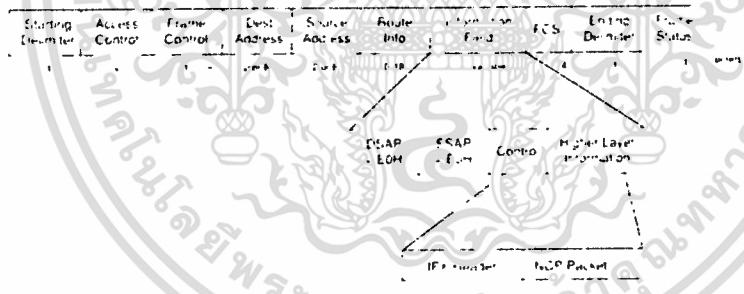


รูปที่ 3.5 Novell's NetWare core protocol packet encapsulated within an Ethernet frame

- IEEE 802.3 และ 802.5 frame มีรูปแบบดังรูปข้างล่าง โดยฟิลด์ของ Ethertype จะถูก set ให้เป็น 8137H จะเห็นได้ว่า IPX Packet สามารถใส่ใน Data field ได้ทั้งหมดโดยไม่ต้องแบ่งเป็นหลายส่วน เนื่องจาก Data field ของ IEEE frame มีความยาวสูงสุด 1500 Octet ใน 802.3 และไม่จำกัดใน 802.5



รูปที่ 3.6 Novell's NetWare core protocol packet encapsulated within an IEEE 802.3 frame



รูปที่ 3.7 Novell's NetWare core protocol packet encapsulated within an IEEE 802.5 frame

ดังกล่าวมาแล้วว่าลักษณะของเฟรม Data Link Layer จะเป็นแบบใดนั้น ขึ้นอยู่กับชนิดของ NIC(LAN card)ซึ่งแต่ละชนิดจะทำงานด้วยความเร็วที่ต่างกัน โดยที่ขนาดของเฟรมมีผลต่อความเร็ว เนื่องจากเฟรมแต่ละรูปแบบสามารถส่งข้อมูลได้ขนาดต่างๆกัน

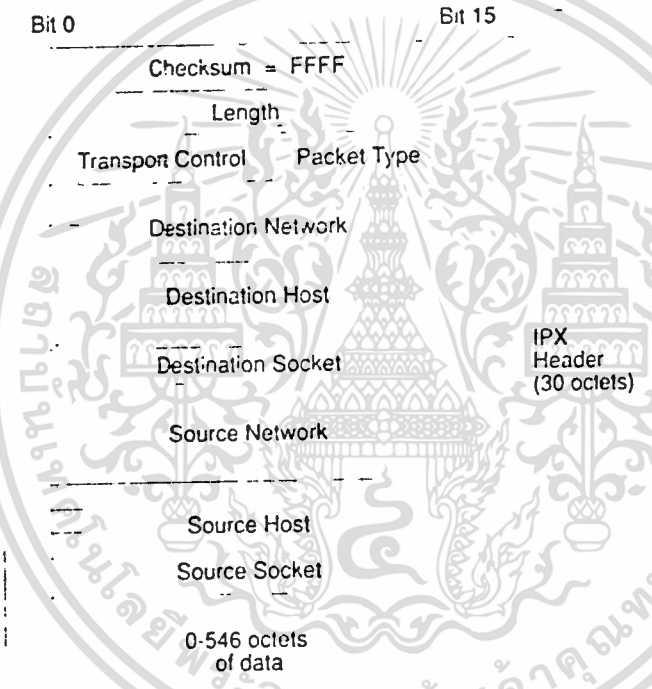
3.4 Netware Network Layer Protocol

ในชั้น Network Layer นี้ Netwareใช้ IPX Protocol ในการ Addressing routing and switching Packet จาก Workstation ด้านส่งไปยังด้านรับ เนื่องจากเป็น Protocol แบบ connectionless จึงไม่มีส่วนของ Packet ที่ใช้ในการตรวจสอบลำดับต่างๆของ Packet ดังนั้นความยาวของ IPX Packet จะมีขนาดตั้งแต่ 30 Octet ถึง 576 Octet

IPX Packet จะมีความยาวสูงสุด 576 Octet ซึ่งจะประกอบด้วย

1.Header ซึ่งจะมีความยาว 30 Octets

2.Data วึ่งจะมีความยาว 0 - 576 Octets



รูปที่ 3.8 Internetwork Packet Exchange (IPX) protocol packet

โครงสร้างของ IPX Packet สามารถแสดงได้ดังรูปข้างบน โดยแต่ละฟิลด์จะมีความหมายดังนี้

- check sum ฟิลด์นี้เป็นผลมาจาก XNS Protocol ซึ่งแต่เดิมใช้ในการ check error ในระดับ Data Link Layer(CRC) แต่ใน Netware ไม่จำเป็นต้องใช้ โดยปกติฟิลด์นี้ IPX จะ set ให้เป็น FFFF เสมอ

- Length ใช้บอกขนาดของ IPX Packet เป็นจำนวน octet ซึ่งจะมีค่าตั้งแต่ 30 - 576 octet

- Transport control ถูกใช้ในการบอกจำนวน Netware Internetwork routers โดยการเริ่มทำการส่ง IPX จะ set ฟิลด์นี้เป็น 0 ก่อน 4 bits แรกจะไม่ได้ใช้งาน ส่วน 4 bits หลังงจะให้เป็นตัวรับ จำนวน router ที่ Packet ถูกส่งผ่านจากเครื่องส่งไปเครื่องรับ ซึ่งถ้า Packet ผ่าน router ก็จะถูก set เพิ่มทีละหนึ่ง ถ้าฟิลด์นี้ถูก set จนถึง 16 Network ก็จะทำการตัด Packet นี้ทิ้งไป

- Packet type ใช้บอกชนิดของ Packet ซึ่ง Xerox ได้ แบ่งได้ดังนี้

0 Unknown Packet type

1 Routing Information Packet (RIP)

2 Echo Packet

3 Error Packet

4 Packet exchange Packet

5 Sequened Packet Protocol

16 - 31 Experimental Protocols

17 Netware Core Protocol

หมายเลข Packet type เหล่านี้ Netware จะใช้ 4 หมายเลขคือ

IPX ใช้ Packet type = 0 หรือ 4

SPX ใช้ Packet type = 5

NCP ใช้ Packet type = 17

- Destination Network จะมีความยาว 4 Octet ใช้บอกเลขหมายของ Network Address ของเครื่องรับ ถ้าถูก sert เป็น 0 แสดงว่า เครื่องรับและเครื่องส่งจะอยู่ใน Network เดียวกัน ถ้าถูก set เป็นค่าอื่น แสดงว่าต้องมีการส่ง packet นี้ผ่าน router

- Destination Host(Destination Node) ใช้ในการระบุ Node Address ของเครื่องรับ (Address ของ LAN card) มีความยาว 6 Octet สำหรับ NIC มาตรฐาน IEEE 802.3 และ 802.5 และ 2 - 3 Octet สำหรับ NIC มาตรฐาน ARCNET สำหรับการระบุให้ส่งไปทุก Node ฟิลด์นี้จะถูก set เป็น FFFFFFFFFFFFH

- Destination Socket ใช้ในการบอกหมายเลข Socket ทางเครื่องรับซึ่ง error ได้ กำหนดหมายเลข ไว้ตามลักษณะการใช้งาน Socket ดังนี้

0001 : Routing Information Packet

0002 : Echo Protocol Packet

0003 : Error Handler Packet

0020H _ 003FH : Experimental

0001H - 0BB8H : Registered with Xerox

0BB9H and Higher : Dynamically assignable

Xerox ได้กำหนดหมายเลขที่ใช้โดย Netware คือ

0451H : File Service Packet

0452H : Service Advertising Packet

0453H : Routing Information Packet

0455H : NetBIOS Packet

0456H : Diagnostic Packet

_ Source Network มีความยาว 4 Octet ซึ่งใช้ในการระบุ Network Address ซึ่งจะถูก set ด้วย IPX โดยถ้ามีค่าเป็น 0 แสดงว่าเป็น Unknown network

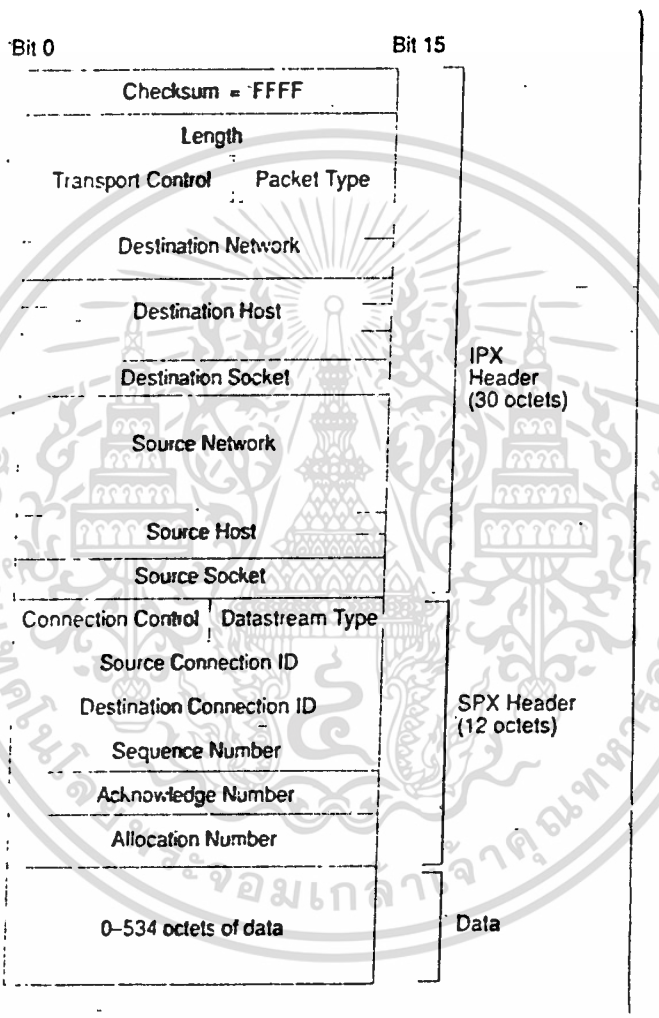
- Souce Node ใช้ในการบอก Node Address ของเครื่องส่ง

- Source Socket ใช้ในการระบุหมายเลข Socket ของเครื่องส่งที่ใช้ในการส่ง Packet

- User Data ข้อมูลจริงที่ user ต้องการส่งมีขนาด 0 - 546 Octet

3.5 Netware Transport Layer Protocol

Protocol ในชั้นนี้ SPX Header จะถูกเพิ่มเข้าใน IPX Header เพื่อทำการรับประกันการส่ง Packet และสร้างเส้นทางระหว่าง Workststions รวมทั้งการส่งข้อมูลอย่างถูกต้องตามลำดับ SPX Packet นั้นจะมีการเพิ่มข้อมูลอีก 12 Octet จาก IPX Header เนื่องจาก Packet ที่ใช้รับส่งข้อมูลมีค่าสูงสุด 576 Octets ดังนั้น Data ที่จะใช้ส่งโดย SPX Protocol จึงมีค่าตั้งแต่ 0 - 534 Octets (น้อยกว่า IPX Protocol) แสดงดังรูปข้างล่าง



รูปที่ 3.9 Sequenced Packet Exchange (SPX) protocol packet

30 Octet แรกมีโครงสร้างเหมือน IPX Header ที่กล่าวมาแล้ว ยกเว้น 2 ฟิลด์ ที่มีข้อแตกต่าง คือ Packet type จะถูก set เป็น 5 ในฐาน 10 และ Destination Host แบบที่ส่งหมดทุก Workstations (FFFFFFFFFFFFH) จะใช้ไม่ได้ใน SPX Protocol ส่วนที่เหลือในส่วนของ IPX Header อีก 12 Octet สามารถอธิบายได้ดังนี้

- Connection Control ฟิลด์นี้มีความยาว 1 Octet ใช้ในการควบคุมการไหลของข้อมูลระหว่าง 2 จุด

- Datastream type ใช้บอกชนิด Data ใน Packet มีหน้าที่เหมือนฟิลด์ Ethernet type

- Source Connection ID ใช้ในการ Demultiplexing เส้นทางที่ใช้ในการส่งข้อมูล

- Sequence Number และ Acknowledgment Number ใช้ในการระบุหมายเลข Packet ที่ใช้ในการส่ง และ Packet ที่ต้องการจะรับต่อไป เพื่อรับประกันความน่าเชื่อถือของการส่งข้อมูล

- Allocation Number ใช้ในลักษณะ end-to-end flow control ใช้บอกจำนวน buffer ที่จองไว้สำหรับเส้นทางนั้น

- Data ข้อมูลที่ต้องการส่งจริงมีความยาวตั้งแต่ 0 - 534 Octets

3.6 Network Higher Layers

Network Core Protocols(NCP) จะใช้ในการติดต่อแบบ Client/Server ระหว่าง Workstations กับ Server NCP Packet จะใส่ตามหลัง IPX Header และนำไปใส่ใน ARCNET, Ethernet,Token ring หรือ เฟรมลักษณะอื่นๆดังอธิบายแล้วในเรื่อง Network Data Link Layer Protocols

3.7 APIs ของ NetWare

APIs ย่อมาจาก Application Program Interfaces หรือรู้จักกันในชื่อ NetWare Core Protocol(NCP) มีฟังก์ชันให้เรียกใช้มากกว่า 200 ฟังก์ชันและทั้งหมดมีการเรียกใช้ INT 21H ซึ่งเป็นอินเทอรัปต์ของดอส NCP จะจัดการในเรื่องไฟล์ การพิมพ์ การติดต่อสื่อสาร การจัดการบนระบบเครือข่าย ที่เรียกใช้ได้โดยเครื่องไมโครคอมพิวเตอร์ใดๆ ที่ต่ออยู่กับระบบเครือข่าย

โปรแกรมที่เขียนกับ API ของ Novell ช่วยในการบริการบางอย่างของระบบเครือข่าย แต่จะประมวลผลได้เฉพาะบนระบบเครือข่ายที่ใช้ระบบปฏิบัติการ NetWare เท่านั้น ส่วนโปรแกรมในระบบเครือข่ายที่เขียนโดยใช้ดอส จะประมวลผลได้บนระบบเครือข่ายท้องถิ่นของเครื่องไมโครคอมพิวเตอร์ได้แทบทุกระบบ แต่ฟังก์ชันในการใช้ และควบคุมระบบเครือข่ายมีจำกัด และการทำงานในระบบเครือข่ายบางครั้งต้องการการทำงานมากกว่าที่ฟังก์ชันของดอสมีให้

สภาพแวดล้อมของระบบเครือข่ายส่วนใหญ่จะสนับสนุน API ตามลำดับขั้นตามที่แสดง โดยชั้น (layer) ของระบบเครือข่าย ซึ่งชั้นดังกล่าวจะสัมพันธ์กับชั้นทั้ง 7 ของ OSI Model ในระดับชั้นล่างจะเป็นการบริการที่ค่อนข้างธรรมดา ในขณะที่ระดับสูงจะให้การบริการที่มีประสิทธิภาพมาก ซึ่งจะอยู่ล้อมรอบโปรแกรมประยุกต์จากรูทีนในระดับต่ำ

NetWare ใช้โพรโทคอล Xerox Network Systems (XNS), Internet Datagram Protocol (IDP)และ Sequenced Packet Protocol (SPP) ในการบริการระบบเครือข่าย และการบริการใน transport layer ในสภาพแวดล้อมของ NetWare การบริการเหล่านี้จะอ้างเป็น Internetwork Packet Exchange (IPX) และ Sequenced Packet Exchange (SPX) ในเอกสารของ XNS จะกำหนดรูปแบบของกลุ่มข้อมูลสื่อสาร และการใช้งานส่วน API โดยจะใช้โครงสร้างข้อมูลที่เรียกว่า event control blocks (ECBs) ซึ่ง Novellได้นำไปใช้ โปรแกรมที่ใช้ API เหล่านี้ประมวลผลได้เฉพาะบนระบบเครือข่ายท้องถิ่นที่ใช่ระบบปฏิบัติการ NetWare เท่านั้น

ในชั้นสูงขึ้นไปคือ session layer จะสนับสนุนมาตรฐาน 2 อันซึ่งเป็นของ IBM คือ NETBIOS และ advanced program-to-program communication logical unit 6.2 (APPC LU 6.2) ซึ่งทั้งสองแบบนั้นสนับสนุนการติดต่อสื่อสารแบบ peer-to-peer โดยพื้นฐานของ NETBIOS จะอยู่บนระบบเครือข่ายท้องถิ่น ส่วน APPC จะอยู่ทั้งบน LAN และ WAN

ในชั้นที่สูงที่สุดหรือ application layer นั้นมาตรฐานทางอุตสาหกรรมของ API นั้น ดอส 3.1 และ 3.3 จะมีฟังก์ชันการเรียกใช้เพียง 10 ฟังก์ชันเท่านั้น มีการทำงานเกี่ยวกับ การล็อคช่วงของไบต์, การหาชื่อของเครื่อง(get machine name),การกำหนดข้อความ setup ของเครื่องพิมพ์,redirect device ในดอส 3.3 จะเพิ่มฟังก์ชันในการกำหนดตัวนับในการควบคุมดูแล และการมอบหมายการเรียก ซึ่งจะนำข้อมูลที่ค้างในบัฟเฟอร์บันทึกลงดิสค์

ในสภาพแวดล้อมของ NetWare รูทีน loader จะทำการติดตั้ง API รูทีนนี้จะกำหนดเวกเตอร์การอินเทอรัปต์ ให้ชี้ไปยังจุดเริ่มต้นของ code ซึ่งจะรองรับการขัดจังหวะ (interrupt) แล้วจะฝังตัวอยู่ในหน่วยความจำ(terminate and stay resident) หมายเลขของอินเทอรัปต์ที่ APIใช้ รวมทั้งพารามิเตอร์ที่ต้องส่งไปและที่ส่งกลับมาให้ จะอยู่ในคู่มืออ้างอิงของ API นั้นๆซึ่ง Novell ได้จัดทำขึ้น

หมายเลขของอินเทอรัปต์เวกเตอร์ที่จะเรียกใช้ API เป็นดังตารางข้างล่าง

API	หมายเลขอินเทอร์รัปต์
NetWare Core Services	INT 21H
NETBIOS	INT 5CH
NetWare LU 6.2	INT 68H
Low-Level API (LLAPI)	INT 7AH
High-Level Language API	INT 44H
PCOX API	INT 6FH

เนื่องจาก API เหล่านี้จะเรียกผ่านอินเทอรัปต์ ปกติแล้วการเรียกใช้จะเรียกจากภาษาแอสเซมบลี้ API แต่ละตัวที่ติดตั้งจะใช้หน่วยความจำล่างจำนวนหนึ่งเพื่อเป็นที่อยู่ของมัน ทำให้หน่วยความจำต้องสูญเสียไปจากหน่วยความจำที่มีอยู่ เช่น Shell ของ NetWare ที่สถานีผู้ใช้จะใช้หน่วยความจำประมาณ 60 Kbytes, NETBIOS ประมาณ 20 Kbytes และ APPC/PC มากกว่า 25 Kbytes

การบริการของศูนย์บริการข้อมูล และการบริการด้านการติดต่อสื่อสารของระบบเครือข่ายที่ใช้โปรแกรมประยุกต์ใช้งานอยู่ จะเรียกใช้ผ่าน NCP API ที่อยู่ใน Shell ของNetWare ส่วน API อื่นเพื่องานเฉพาะอย่างต้องซื้อแยกต่างหากและจะต้องถูกโหลดเข้าไปใน หน่วยความจำที่สถานีผู้ใช้ก่อนจะมีการเรียกใช้ เช่น message handling, การจัดการข้อมูล , valued-added processes และการติดต่อสื่อสารแบบ wide-area

Message Handling Services (MHS)

MHS API จะติดต่อกับโปรแกรมประยุกต์ เพื่อช่วยในการจัดเก็บข้อมูล และส่งต่อ (store-and-forward) ข้อความบนระบบเครือข่าย NetWare,ระบบเครือข่ายท้องถิ่นอื่น หรือ wide area network ซึ่งจะทำให้ผู้เขียนโปรแกรมไม่ต้องกังวล เกี่ยวกับโพรโทคอลของระบบเครือข่าย และเทคนิคในการส่งข้อมูลของระบบเครือข่ายร่วม นอกจากนี้ MHS ยังมีการรักษาความปลอดภัยของข้อมูล และการตรวจสอบความผิดพลาดระหว่างการส่งข้อความอีกด้วย

dedicated MHS store-and-forward server จะเป็นศูนย์กลางของ MHS API โดยศูนย์บริการนี้จะรวบรวมข้อความจากโปรแกรมประยุกต์บนสถานีผู้ใช้ ของระบบเครือข่ายของมันและส่งไปยัง MHS server ของระบบเครือข่ายอื่น

MHS API จะส่งข้อความและไฟล์แบบใดก็ได้ ในระหว่างโปรแกรมบนระบบเครือข่าย เช่น electronic-mail,ระบบบัญชี,ระบบสั่งซื้อสินค้า เป็นต้น ในการใช้ MHS โปรแกรมประยุกต์จะเติมส่วนหัวด้วยข้อความ ASCII 18 บรรทัด ซึ่งจะระบุชื่อปลายทาง,กลุ่ม,ชนิดของข้อความ(message content type),originating application type,ID number ,attachment file name และข้อมูลในการควบคุมอื่นๆ ส่วนหัวของข้อความนี้จะเป็นสลิป(slip) ในการส่งที่จะใช้โดย MHS server ในระหว่างกระบวนการส่ง

การจัดการข้อมูล

ฟังก์ชันการทำงานการจัดการข้อมูลมีอยู่ใน Btrieve API โดย Btrieve เป็นตัวจัดการเรคอร์ดและไฟล์ที่ได้รับความนิยมอย่างมาก ผลิตโดยบริษัท SoftCraft Inc.

วิธีการเข้าถึงข้อมูลแบบ indexed sequential ของ Btrieve จะช่วยให้เข้าถึงเรคอร์ดในฐานข้อมูลบน server ได้ด้วยความรวดเร็ว ในขณะที่ VAP ประมวลผลอยู่บน server Btrieve จะช่วยในการจัดไฟล์และเรคอร์ดสำหรับโปรแกรมประยุกต์ นอกจากนี้ยังทำงานในเรื่องการติดตามการทำงาน และการล็อคเรคอร์ดด้วยการใช้ XQL และ Xtrieve ทำให้ Btrieve มีความสามารถในการใช้งานฐานข้อมูลแบบ relational ได้

การตรวจสอบข้อผิดพลาดในระบบเครือข่าย

NetWare 2.1 มี API ที่ใช้ตรวจสอบการทำงานของระบบเครือข่าย ซึ่งใช้โพรโทคอล IPX และ SPX ในการวิเคราะห์โทโพโลยีและประสิทธิภาพของระบบเครือข่าย ซึ่งจะช่วยให้ทราบว่ามีโหนดไหนบ้างที่ต่ออยู่กับระบบเครือข่าย และให้ข้อมูลเกี่ยวกับซอฟต์แวร์ ที่อยู่บนแต่ละโหนดด้วย

การติดต่อสื่อสารแบบ peer-to-peer

Novell มี API อยู่หลายตัวที่ทำงานเกี่ยวกับการติดต่อสื่อสารแบบ peer-to-peer บนระบบเครือข่าย เช่น IPX,SPX และตัวจำลอง NETBIOS IPX จะสนับสนุนการทำงานในระดับ network layer ของ OSI Model ที่ช่วยในการส่งข้อมูลด้วยความรวดเร็ว แต่จะไม่รับประกันการส่ง และจะส่งข้อมูลสื่อสารแบบไม่เรียงลำดับ ส่วน SPX จะเป็นโพรโทคอลในระดับ transport layer ซึ่งจะรับรองการส่งและจะส่งกลุ่มข้อมูลสื่อสารตามลำดับในระหว่างสถานีที่ติดต่อสื่อสารกันอยู่ IPX และ SPX เป็นโพรโทคอลในระดับต่ำ ที่จะช่วยโปรแกรมประยุกต์ในการส่งข้อมูลระหว่างสถานีผู้ใช้ที่ต้องการความรวดเร็วและประสิทธิภาพในการทำงาน

ตัวจำลอง NETBIOS ของ Novell จะใช้ IPX สำหรับการบริการการส่งกลุ่มข้อมูลสื่อสาร NETBIOS จะเพิ่มความสามารถของ IPX โดยการรับรองการส่ง และส่งกลุ่มข้อมูลสื่อสารอย่างเป็นลำดับ ซึ่งจะมีการจัดการ การควบคุม session ในระหว่างกระบวนการที่มีการกำหนดชื่อไว้แล้วบนระบบเครือข่าย NETBIOS จะช่วยสถานีผู้ใช้ในการส่ง และรับข้อมูลได้มากถึง 131,070 bytes ต่อการเรียกใช้ 1 ครั้ง ต่างกับ IPX และ SPX ที่สามารถส่งข้อมูลได้สูงสุดในแต่ละครั้งเพียง 546 และ 534 bytes ตามลำดับ

Value-Added Processes (VAP)

VAPa จะช่วยในการทำงานด้านการบริการพิเศษ ที่ไม่มีอยู่ในระบบปฏิบัติการของระบบเครือข่าย เช่น การพิมพ์ การคอมไพล์ การสำรองข้อมูล และการบริการด้านการจัดการฐานข้อมูล

VAP API ติดต่อโดยตรงกับระบบปฏิบัติการของระบบเครือข่าย โดย VAPs จะจองเนื้อที่ใน server หรือ Bridge ซึ่งรับข้อมูลเข้าจากคอนโซล หรือแสดงข้อมูลบนคอนโซล สามารถแตกเป็นกระบวนการย่อย และ ติดต่อสื่อสารกับโหนดอื่นๆในระบบเครือข่าย โดยใช้โพรโทคอลแบบ peer-to-peer ได้ นอกจากนี้เนื่องจากศูนย์บริการจะบอกว่า VAP เป็นเหมือนผู้ใช้ที่ LOGIN เข้ามาคนหนึ่ง ดังนั้น VAP สามารถใช้ API ในระดับสูงตามที่ได้พูดถึงในตอนต้นได้ ซึ่งการใช้ VAP จะช่วยขยายความสามารถบางอย่าง ที่ระบบปฏิบัติการของระบบเครือข่ายท้องถิ่นไม่มีอยู่

Wide-area communications

API สำหรับ wide-area communication จะถูกจัดการโดยการเชื่อมต่อเป็นสะพานสื่อสาร หรือประตูสื่อสาร ระบบเครือข่ายท้องถิ่นของ NetWare สามารถจะเชื่อมต่อเป็น wide area network ได้หลายวิธี รวมทั้ง System Network Architecture (SNA) gateway กับเครื่องเมนเฟรมของ IBM และมินิคอมพิวเตอร์,transport control protocol /internal protocol (TCP/IP) gateway,x.25 และสะพานสื่อสารแบบ asynchronous

API ในด้านการติดต่อสื่อสารของ NetWare จะช่วยผู้พัฒนาโปรแกรมในบริการที่ configure และจัดการเชื่อมต่อ wide-area ในการส่งไฟล์ , quera database และการเรียกใช้ remote producer ซึ่ง Novell ได้แนะนำ API ในการติดต่อสื่อสารแบบ wide area โดยเฉพาะอย่างยิ่ง TCP/IP API และ Asynchronous Communication Server

Comprehension Core Services

Core service จะช่วนทำงานเกี่ยวกับไฟล์ การพิมพ์ และการบริการด้านการจัดการอื่นๆ การบริการที่มีอยู่ใน NCP API บน server ได้แก่

- การบริการด้านบัญชีผู้ใช้ (accounting Service)
- Appletalk Filling Protocol(AFP) services
- การบริการด้านฐานข้อมูลเครือข่าย (bindery services)
- Communication services
- การบริการการเชื่อมต่อ (connection services)
- Diagnostic services
- การจัดการไดเรคทอรี(directory services)
- การจัดสภาพแวดล้อมของศูนย์บริการข้อมูล (file-server environment services)
- การจัดการไฟล์ (file services)
- Message services
- การบริการด้านการพิมพ์ (print services)
- การจัดแถวรองานพิมพ์ (queue services)
- Service Advertising Protocol(SAP) services

- Synchronization services
- ระบบติดตาม transaction (transcation tracking services)
- Value_Added Process(VAP) services
- การจัดการสภาพแวดล้อมของสถานีผู้ใช้ (workstation services)

บทที่ 4 ทฤษฎีการรับส่งข้อมูล

การรับส่งข้อมูลบนระบบ LAN ของ NETWARE นั้น ข้อมูลจะถูกส่งในลักษณะ Packet โดยใช้ Protocal IPX/SPX ซึ่งถูกพัฒนามาจาก XNS (Xerox Network Standard Protocal) Protocal ทั้งสองนี้จะถูกใช้ในการสื่อสารแบบ Peer to Peer (Workstation แต่ละเครื่องสามารถรับส่งข้อมูลกันได้โดยตรงไม่จำเป็นต้องผ่าน fileserver) ซึ่งการติดต่อแบบนี้จะช่วยลดความยุ่งยากในการรับส่งข้อมูล และทำให้สามารถรับส่งข้อมูลได้เร็วขึ้น อีกทั้งยังช่วยลดภาระแก่ file server ทำให้ file server นำเวลาที่จะต้องใช้ในการทำงานนี้ ไปใช้ทำงานอื่นๆ แทนได้

โดยทั่วไปการรับส่งข้อมูลระหว่าง PC to PC มี 2 แบบคือ

1.**Datagram**(connectionless) การส่งในลักษณะนี้ ข้อมูล Packet จะถูกส่งไปบน Network โดยอิสระไปยังจุดหมาย โดยไม่ขอการตอบรับจากเครื่องรับปลายทาง และไม่รับรองถึงลำดับของ Packet ที่ถูกส่งไปถึงด้วย การส่งในลักษณะนี้มีข้อได้เปรียบคือ ความเร็วในการรับส่งข้อมูลจะสูง

2.**Session**(oriented connection) การติดต่อกันแบบนี้ ก่อนทำการรับส่งข้อมูลจะมีการสร้าง Logical route ก่อน แล้วจึงทำการส่งข้อมูล Packet โดยการติดต่อแบบนี้รับประกันการส่งข้อมูลถึงเครื่องปลายทาง (มีการรอการตอบรับจากเครื่องปลายทาง) และยังรับประกันว่าข้อมูลจะถูกส่งไปตามลำดับ

IPX Protocal

เป็น Protocal ที่จัดอยู่ในระดับชั้นที่ 3 (Network Layer) ของมาตรฐาน OSI โดยจะเป็นการติดต่อกันแบบ Datagram ซึ่งจะเร็วกว่าเมื่อเทียบกับ SPX Protocal เนื่องจากไม่จำเป็นต้องสร้างเส้นทางก่อนการรับส่งข้อมูล ใน NETWARE มีการตรวจสอบแล้วว่าข้อมูลที่รับส่งมีความถูกต้องถึง 95%

SPX Protocal

เป็นProtocalในระดับชั้นที่สูงขึ้นมา (Transport Layer) ซึ่งอยู่ในชั้นที่ 4 ของมาตรฐาน OSI โดยเป็นการติดต่อกันแบบ Session ซึ่งจะมีการรับประกันการส่งข้อมูลอย่างเป็นลำดับ อีกทั้งยังมีการตรวจสอบ error และแก้ไข error อีกด้วย

4.1 ลักษณะของ address และการเชื่อมต่อของ PC บน Network

Workstation แต่ละเครื่อง จะมีรหัสเฉพาะตัวซึ่งประกอบไปด้วย Network Address 4 bytes และ Node Address ซึ่งเรียกรวมกันว่า "Physical Address" ของแต่ละเครื่อง

Network Address จะบ่งชี้หมายเลขของ Network ของWorkstation แต่ละเครื่อง

Node Address แสดงถึงหมายเลขประจำ Workstation แต่ละตัว

ในกรณีที่มีการเชื่อมต่อระหว่างกลุ่ม user 2 กลุ่ม (หลาย server) จะมีอุปกรณ์ที่ใช้ในการเชื่อมต่อที่ทำหน้าที่เป็น router หรือ bridge (ในการใช้งานจริงอุปกรณ์นี้อาจเป็น Workstation ก็ได้) ซึ่งอุปกรณ์ที่ใช้เชื่อมต่อนี้จะมี Address 2 ชุด โดยแต่ละชุดก็จะถูกรู้จักจากแต่ละเซกเมนต์ที่ต่ออยู่(ดังแสดงในรูป) Address แต่ละชุดนี้จะประกอบไปด้วย Network Address 4 bytes (หมายถึงหมายเลข Network แต่ละเซกเมนต์ เช่นเดียวกัน Workstation ทั่วไป) และ Address ประจำอุปกรณ์อีก 6 bytes ซึ่งเรียกว่า "Intermediate Address"

4.2 การส่งข้อมูลโดยใช้ Protocal IPX

การรับส่งข้อมูลด้วย IPX นี้ ข้อมูลจะถูกส่งระหว่าง Network กับ Workstation โดยผ่าน socket เพราะฉะนั้นก่อนเริ่มต้นทำการติดต่อจึงต้องสร้าง socket ขึ้นมาก่อน จากนั้นจึงระบุ Destination Address ซึ่งประกอบด้วย Network Address และ Node Address ของเครื่องรับปลายทาง และในกรณีที่เป็นการส่งข้อมูลในลักษณะที่ เครื่องรับและเครื่องส่งอยู่คนละเซกเมนต์ (คนละ Network) จะต้องมีการระบุถึง Immediate Address ของอุปกรณ์ที่ทำหน้าที่เป็น Bridge ด้วย โดยข้อมูลต่างๆเหล่านี้สามารถ set ในโครงสร้างของ IPX header และ ECB(Event Control Block) เมื่อ set ข้อมูลเหล่านี้พร้อมแล้วจึงสามารถเริ่มการรับส่งข้อมูลได้

การเรียกใช้งานฟังก์ชันของ IPX สามารทำได้ 2 วิธีคือ

วิธีแรก เรียกจาก pointer ของ IPX โดยตรง

วิธีที่สอง เรียกใช้ interrupt 74h ซึ่งวิธีนี้มีข้อเสียอยู่คือ มี software บางตัวใช้ ineterrupt เบอร์นี้ด้วยเช่นกัน เพราะฉะนั้นถ้าจะใช้วิธีนี้ต้องแน่ใจว่าโปรแกรมของเราจะไม่ใช้ร่วมกันกับ(concurrent) software ที่ใช้ interrupt เบอร์นี้ด้วย

4.3 การติดต่อระหว่าง PC - PC โดยทั่วไปมีรูปแบบดังนี้

1.การเริ่มต้นการติดต่อ การทำงานขั้นตอนนี้เครื่องส่งจะทำการขอติดต่อกับเครื่องรับ และต้องรอจนกว่าเครื่องรับจะมีการตอบรับมาจึงจะเริ่มการติดต่อ รวมถึงการเปิด socket และการหาค่า Address ของเครื่องรับด้วย

2.รับและส่งข้อมูล ขั้นตอนนี้จะใช้ฟังก์ชันของ IPX ทำการส่งข้อมูลโดยมีข้อมูลจาก Address จากขั้นตอนแรกในการระบุจุดหมายปลายทาง และหมายเลขของ socket ที่ได้สร้างไว้ เป็นช่องทางในการรับส่งข้อมูลของโปรแกรม

3.การยกเลิกการติดต่อ หลังจากการรับส่งข้อมูลได้เสร็จสิ้นลง ก็จะทำการยกเลิกช่องทาง (socket) ในการติดต่อสื่อสาร

4.4 SOCKET

เป็นช่องทางที่ใช้ในการติดต่อข้อมูลระหว่าง PC กับ Network เพื่อให้สะดวกในการแยกแยะ Packet ได้ชัดเจน (ในกรณีที่มีโปรแกรมทำงานอยู่หลายโปรแกรมพร้อมกัน)ว่าเป็น Packet ของโปรแกรมใด ดังนั้นก่อนที่โปรแกรมประยุกต์ใดๆจะเริ่มทำงานติดต่อผ่าน Network จึงจำเป็นต้องสร้าง socket ขึ้นก่อน

ใน Netware นั้น ได้สงวน socket หมายเลข 0000A-3FFFh และ 8000h-FFFFh ไว้สำหรับใช้เอง ดังนั้นการสร้าง socket ของโปรแกรมใดๆจึงควรใช้หมายเลข socket ตั้งแต่ 4000h-7FFFh โดยชนิดของ socket มีอยู่ 2 แบบคือ

1.socketชั่วคราว คือจะมีการใช้งานจนกว่าจะปิดหรือลบ socket ออก หรือจนกระทั่งโปรแกรมสิ้นสุดลง

2.socketถาวร มักจะใช้ในโปรแกรมฝังตัว(Resident program) จะใช้งานจนกว่าจะมีการปิดหรือการยกเลิกอย่างชัดเจนจากโปรแกรม ดังนั้นการเปิด socket จึงจำเป็นต้องระบุชนิดของ socket ให้ตรงกับลักษณะของโปรแกรมที่สร้างขึ้นมาด้วย

4.5 ECB (Event Control Block)

ECB เป็นอีกโครงสร้างหนึ่งนอกเหนือจาก IPX header ที่จำเป็นในการติดต่อสื่อสารข้อมูล โดยมีหน้าที่ในการควบคุมลำดับของเหตุการณ์ในการรับส่งข้อมูล ซึ่ง ECB ทั้งแบบรับและแบบส่งจะมีโครงสร้างเหมือนกัน โดยโครงสร้างของ ECB แสดงได้ดังนี้

Link Address	Pointer
ESR Address	Pointer
Status flag	Byte
Complrtioin code	Byte
Socket number	Word
IPX workspace	Byte[2]
Driver workspace	Byte[12]
Immediate Address	Byte[6]
Fragment Count	Word
Fragment Address	Pointer
Fragment size 1	Word
Fragment Address 2	Pointer
Fragment size 2	Word

Link Address จะถูกให้โดย IPX ใช้ในการบอกสถานะ ECB เพื่อทำการรับส่งข้อมูล

ESR Address Address ของ routine ที่จะถูกเรียกให้ทำงาน เมื่อ Packet ถูกส่งมาถึงเครื่องรับ หรือกรณีที่ Packet ถูกส่งไปจนสำเร็จ

Status flag แสดง status ของ ECB

Completion code แสดงสถานะสุดท้ายของ ECB เมื่อเสร็จสิ้นการรับส่งข้อมูล

Socket number เบอร์ socket ที่ใช้ในการรับส่งข้อมูล

IPX workspace ถูกใช้โดย IPX

Drive workspace ถูกใช้โดย IPX สำหรับ Network driver

Immediate Address ค่า Address ของ bridge ในกรณีที่เป็นการส่งข้อมูลระหว่าง 2 Network หรือเป็นค่า Address ของเครื่องรับในกรณีที่ส่งข้อมูลใน Network เดียวกัน

Fragment count,Fragment size,Fragment address

Packet ที่จะใช้ส่งสามารถแบ่งได้เป็นส่วนๆ โดย Fragment count แสดงจำนวนส่วนของ Packet ที่ถูกแบ่ง Fragment size ใช้บอกขนาดของแต่ละส่วนที่ถูกแบ่ง ส่วน Fragment address แสดง pointer ที่ชี้ไปยัง buffer ของแต่ละส่วน

4.6 ESR

คือโปรแกรม Software Interrupt ซึ่งจะทำงานเมื่อการรับส่งข้อมูลโดย IPX Protocol ทำสำเร็จ โดย IPX จะเรียก ESR ที่ถูกเขียนขึ้นมาโดยโปรแกรมเมอร์ ให้ทำงานในลักษณะต่างๆตามต้องการ แต่ในกรณีเรา set ค่า 0 ที่ฟิลด์ event-service-routine ในโครงสร้าง ECB IPX จะไม่เรียกโปรแกรม ESR ขึ้นทำงาน เมื่อกระบวนการรับส่งข้อมูลสำเร็จ

ขณะที่ IPX เรียก ESR จะมีสิ่งที่เกิดขึ้นดังนี้

- Interrupt จะถูก disable
- ESR จะถูกเรียกใช้ในลักษณะ far call
- far pointer ที่ชี้ไปที่ ECB จะดูได้จาก register ES:DI
- cup register จะถูก save เก็บไว้
- DS registerไม่จำเป็นที่จะชี้ไปที่ Data area ของโปรแกรม Application
- ฟิลด์ in_use flag ใน ECB จะถูก set เป็น 0
- การเรียกใช้ฟังก์ชันของ DOS เช่น disk file I/O อาจจะไม่ปลอดภัย

การออกแบบการทำงานในส่วน ESR ควรออกแบบเหมือน Interrupt Handler ทั่วไป คือให้มีคำสั่งน้อยคำสั่ง เพื่อไม่ให้ใช้เวลามากเกินไปในการ process ESR และระวังในการใช้ฟังก์ชันของ DOS ซึ่งอาจจะทำให้เกิด error ได้

บทที่ 5 ทฤษฎีในส่วนของ Sound Blaster

ในที่นี้การศึกษาทฤษฎีและการทำงานของ sound card จะศึกษาจาก sound blaster ของบริษัท Creative Labs ซึ่งนิยมใช้กันแพร่หลาย

5.1 หลักการของเสียงและการแปลงสัญญาณเสียง

เสียงที่หมายถึงในโครงงานนี้จะกล่าวถึงเฉพาะเสียงคำพูดเท่านั้น และจะทำการพิจารณาใน 2 ขั้นตอนคือ ขั้นตอนการแปลงเสียงไปเป็นสัญญาณ digital และการแปลงกลับ โดยทั่วไปในการแปลงสัญญาณ analog ให้เป็นสัญญาณ digital สามารถทำได้โดยกระบวนการที่เรียกว่า sampling ใน sound blaster ก็เช่นเดียวกัน กระบวนการ sampling จะถูกทำ โดย Analog to Digital Converter(ADC)

ADC จะทำการสุ่มสัญญาณเสียงด้วยอัตราสุ่ม(sampling rate)ที่กำหนดไว้ล่วงหน้า และแปลงผลลัพธ์ที่ได้จากการสุ่มให้เป็นค่าตัวเลข(digit) อัตราสุ่ม อาจเรียกได้อีกอย่างว่า ความถี่ในการสุ่ม ซึ่งมีหน่วยเป็น Hertz ความถี่ที่เหมาะสมในการสุ่มสัญญาณตามทฤษฎีแชนนอน (Shannon theorem) กำหนดไว้ว่าต้องมีค่าอย่างน้อยเท่ากับ 2 เท่าของความถี่ของสัญญาณที่ถูกสุ่ม ในกรณีของเสียงพูดสัญญาณจะมีความถี่สูงสุดประมาณ 4,000 Hertz เพราะฉะนั้นความถี่ในการสุ่มต้องมีค่าอย่างน้อย 8,000 Hertz

นอกจากความถี่ในการสุ่มแล้ว ยังมี parameter อีกตัวหนึ่งที่เกี่ยวข้องกับการสุ่มนั่นคือ ขนาดในการสุ่ม(sampling size) ขนาดในการสุ่มจะเป็นตัวกำหนดช่วงกว้าง(range) ของสัญญาณ ใน sound blaster ทั่วไปจะมีขนาด 8 บิท แต่ในรุ่นใหม่ๆเช่น sound blaster 16 จะมีขนาด 16 บิท การสุ่มหนึ่งครั้งจะถูกแปลงเป็นเลขฐานสอง มีขนาดตามขนาดในการสุ่ม

ในขั้นตอนการแปลงกลับจากสัญญาณ digital เป็นสัญญาณเสียง จะถูกทำโดย Digital to Analog Converter(DAC)

คุณภาพของเสียงที่ได้จากการสุ่ม(sampling)เมื่อนำมาแปลงกลับ ขึ้นอยู่กับความถี่ ในการสุ่ม ยิ่งความถี่ในการสุ่มมากคุณภาพเสียงก็จะดีมากขึ้น แต่ปริมาณของข้อมูลก็จะมากขึ้นตามด้วย ดังนั้นการเลือกความถี่ในการสุ่มจึงพิจารณาจากความต้องการในคุณภาพของงาน และเนื้อที่ที่จะใช้ในการเก็บข้อมูล

5.2 คุณสมบัติของ Soundblaster และการนำมาใช้งาน

Sound card เป็นอุปกรณ์ที่ช่วยเพิ่มประสิทธิภาพการทำงานเกี่ยวกับเสียง ในคอมพิวเตอร์ส่วนบุคคล(pc) การพัฒนาทางเทคโนโลยีในยุคปัจจุบัน ทำให้คอมพิวเตอร์ส่วนบุคคลมีความสามารถในการทำงานด้านกราฟฟิคและเสียงมากขึ้น ความสามารถที่เพิ่มขึ้นเหล่านี้กำลังจะกลายเป็นมาตรฐานการทำงานของ pc โดยทั่วไป รู้จักกันในศัพท์ที่เรียกว่า "Multimedia"

ในโครงงานนี้จึงนำ Sound Blaster มาประยุกต์ใช้งานโดยการเขียนโปรแกรมควบคุม ซึ่งรายละเอียดของคุณสมบัติของ Sound Blaster และการนำมาใช้งานมีดังนี้

Sound Blaster

Sound Blaster เป็น sound card ยี่ห้อหนึ่งซึ่งนิยมใช้กันแพร่หลาย ของบริษัท Creative Labs มีการพัฒนาเป็นเวอร์ชั่นต่างๆหลายเวอร์ชั่น แต่ละเวอร์ชั่นมีคุณสมบัติแตกต่างกัน แต่หน้าที่การทำงานหลักๆของ Sound Blaster จะเหมือนกันคือ

1.สร้าง,เลียนแบบ เสียงดนตรีหรือเสียงธรรมชาติอื่นๆด้วยทฤษฎี FM Synthesis

2.บันทึก(record)สัญญาณเสียง ด้วยทฤษฎี sampling และเก็บข้อมูลอยู่ในรูปของสัญญาณ digital รวมทั้งเล่นกลับ(playback)ข้อมูล digital ให้อยู่ในรูปสัญญาณเสียงดั้งเดิม

3.มีการนำเทคโนโลยีด้านการบีบอัดข้อมูล(Data compression)มาใช้

นอกจากนี้ Sound Blaster ยังมีความสามารถที่จะติดต่อ กับหน่วยความจำของคอมพิวเตอร์ได้โดยตรง ไม่ต้องผ่าน CPU โดยใช้ DMA channel (Direct Memory Acces) ทำให้ประหยัดเวลาในการประมวลผลสัญญาณ

ใน Sound Blaster แต่ละตัวจะมี chip ประมวลผลสัญญาณเป็นของตนเอง ทำให้การประมวลผลสัญญาณเสียงสามารถทำได้ พร้อมๆกับที่เครื่องคอมพิวเตอร์กำลังทำงานอื่นโดยใช้ CPU

เอาท์พุทของ Sound Blaster คือลำโพง ส่วนอินพุทของ Sound Blaster มีได้หลายอย่าง แต่สิ่งที่เป็นพื้นฐานก็คือไมโครโฟน นอกจากนี้อาจเป็น ช่อง CD หรือช่อง line-in

ในโครงงานนี้เลือกใช้ Sound Blaster รุ่น SB16

การนำมาใช้งาน

ในที่นี้จะใช้งาน Sound Blaster ในส่วนที่เกี่ยวกับ digital sound channel เท่านั้นนั่นคือ พิจารณาการทำงานเฉพาะเรื่องการบันทึก(record)และการเล่นกลับ(playback) สัญญาณเสียง

สิ่งที่ต้องการในโครงงานนี้คือ ใช้ Sound Blaster บันทึกสัญญาณเสียงที่เครื่องคอมพิวเตอร์ต้นทาง และทำการส่งข้อมูลของสัญญาณเสียงที่ถูกแปลงเป็นข้อมูล digital แล้ว ไปยังเครื่องคอมพิวเตอร์ปลายทาง เมื่อเครื่องคอมพิวเตอร์ปลายทางได้รับ ก็จะทำการแปลงข้อมูลนั้นให้กลับเป็นสัญญาณเสียงโดยใช้ Sound Blaster (ในแต่ละเครื่องที่มีการติดต่อต้องมี Sound Blaster เป็นของตัวเอง)

รูปแบบของไฟล์เสียงที่ใช้ใน Sound Blaster มีหลายรูปแบบ เช่น Creative Voice File format (.VOC),Microsoft Waveform Audio File format(.WAV), Creative Music File format(.CMF) ในโครงงานนี้เลือกพิจารณา Creative Voice File format(.VOC) เพราะเป็นรูปแบบง่ายๆที่ใช้ทั่วไปเกี่ยวกับเสียงที่ไม่ใช่เสียงดนตรี

การเขียนโปรแกรมสั่งงาน Sound Blaster จะเรียกใช้ฟังก์ชันของไดรเวอร์ที่ชื่อ CT-VOICE.DRV เพื่อให้ Sound Blaster ทำหน้าที่ตามต้องการ

ฟังก์ชันที่สำคัญๆของไดรเวอร์ที่นำมาใช้งาน ก็เช่น ฟังก์ชันที่ check สถานะการทำงานเริ่มต้นของ sound card , ฟังก์ชัน record สัญญาณเสียง , ฟังก์ชัน playback ข้อมูลของสัญญาณเสียง เป็นต้น

ข้อมูลที่ได้จากการทำงานของฟังก์ชันของไดรเวอร์ จะอยู่ในรูปแบบของ VOC format แต่ในการทำงานจริงของโครงงานนี้จะไม่เก็บข้อมูลในรูปแบบของไฟล์ถาวร แต่จะใช้เนื้อที่ในหน่วยความจำเก็บเฉพาะส่วนที่เป็นข้อมูลจริงๆเท่านั้น เพื่อนำข้อมูลนั้นไปใช้ได้อย่างสะดวกและรวดเร็ว

รายละเอียดของ Creative Voice File format และ ฟังก์ชันใช้งานของไดรเวอร์ อยู่ในตอนต่อไป

5.3 โครงสร้างของ CT-Voice format (Creative Voice File format)

CT-Voice format เป็นรูปแบบของไฟล์ข้อมูล digital ที่ได้จากการสุ่มสัญญาณเสียง analog และถูกแปลงเป็นสัญญาณตัวเลขแล้ว รูปแบบของไฟล์ชนิดนี้กำหนดโดยบริษัท Creative Labs โดยไฟล์ชนิดนี้จะถูกระบุสกุลว่า .VOC

VOC file จะแบ่งออกเป็น 2 บล็อค คือ ส่วนหัว(header) และ ส่วนข้อมูล(actual data)

1.CT-Voice header block

เป็นบล็อคที่ใช้บ่งลักษณะของไฟล์ว่าเป็น CT-format file แยกเป็นส่วนย่อยๆ ดังนี้

Bytes $00-$13(0-19) บรรจุข้อความ "Creative Voice File"

Bytes $14-$15(20-21) บรรจุ offset address ของข้อมูล เป็นไบท์สูงและ ไบท์ต่ำ

Bytes $16-$17(22-23) บรรจุหมายเลขเวอร์ชั่นของ CT-Voice format

Bytes $18-$19(24-25) บรรจุค่าคอมพลีเมนท์ของหมายเลขเวอร์ชั่นซึ่งบวกกับค่า $1234 เป็นไบท์สูงและไบท์ต่ำ

2.Data blocks

ประกอบด้วยบล็อคย่อยๆ 8 บล็อค แต่ละบล็อคมีโครงสร้างคล้ายๆกัน

Block 0 - End Block

เป็นบล็อคที่ใช้บ่งว่าหมดข้อมูลแล้ว จะอยู่ที่ท้ายของ VOC file เท่านั้น

Structure of the End Block	
Block Type	1 byte = 0
Block Length	none
Data Bytes	none

Block 1 - New Voice Block

บรรจุข้อมูลแท้ๆที่สามารถนำมาเล่นกลับ (playback) ได้ SR คือ sampling rate ที่ใช้ในการบันทึก(record)เสียง ส่วน pack byte เป็นไบท์ที่ใช้ระบุว่าข้อมูลที่บันทึกมีการ pack หรือไม่ถ้ามีก็บอกให้ทราบว่าเป็นการ pack ชนิดไหน

Structure of the New Voice Block	
Block Type	1 byte =1
Block Length	3 bytes
SR Byte	1 byte
Pack Byte	1 byte = 0,1,2,3
Data Byte	x bytes

Block 2 - Subsequent Voice Block

เป็นบล็อคที่ใช้ในกรณีที่มีข้อมูลขนาดใหญ่เกินกว่าที่จะ load ลงหน่วยความจำภายในครั้งเดียว บล็อคนี้จะทำให้สามารถแบ่งข้อมูลออกเป็นบล็อคเล็กๆหลายบล็อคได้

Structure of the Subsequent Voice Block	
Block Type	1 byte = 2
Block Length	3 bytes
Data Byte	x bytes

Block 3 - Silence Block

เป็นบล็อคที่ใช้ในการลดขนาดของข้อมูล โดยจะแทนที่ข้อมูลที่ไม่มีเสียง(เงียบ) ด้วยบล็อคนี้

Structure of the Subsequent Voice Block	
Block Type	1 byte = 2
Block Length	3 bytes
Duration	2 byte
Sample rate	1 byte

Block 4 - Marker Block

ใช้ในการ mark ตำแหน่งของข้อมูลที่ต้องการใช้งาน(playback)

Structure of the Marker Voice Block	
Block Type	1 byte = 4
Block Length	3 bytes = 2
Marker	2 bytes

Block 5 - Message Block

ใช้ในการแทรกตัวอักษรแอสกี ลงไปใน VOC file

Structure of the Message Block	
Block Type	1 byte = 5
Block Length	3 bytes
Ascii Data	x byte
End Character	1 byte

Block 6 - Repeat Block

เป็นบล็อคที่ใช้กำหนดให้นำข้อมูลที่เริ่มตั้งแต่จุดนั้นไปเล่นกลับซ้ำตามจำนวนที่กำหนด

Structure of the Repeat Block	
Block Type	1 byte = 6
Block Length	3 bytes = 2
Counter	2 bytes

Block 7 - Repeat End Block

เป็นบล็อคที่ใช้กำหนดจุดสิ้นสุดของข้อมูลที่จะให้เล่นซ้ำ

Structure of the Repeat End Block	
Block Type	1 byte = 7
Block Length	3 bytes = 0

Block 8 - Extended Header Block

ใช้ใน SB Pro เพราะว่าใน data block 1 ให้รายละเอียดของข้อมูลไม่ได้ครบถ้วน ในกรณีที่เป็นข้อมูลเสียงแบบ stero

Structure of the Extended Header Block	
Block Type	1 byte = 8
Block Length	3 bytes = 4
Sample Rate	2 byte
Data Mode	1 byte

5.4 ฟังก์ชันใช้งานของ CT-driver (CT-VOICE.DRV)

Function 0 (BX=0) : กำหนดเวอร์ชั่นของไดรเวอร์

Determine driver version	
Input	BX = 00
Output	AH = Main number AL = Sub-number
Remarks	none

Function 1 (BX=1) : ตั้งค่า port address

Set Port Address	
Input	BX = 01 AX = Port Address
Output	none
Remarks	can only be called before function 3

Function 2 (BX=2) : ตั้งค่า interrupt

Set Interrupt	
Input	BX = 02 AX = Interrupt Address
Output	none
Remarks	can only be called befors function 3

Function 3 (BX=3) : check สถานะต่างๆของ sound card ให้อยู่ในสภาวะที่ถูกต้อง

Initialize driver	
Input	BX = 03
Output	AX =0 Successful AX = 1 SB not found Ax = 2 Port address error AX = 3 Interrupt error
Remarks	none

Function 4 (BX=4) : เปิด/ปิด ลำโพง

Loudspeaker on/off	
Input	BX = 04 AL = 0 off Al = 1 on
Output	none
Remarks	none

Function 5 (BX=5) : ตั้งค่า statusword address

ค่า statusword เป็นค่าที่ใช้แสดงสถานะการทำงานของ sound card ถ้ามีค่าใดๆที่ไม่เท่ากับ 0 แสดงว่าข้อมูล sample กำลังถูกดำเนินการอยู่(playback/record) ถ้ามีค่าเป็น 0 ข้อมูล sample ไม่ได้ถูกดำเนินการใดๆ หรือดำเนินการเสร็จเรียบร้อยแล้ว

Set Status Address	
Input	BX = 05 ES:DI= Status Address
Output	none
Remarks	none

Function 6 (BX=6) : เล่นกลับ(playback)ข้อมูล (แปลงข้อมูล digital เป็นสัญญาณเสียงดั้งเดิม)

Sample Playback	
Input	BX = 06 ES:DI=Sample Address
Output	none
Remarks	Statusword is changed according to encountered marker block

Function 7 (BX=7) : บันทึกข้อมูล (แปลงสัญญาณเสียง เป็นสัญญาณ digital และ เก็บไว้)

Record Sample	
Input	BX = 07 AX = Sampling rate DX:CX = Length ES:DI = Sample Address
Output	none
Remarks	none

Function 8 (BX=8) : ยกเลิกการทำงานทุกอย่างของ sound card

Abort Sample	
Input	BX =08
Output	none
Remarks	Statusword = 0

Function 9 (BX=9) : ยกเลิกการติดตั้ง driver เดิม และทำการติดตั้งใหม่

De-install driver	
Input	BX =09
Output	none
Remarks	none

Function 10 (BX=10) : หยุดการทำงานของฟังก์ชัน playback ชั่วขณะ

Pause Sample	
Input	BX =10
Output	AX = 0 Successful AX = 1 Not Successful
Remarks	none

Function 11 (BX=11) : ใช้ในการสั่งให้ฟังก์ชัน playback ทำงานต่อไป หลังจากสั่งให้หยุดด้วยฟังก์ชัน 10 แล้ว

Cotinue Sample	
Input	BX =11
Output	AX = 0 Successful AX = 1 Not Successful
Remarks	none

Function 12 (BX=12) : ทำหน้าที่หยุดการทำงานเล่นกลับ(playback)แบบซ้ำๆ

Interrupt Loop	
Input	BX = 12 AX = 0 at end of loop AX = 1 immediatly
Output	AX = 0 Successful AX = no loop being executed
Remarks	none

Function 13 (BX=13) : เป็นฟังก์ชันที่เพิ่มขึ้นมาให้ผู้ใช้กำหนดหน้าที่และการใช้งานเอง

User-defined function	
Input	BX =13 DX:AX Function Address
Output	ES:BX Address of the current data block
Remarks	none

บทที่ 6 การทดลอง

6.1 ขั้นตอนการทดลอง

1.ศึกษาการทำงนของ Protocol IPX และเขียนโปรแกรมรับส่งข้อมูลด้วย Prorocol IPX จากนั้นนำไปทดสอบการทำงานของโปรแกรมรับส่งข้อมูลที่ต้องการ

2.ศึกษาการทำงานของ Sound card และการเขียนโปรแกรมควบคุม Sound card ทั้งในด้านการบันทึกเสียง และการเล่นกลับ จากนั้นนำไปทดสอบการทำงานของโปรแกรม

3.นำการทดลองขั้นตอนที่ 1 และ 2 มารวมกัน โดยการทำงานของโปรแกรมในขั้นตอนนี้จะเริ่มจากการบันทึกสัญญาณเสียง นำข้อมูลที่ได้ส่งไปยังเครื่องคอมพิวเตอร์ปลายทาง ที่ปลายทางจะทำการรับข้อมูล และแปลงข้อมูลนั้นกลับเป็นสัญญาณเสียงตามเดิม

4.พัฒนาในด้าน application ต่างๆ ของโปรแกรม เช่น การส่งข้อความขอติดต่อจากผู้ส่งไปยังผู้รับ , check สถานะความจำของ Hardware เป็นต้น

หมายเหตุ

แนวความคิดเบื้องต้นของโครงงานนี้ เดิมทีต้องการให้การทำงานขั้นสมบูรณ์ของการติดต่อระหว่าง PC 2 เครื่องเป็นแบบ full-duplex และจะทำการพัฒนาการทำงานขั้นต่อไป ให้สามารถติดต่อกันได้ตั้งแต่ 3 เครื่องขึ้นไปในลักษณะคล้ายกับการประชุมทางโทรศัพท์ แต่เนื่องจากข้อจำกัดของ Sound Blaster ที่นำมาใช้ในการแปลงสัญญาณเสียง ซึ่งจะถูกใช้งานใน 2 ขั้นตอนคือ ขั้นตอนของการบันทึก(record)แปลงสัญญาณเสียงให้อยู่ในรูปของข้อมูลตัวเลข และขั้นตอนของการเล่นกลับ(playback)แปลงข้อมูลตัวเลขกลับเป็นสัญญาณเสียง ทั้ง 2 ขั้นตอนนี้ไม่สามารถทำงานได้ในเวลาเดียวกัน จึงก่อให้เกิดปัญหาเมื่อต้องการให้การติดต่อเป็นแบบ full-duplex เพราะเมื่อมีการส่งข้อมูลพร้อมกันทั้ง 2 ด้านของการติดต่อ ข้อมูลนั้นเมื่อถึงฝ่ายรับแล้วจะยังไม่สามารถแปลงกลับเป็นสัญญาณเสียงได้ทันที ต้องรอจนกว่าผู้ใช้เครื่องคอมพิวเตอร์เสร็จสิ้นการใช้งาน Sound Blaster ในขั้นตอนการบันทึกสัญญาณเสียงก่อน

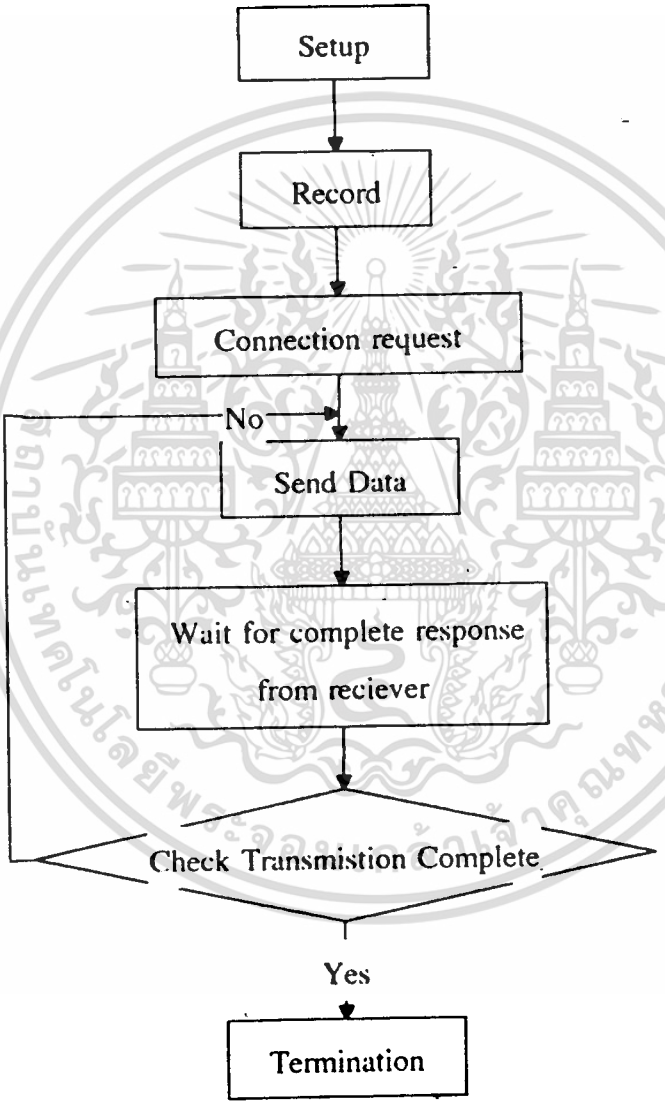
ฉะนั้นแนวความคิดของโครงงานจึงถูกปรับเปลี่ยนให้เหมาะสมกับขีดความสามารถในการทำงานของ Sound Blaster ดังนั้นลักษณะของโครงงานจึงเป็นการฝากข่าวสารทางเสียงส่งไปยังผู้รับทันที โดยในขณะนั้นผู้รับข่าวสารต้องกำลังทำงานบนเครื่องคอมพิวเตอร์ในระบบ LAN โดยเข้า login ในชื่อ user ของตนเอง

6.2 ลักษณะการทำงานของโครงงานนี้เมื่อสมบูรณ์แล้ว เป็นดังนี้

ลักษณะของโปรแกรมการทำงานจะแบ่งเป็น 2 ส่วนคือ โปรแกรมส่ง และโปรแกรมรับ

1.โปรแกรมส่ง มีการทำงานดังนี้

เมื่อต้องการติดต่อกับผู้รับ ก็จะเรียกโปรแกรมส่งขึ้นมาทำงาน โดยระบุ user ID. ของผู้รับ ต่อจากนั้นก็จะทำการบันทึกสัญญาณเสียงโดย Sound Blaster เมื่อได้ข้อมูลที่แปลงแล้ว ก็จะทำการส่งข้อมูลนั้นไปยังผู้รับด้วย Protocal IPX โดยมีการทำงานดังบล็อคไดอะแกรมข้างล่าง



บล็อคไดอะแกรมด้านส่ง(send)

setup

- ในส่วนของการบันทึกเสียง

จะทำการ initial CT-VOICE driver เพื่อเช็คสถานะการทำงานต่างๆของไดรเวอร์ว่าถูกต้องหรือไม่และให้พร้อมในการเรียกฟังก์ชันใช้งานต่างๆของไดรเวอร์

- ในส่วนของการส่งข้อมูล

1. check ว่า IPX install หรือไม่
2. open socket ที่เป็นช่องทางในการรับส่งข้อมูล
3. หาค่า Address ของผู้รับ (ทั้ง Network Address,Node Address)
4. ถ้าเป็นการส่งข้อมูลข้าม Networkmedkisk ทำการหา Immediate Address ด้วย

record ข้อมูล ผู้ส่งจะเริ่มการบันทึกสัญญาณเสียงโดยใช้ Sound Blaster ข้อมูลที่ได้จะถูกเก็บไว้ใน buffer โดยจะมีการคืนค่า address เริ่มต้น และขนาดของ buffer ให้กับการทำงานในขั้นต่อไปเพื่อใช้ในการส่งข้อมูล

Connection_request ทำการส่งข้อความไปบอกทางด้านผู้รับ ว่าขณะนี้ข้อมูลพร้อมจะส่งมาให้ และรอการตอบรับจากผู้รับ ถ้ามีการตอบรับก็จะทำงานในขั้นตอน Send data ต่อไป ในกรณีที่รอสัญญาณตอบรับ จนถึงค่าเวลาค่าหนึ่งแล้วยังไม่มีการตอบรับ ก็จะแจ้งให้ผู้ส่งทราบว่า ขณะนี้ไม่สามารถส่งข้อมูลได้ แล้วทำการสิ้นสุดโปรแกรม

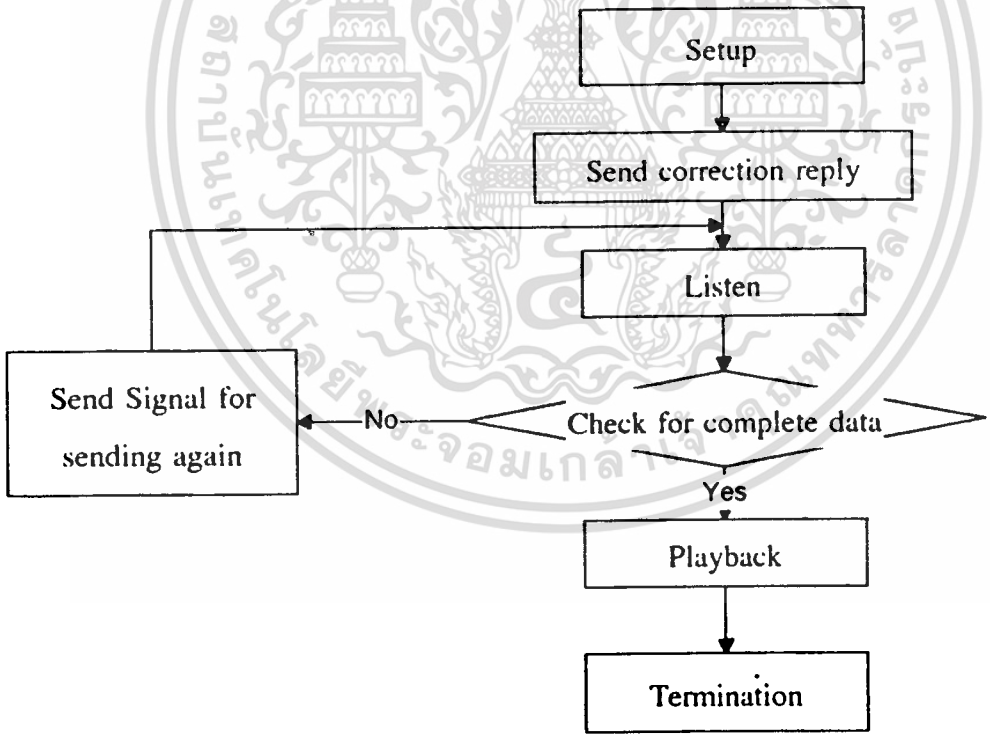
Send_data เมื่อได้รับสัญญาณตอบรับจากผู้รับแล้ว ก็เริ่มทำการส่งข้อมูลสัญญาณเสียง ที่แปลงเป็นสัญญาณ digital โดย Sound Blaster ออกไปบน Network ไปยังผู้รับ โดยข้อมูล Address ของผู้รับ ในการกำหนดจุดหมายด้วย Protocol IPX ต่อไป (โดยทำการส่ง start frame เป็นการบอกทางเครื่องรับให้พร้อมรับข้อมูล แล้วทำการแบ่งข้อมูลเป็นบล็อคๆ แล้วส่งข้อมูลออกไปจนครบ จากนั้นจะส่ง stop frame พร้อมกับจำนวน packet ที่ส่งไป เพื่อให้ผ่ายรับ check ได้ว่าส่งข้อมูลครบหรือไม่)

Wait_for_complete_response_from reciever หลังจากการส่งข้อมูลต่างๆเสร็จสิ้น ก็จะรอการตอบรับ จากผู้รับว่าได้รับข้อมูลถูกต้องหรือไม่ ถ้ามีการตอบรับว่าถูกต้อง ก็จะถือว่าการส่งรับข้อมูลเสร็จสิ้นสมบูรณ์ แต่ถ้ามีการตอบรับว่าทางด้านรับได้ข้อมูลไม่ครบ ก็จะทำการส่งข้อมูลทั้งหมดใหม่อีกครั้ง และรอการตอบรับเช่นเดิม

Termination หลังจากการส่งรับข้อมูลเสร็จสิ้นลงแล้วก็จะทำการเลิกโปรแกรม ขั้นตอนนี้จะทำการ free เนื้อที่หน่วยความจำที่จองไว้เป็น buffer และยกเลิกไดรเวอร์ของ Sound Blaster และ close socket ที่เปิดไว้ จากนั้นจึงเป็นการสิ้นสุดการทำงานอย่างสมบูรณ์

2.โปรแกรมรับ มีการทำงานดังนี้

เมื่อผู้รับได้ข้อความขอติดต่อจากผู้ส่งแล้ว ก็จะทำการเรียกโปรแกรมรับข้อมูล พร้อมกับระบุ user ID ของผู้ส่ง(ผ่านทาง Command Line เช่นกัน) จากนั้นทำการรอรับข้อมูลที่จะส่งมา เมื่อรับข้อมูลแล้ว ก็จะนำข้อมูลที่ได้มา playback ให้เป็นสัญญาณเสียงต่อไป โดยมีรายละเอียดการทำงานตาม Block diagram ดังนี้



บล็อคไดอะแกรมด้านรับ(listen)

Setup

- ในส่วนของการเล่นกลับ จะใลักษณะเช่นเดียวกับด้านส่ง
- ในส่วนของการรับข้อมูล จะมีลักษณะเช่นเดียวกับด้านส่งดังนี้

1.check ว่า IPX install หรือไม่

2.open socket ที่เป็นช่องทางในการรับส่งข้อมูล

3.หาค่า Address ของผู้รับ (ทั้ง Network Address และ Node Address)

4.ถ้าเป็นการส่งข้อมูลข้าม Network ทำการหา Intermediate Address ด้วย

Listen การทำงานส่วนนี้จะทำการรับข้อมูลที่ส่งมาเป็นบล็อคๆ จากนั้นนำข้อมูลแต่ละบล็อคนี้มารวมกันให้เป็นข้อมูลเดิม เก็บไว้ใน buffer รวมทั้ง start frame เพื่อเป็นตัวบอกว่า packet ต่อไปเป็นข้อมูลของจำนวน packet ที่ถูกส่งมา เพื่อตรวจสอบความถูกต้อง ในการรับข้อมูลต่อไป

Check for complete data หลังจากการรับข้อมูลเสร็จสิ้น ก็จะรับข้อมูลที่บอกจำนวน packet ที่ถูกส่งมาใน End frame มาทำการตรวจสอบว่า การรับนั้นได้รับจำนวน packet เท่ากับที่ถูกส่งมาหรือไม่ หมายถึงการตรวจสอบความถูกต้องในการรับ ถ้าข้อมูลที่รับมาถูกต้องเรียบร้อย ก็จะนำข้อมูลพร้อม playback ต่อไป แต่ถ้าข้อมูลที่รับได้ผิดพลาด ก็จะทำการส่งสัญญาณออกไปทางฝ่ายส่งให้ส่งข้อมูลมาใหม่อีกครั้ง และทำการรับข้อมูลและตรวจสอบความถูกต้องต่อไป

Playback ข้อมูลที่ได้จากขั้นตอนก่อน จะถูกนำมาแปลงกลับเป็นสัญญาณเสียง เมื่อทำการแปลงข้อมูลเสร็จแล้ว ก็จะล้างข้อมูลเดิมที่มีอยู่ใน buffer แล้วกลับไปรอรับข้อมูลใหม่

Termination การทำงานเช่นเดียวกับด้านส่ง

บทที่ 7 สรุปผลและวิจารณ์ผลการทดลอง

จากการทดลองการใช้งานโปรแกรมบันทึกสัญญาณเสียง และเล่นกลับข้อมูลที่บันทึก สามารถทำงานได้ถูกต้อง แต่ข้อมูลที่ได้จากการบันทึก โดยใช้ sampling rate 5,000 Hz ซึ่งเป็นอัตราที่ต่ำสุดที่ SB16 สามารถทำได้ในรูปแบบของไฟล์เสียงแบบ voc เพื่อให้ขนาดข้อมูลมีขนาดเล็กที่สุด ยังคงมีขนาดใหญ่ ซึ่งการบันทึกใน 1 วินาที จะได้ขนาดข้อมูล 5,000 ไบต์ ในการทดลองใช้งานจริงๆ สามารถจองได้เวลามากสุด 30 วินาที ซึ่งเป็นขนาดข้อมูล 150,000 ไบต์

ข้อมูลที่ได้ก็ยังมีขนาดใหญ่อยู่ หากจะนำไปใช้งานจริงอาจจะสิ้นเปลืองเนื้อที่ในการจอง buffer จึงอาจต้องมีการศึกษาเกี่ยวกับการบีบอัดข้อมูล เพื่อประโยชน์ในการลดพื้นที่หน่วยความจำและเวลาในการส่งข้อมูล

ปัญหาอีกอย่างที่เกิดขึ้นในการส่งข้อมูลคือ เนื่องจากความเร็วในการทำงานของ คอมพิวเตอร์ทางด้านรับและด้านส่ง ไม่สามารถทำงานทันกันได้ เช่นการใช้เครื่องคอมพิวเตอร์ ด้านส่งมีอัตราเร็วกว่าด้านรับ ทำให้ข้อมูลทางด้านรับที่ได้ไม่ถูกต้อง เนื่องจากทางด้านรับไม่ สามารถรับข้อมูลได้ทัน เมื่อเทียบกับความเร็วในการส่ง และ Protocal IPX ไม่มีการตรวจ สอบว่าด้านรับ รับข้อมูลได้หรือไม่ ดังนั้นจึงมีทางแก้ไข 2 ทางคือ

1.เปลี่ยนการใช้ Protocal IPX เป็น Protocal SPX

2.เขียนโปรแกรมเช็คการรับส่งข้อมูลเอง

การทำงานที่ได้เมื่อรันโปรแกรมเป็นไปอย่างถูกต้อง และในส่วนของการนำไปพัฒนาขั้นต่อไป อาจพัฒนาในด้านการบีบอัดข้อมูล และการเขียนโปรแกรมควบคุม file server เพื่อฝากข้อมูลไว้ในกรณีที่ผู้รับไม่ได้ login ใช้งานคอมพิวเตอร์อยู่ในขณะนั้น

หนังสืออ้างอิง

1.Network Programming in C, Barry Nance, Que corporation.

2.Programmer 's Guide NetWare, Charles G. Rose,Mac-Graw Hill Inc.

3.ระบบเครือข่ายคอมพิวเตอร์ LAN, อัครเสน สุนทรผ่อง, จักร พิชัยสรทัต, บริษัท ซีเอ็ดยูเคชั่น จำกัด.

4.LAN Protocol Handbook,Mark A. Miiler,P.E.

5.คู่มือการใช้ NetWare ,ต้น ตัณฑ์สุทธิวงศ์,สุพจน์ ปุณณชัยยะ

6,Sound Blaster :The Official Book ,Richard Heimlich with David M. Golden Ivan Luk and Peter M. Ridse,Osborne/MacGraw-Hill.

7.The Sound Blaster Book , Axel Stolz ,Abacus

กิตติกรรมประกาศ

ขอขอบคุณ ดร.มนัส สังวรศิลป์ ผู้ให้ความเอื้อเฟื้อในการจัดหาอุปกรณ์ที่ใช้ในการทำปริญญานิพนธ์นี้จนสำเร็จลุล่วง

ขอขอบคุณ พี่ไก่ และ พี่ลิง ที่ให้คำปรึกษาเกี่ยวกับระบบ LAN ,Sounf Blaster และการเขียนโปรแกรม และขอขอบคุณ พี่ตี๋ (จิรศักดิ์ เหลืองอุไร)ที่ให้คำปรึกษาและคำแนะนำในการเขียนโปรแกรมตลอดมา

ภาคผนวก

/* Source code ของโปรแกรมด้านส่ง (s_send.c) */

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <mem.h>
#include <conio.h>
#include <ctype.h>
//#include "ipx.h"
#include <io.h>
#include <bios.h>
#include <fcntl.h>
#include <alloc.h>
#define  FALSE  0
#define  TRUE   1
#define  DWORD  unsigned long
#define  WORD   unsigned int
#define  BYTE   unsigned char
#define  VOCPTR char far*
#define  IPXINSTALL 0x7f
#define  GET_CON 0xde


typedef struct
          { unsigned int  checksum;
            unsigned int  length;
            unsigned char transport_control;
            unsigned char packet_type;
            unsigned char dest_network_number[4];
            unsigned char dest_node_address[6];
            unsigned int  dest_socket;
            unsigned char source_network_number[4];
            unsigned char source_node_address[6];
            unsigned int  source_socket;
          }IPXHEADER;
```

```
typedef struct
        { void far      *link_address;
          void far      (*esr)(void);
          unsigned char in_use;
          unsigned char completion_code;
          unsigned int  socket_number;
          unsigned char workspace[4];
          unsigned char driver_workspace[12];
          unsigned char immediate_address[6];
          unsigned int  packet_count;
          struct
            { void far *address;
                unsigned int length;
            } packet[2];
        }ECB;

typedef struct
        { unsigned char network_num[4];
          unsigned char node_address[6];
        } NET_ADDRESS;

typedef struct
        { unsigned int  len;
          unsigned char count;
          unsigned char connection_num[100];
        } CON_ADDRESS;

  ECB           ecb_send,ecb_read;
  IPXHEADER     head_send,head_read;

  WORD   voc_status_word     = 0;
  WORD   voc_err_stat        = 0;
  BYTE   vocfile_headerlength = 0;
  BYTE   vocdriver_installed  = 0;
  WORD   vocdriver_version    = 0;
  VOCPTR voc_ptr_to_driver    = 0;
```

```
void _print_voc_errmessage(void)
/* INPUT: None; OUTPUT: None; PURPOSE: Displays SB error as text. */
{
  switch (voc_err_stat) {
    case 100 : printf(" CT-VOICE.DRV driver file not loaded");break;
    case 110 : printf(" No memory free for driver file ");break;
    case 120 : printf(" Wrong driver file ");break;
    case 200 : printf(" VOC file not found ");break;
    case 210 : printf(" No memory free for VOC file ");break;
    case 220 : printf(" File is not in VOC format ");break;
    case 300 : printf(" Memory allocation error occurred ");break;
    case 400 : printf(" No Sound Blaster card found ");break;
    case 410 : printf(" Wrong port address used ");break;
    case 420 : printf(" Wrong interrupt used ");break;
     }
}

WORD _voc_getversion(void)
/* INPUT: None; OUTPUT: DRV ver. no.; PURPOSE: Determines DRV ver. no. */
{
  WORD    vdummy;
  asm {
    mov      bx,0
    call     voc_ptr_to_driver
    mov      vdummy, ax
     }
  return(vdummy);
}

WORD _voc_init_driver(void)
/* INPUT: None; OUTPUT: Error msg. no., depending on results
 * PURPOSE: Initializes driver software. */
{
     int            filehandle;
```

```
        unsigned long    filesize;
        WORD             blocksize;
        char huge       *filepointer;
        WORD             byte_read;
        BYTE             check;
        WORD             segment;
        WORD             vseg;
        WORD             vofs;
        WORD             vout;
    vocdriver_installed = FALSE;
/* Driver file not found */
    if (_dos_open("CT-VOICE.DRV",O_RDONLY,&filehandle) != 0)
       {
        voc_err_stat = 100;
        return(FALSE);
       }
    filesize = filelength(filehandle);
    blocksize = (filesize + 15L) /16;
/* Insufficient memory */
    check = _dos_allocmem(blocksize,&segment);
    if (check != 0)
       {
        voc_err_stat = 110;
        return(FALSE);
       }
    FP_SEG(voc_ptr_to_driver) = segment;
    FP_OFF(voc_ptr_to_driver) = 0;
    filepointer = (char huge*)voc_ptr_to_driver;
/* Load driver file */
    do
      {
       _dos_read(filehandle,filepointer,0x8000,&byte_read);
       filepointer += byte_read ;
      } while (byte_read == 0x8000) ;
    _dos_close(filehandle) ;
/* Determine driver version */
```

```
   vocdriver_version = _voc_getversion();
/* Start driver */
   vseg = FP_SEG(&voc_status_word);
   vofs = FP_OFF(&voc_status_word);
   asm {
      mov      bx,1
      mov      ax,0x220
      call     voc_ptr_to_driver
      mov      bx,2
      mov      ax,7
      call     voc_ptr_to_driver
      mov      bx,3
      call     voc_ptr_to_driver
      mov      vout,ax
      mov      bx,5
      mov      es,vseg
      mov      di,vofs
      call     voc_ptr_to_driver
       }
/* No Sound Blaster card found */
   if (vout == 1)
      {
       voc_err_stat = 400;
       return(FALSE);
      }
/* Wrong port address used */
   if (vout == 2)
      {
       voc_err_stat = 410;
       return(FALSE);
      }
/* Wrong interrupt number used */
   if (vout == 3)
      {
       voc_err_stat = 420;
       return(FALSE);
```

```
    }
  vocdriver_installed = TRUE;
  return(TRUE);
}
void _voc_deinstall_driver(void)
/* INPUT: None; OUTPUT: None; PURPOSE: Switches off driver, releases memory. */
{
  if (vocdriver_installed)
    {
     asm {
          mov   bx,9
          call  voc_ptr_to_driver
           }
     _dos_freemem(FP_SEG(voc_ptr_to_driver));
    }
}
void _voc_set_speaker(BYTE on_off)
/* INPUT: TRUE for speaker on, FALSE for speaker off; OUTPUT: None;
 * PURPOSE: Toggles SB card output. */
{
  asm {
    mov      bx,4
    mov      al,on_off
    call     voc_ptr_to_driver
     }
}
void _voc_set_filter_r(void)
{
  asm {
    mov      bx,22
    mov      ax,0
    mov      cx,0
    call     voc_ptr_to_driver
     }
}
void _voc_stop(void)
```

```
/* INPUT: None; OUTPUT: None; PURPOSE: Stops a sample. */
{
  asm {
    mov     bx,8
    call    voc_ptr_to_driver
     }
}
void _voc_input(unsigned char huge *bufferaddress,unsigned long int count)
/* INPUT: Pointer to allocated memory; OUTPUT: None; PURPOSE: Records sample. */
{  unsigned int   low,high;

  WORD  vseg;
  WORD  vofs;
  vseg = FP_SEG(bufferaddress);
  vofs = FP_OFF(bufferaddress);
  low =(count & 0xffff);
  high=(count >> 16);

  asm {
    mov     bx,7
    mov     es,vseg
    mov     di,vofs
    mov     ax,5000
    mov     dx,high
    mov     cx,low
    call    voc_ptr_to_driver
     }
}

void textnumerror(void)
{
  printf("Error # %d: ",voc_err_stat);
  _print_voc_errmessage();
   exit(voc_err_stat);
}
```

```
void record(unsigned char huge *buf,unsigned long int count)
{
  unsigned char huge *voc_buffer;
  char far* old_voc_ptr;
  char ch;
 // char buf[30000];
//  long int  count=0;


  clrscr();
  if (_voc_init_driver() == FALSE) textnumerror();
  printf("\nPress r to record sample : ");
  ch = getche();
  if (ch == 'r')
    {
          printf("\nPress any key to stop recording.");
          _voc_set_speaker(FALSE);
          _voc_set_filter_r();
          voc_buffer = buf;//(char far*)malloc(30000);
          if (voc_buffer == NULL) textnumerror();
          _voc_input(voc_buffer,count);

        do{}
          while ((voc_status_word != 0)&& (kbhit() == 0));
//          _voc_stop();
     }

//   farfree(voc_buffer);
  _voc_deinstall_driver();
}

void ipx_install(void)
{
    asm mov ax,0x7a00;
    asm int 0x2f;
    if(_AL != 0xff)
```

```
    {  printf("\n IPX not installed.");
         exit(0);
    }
  //ipx_spx = MP_FP(_ES,_IS);
  printf("\n IPX is installed now.");
}

void create_socket(unsigned int socket)
{
  asm mov al,0x00;
  asm mov bx,0x00;
  asm mov dx,socket;
  asm int 0x7a;
  if (_AL != 0x00)
      {  printf("\n Can't open socket ");
        exit(0);
      }
  printf("\n socket complte ");
}

void get_connection_number(char *name,CON_ADDRESS  *con_buff_ptr)
{  struct
   {  unsigned int  len;
      unsigned char buff_type;
      unsigned int  object_type;
      unsigned char object_name_len;
      unsigned char name[47];
   } request_buff;

   request_buff.len = 51;
   request_buff.buff_type       = 0x15;
   request_buff.object_type     = 0x0100;
   request_buff.object_name_len = (unsigned char) strlen(name);
   strcpy(request_buff.name,name);
   con_buff_ptr->len = 101;
```

```
    _SI = FP_OFF ((void far*)&request_buff);
    _DS = FP_SEG ((void far*)&request_buff);
    _DI = FP_OFF ((void far*)con_buff_ptr);
    _ES = FP_SEG ((void far*)con_buff_ptr);

    asm mov ah,0xe3;
    asm int 0x21;

}

void get_internet_add(unsigned char number,NET_ADDRESS *net_buff_ptr)
{  struct
    {  unsigned int  len;
        unsigned char type;
        unsigned char con_number;
    } request_buff;
  struct
    {  unsigned int  len;
        unsigned char network_num[4];
        unsigned char node_address[6];
        unsigned int  socket;
    } reply_buff;

    request_buff.len        = 2;
    request_buff.type       = 0x13;
    request_buff.con_number = number;
    reply_buff.len          = 12;

    _SI = FP_OFF ((void far*)&request_buff);
    _DS = FP_SEG ((void far*)&request_buff);
    _DI = FP_OFF ((void far*)&reply_buff);
    _ES = FP_SEG ((void far*)&reply_buff);
```

```
    asm mov ah,0xe3;
    asm int 0x21;

    if (_AL != 0)
        {  printf("\n Unable get internet address.");
           exit(0);
        }
    memcpy(net_buff_ptr->network_num,reply_buff.network_num,4);
    memcpy(net_buff_ptr->node_address,reply_buff.node_address,6);
}

void get_local_target(NET_ADDRESS net_buff,unsigned char *imme_add_ptr)
{  struct
    {  unsigned char network_num[4];
       unsigned char node_address[6];
       unsigned int  d_socket;
    } request_buff;

    memcpy(request_buff.network_num,net_buff.network_num,4);
    memcpy(request_buff.node_address,net_buff.node_address,6);
    request_buff.d_socket  = 0x5555;

    _SI = FP_OFF ((void far*)&request_buff);
    _DS = FP_SEG ((void far*)&request_buff);
    _DI = FP_OFF ((void far*)imme_add_ptr);
    _ES = FP_SEG ((void far*)imme_add_ptr);

    asm mov bx,0x02;
    asm int 0x7a;

    if (_AL != 0)
        {  printf("\n Unable get local target.");
           exit(0);
        }
}
```

```
void get_local_con_number(unsigned char *user_num)
{
    asm mov ah,0xdc;
    asm mov al,0x00;
    asm int 0x21;
    *user_num = _AL;
}

void  get_user_info(unsigned char user_num,unsigned char *name)
{   struct
      {  unsigned int    len;
         unsigned char   type;
         unsigned char   num;
      } request_buff;
   struct
      {  unsigned int    len;
         unsigned char   object_id[4];
         unsigned int    object_end;
         unsigned char   object_name[48];
         unsigned char   log_time[7];
          } reply_buff;

         request_buff.len   = 2;
         request_buff.type  = 0x16;
         request_buff.num   = user_num;

         reply_buff.len  = 61;

    _SI = FP_OFF ((void far*)&request_buff);
    _DS = FP_SEG ((void far*)&request_buff);
    _DI = FP_OFF ((void far*)&reply_buff);
    _ES = FP_SEG ((void far*)&reply_buff);

    asm mov ah,0xe3;
    asm int 0x21;
```

```
    if (_AL != 0)
        {  printf("\n Unable get user_info.");
           exit(0);
        }
    memset(name,0,48);
    memcpy(name,reply_buff.object_name,48);
}

void ipx_breath(void)
{   asm mov bx,0x000a;
    asm int 0x7a;
}

void ipx_send( ECB          *ecb_send_ptr,
               IPXHEADER    *head_send_ptr,
               NET_ADDRESS  net_buff,
               unsigned char *imme_add_ptr,
               unsigned char *packet_ptr)
{
  ecb_send_ptr->in_use=1;


        memset(ecb_send_ptr,0,sizeof(ECB));
        ecb_send_ptr->socket_number     = 0x4444;
        memcpy(ecb_send_ptr->immediate_address,imme_add_ptr,6);
        ecb_send_ptr->packet_count      = 2;
        ecb_send_ptr->packet[0].address = head_send_ptr;
        ecb_send_ptr->packet[0].length  = sizeof(IPXHEADER);
        ecb_send_ptr->packet[1].address = packet_ptr;
        ecb_send_ptr->packet[1].length  = 500;


        head_send_ptr->packet_type = 4;
        memcpy(head_send_ptr->dest_network_number,net_buff.network_num,4);
        memcpy(head_send_ptr->dest_node_address,net_buff.node_address,6);
        head_send_ptr->dest_socket  = 0x5555;
```

```
        _ES = FP_SEG ((void far*) ecb_send_ptr);
        _SI = FP_OFF ((void far*) ecb_send_ptr);
        asm mov bx,0x03;
        asm int 0x7a;
//      ipx_breath();


        printf("\n completion code: %x  ",ecb_send_ptr->completion_code);
        ipx_breath();
}

void ipx_listen( ECB            *ecb_read_ptr,
                 IPXHEADER      *head_read_ptr,
                 unsigned char  *packet_ptr
               )
{
    memset(ecb_read_ptr,0,sizeof(ECB));
    memset(head_read_ptr,0,sizeof(IPXHEADER));

    ecb_read_ptr->socket_number     = 0x4444;
    ecb_read_ptr->packet_count      = 2;
    ecb_read_ptr->packet[0].address = head_read_ptr;
    ecb_read_ptr->packet[0].length  = sizeof(IPXHEADER);
    ecb_read_ptr->packet[1].address = packet_ptr;
    ecb_read_ptr->packet[1].length  = 500;

    _ES = FP_SEG ((void far*) ecb_read_ptr);
    _SI = FP_OFF ((void far*) ecb_read_ptr);

     asm mov bx,0x0004;
     asm int 0x7a;

    while(ecb_read_ptr->in_use)
        {   ipx_breath();
        }
   if (ecb_read_ptr->completion_code != 0)
```

```
        {   printf("\nlisten error");
            printf("\ncompletion code : %x",ecb_read_ptr->completion_code);
            exit(0);
        }


}

int send( ECB               *ecb_send_ptr,
          IPXHEADER         *head_send_ptr,
          NET_ADDRESS       net_buff,
          unsigned char     *imme_add_ptr,
          unsigned char huge  *data_ptr,
          unsigned long int   count)
{ static unsigned int   i,j;
  unsigned long int     x;
  unsigned char  packet[500],num_pac[20],num_count[25],response[8];

   i=0; j=0; x=0;
   i = count/500; j = count%500;
   if (j != 0).   i = i+1;
   itoa(i,num_pac,10);
   ltoa(count,num_count,10);

   //send start signal//
   memset(packet,0,500);
   memset(packet,0x01,8);
   strcat(packet,num_count);
   printf("\nmemory : %s ",packet);
   ipx_send(ecb_send_ptr,head_send_ptr,net_buff,imme_add_ptr,packet);
   ipx_breath();
   delay(100);

   //send data//
   while(i)
     {  j=0;
```

```
        memset(packet,0,500);
        while((j<500) && (x<count))
           { packet[j] = *(data_ptr+x);
                x++;j++;
           }
        ipx_send(ecb_send_ptr,head_send_ptr,net_buff,imme_add_ptr,packet);
        ipx_breath();   i--;
        delay(100);
        printf("\n i : %d ",i);
        printf("\n j : %d ",j);
  }

//send stop signal and number of sended packet//
memset(packet,0,500);
memset(packet,0x1b,8);
strcat(packet,num_pac);
printf("\nend packet: %s ",packet);
delay(100);
ipx_send(ecb_send_ptr,head_send_ptr,net_buff,imme_add_ptr,packet);
ipx_breath();

//receive complete response//
memset(packet,0,500);
ipx_listen(&ecb_read,&head_read,packet);
memset(response,0x06,8);
if(strncmp(response,packet,8))
      {
       printf("\nUNCOMPLETE SENDING");
       return(0);
      }
 else
   {
       printf("\nCOMPLETE SENDING");
       return(1);
   }
```

```
}


void send_message( unsigned char  number,
                   unsigned char  *mes
                 )
{  struct
        {  unsigned int    len;
           unsigned char   type;
           unsigned char   con_count;
           unsigned char   con_number;
           unsigned char   m_len;
           unsigned char   message[55];      //LIMIT AT 45 BYTE
        }request_buff;
   struct
        {  unsigned int    len;
           unsigned char   con_count;
           unsigned char   result;
        }reply_buff;



        request_buff.type       = 0x00;
        request_buff.con_count  = 1;
        request_buff.con_number = number;
        request_buff.m_len      = 55;
        memcpy(request_buff.message,mes,55);
        request_buff.len        = (request_buff.m_len+4);


        reply_buff.len        =0x04;


        _SI = FP_OFF ((void far*)&request_buff);
        _DS = FP_SEG ((void far*)&request_buff);
        _DI = FP_OFF ((void far*)&reply_buff);
        _ES = FP_SEG ((void far*)&reply_buff);
        asm mov ah,0xe1;
```

```
        asm int 0x21;

  //    printf("\nAL : %x",_AL); //00 or fe
    //    printf("\nresult : %x",reply_buff.result);
        /* 00   success
           fc   rejected
           fd   invalid connection number
           ff   block*/
}

void setup ( unsigned char number,
             unsigned char *name
           )
{  unsigned char  packet[500],
                  response[8];

   unsigned char  *call;

   unsigned int   a;

   call = (unsigned char *) calloc(56,1);
   if (call == NULL)
     {   printf("\nCAN'T ALLOCATE MEMORY (call)");
         exit(0);
     }
   sprintf(call,"YOU HAVE MESSAGE! PLEASE RUN:PRO_WAIT ");
   strupr(name);
   strcat(call,name);

     a = strlen(call);
     memset(&(call[a]),0x20,(55-a));
     call[55]=0;
     send_message(number,call);

     //wait for complete connection //
     memset(packet,0,500);
```

```
    ipx_listen(&ecb_read,&head_read,packet);
    memset(response,0x06,8);
    if(strncmp(response,packet,8))
        {
         printf("\nCAN'T CONNECT WITH RECEIVER");
         exit(0);
        }
   else
    {
         printf("\nCONNECTION SUCCESS");
    }
}


void main(int argc, char *argv[])
{ unsigned char     imme_add[6],
                    sender_num,
                    sender_name[48],
                    *filename="5.voc",
                    *tail=".voc";
  unsigned char huge *data;
  unsigned          c;
  unsigned int      send_socket,bytes,n;
  unsigned long int  count=0,number;
  int               handle;
  CON_ADDRESS       con_buff;
  NET_ADDRESS       net_buff;
  char              string[2];
  FILE*             stream;



        clrscr();
        if (argc != 2)
          {
              printf("\nCOMMAND NOT CORRECT\n");
```

```
        printf("\nUSAGE  : pro_send [userid]\n");
        exit(0);
    }
ipx_install();

send_socket=0x4444;
create_socket(send_socket);

get_connection_number(argv[1],&con_buff);

if (_AL != 0)
    {  printf("\n Unable get connetion number.");
       exit(0);
    }

switch(con_buff.count)
 {
   case 0  : printf("\n User ID not  available");
             exit(0);

   case 1  : break;

   default : printf("\n There are user id logging in more than one");
             exit(0);
 }
printf("\n con_number of %s is %d ",argv[1],con_buff.connection_num[0]);

get_internet_add(con_buff.connection_num[0],&net_buff);

get_local_target(net_buff,imme_add);

get_local_con_number(&sender_num);

get_user_info(sender_num,sender_name);

//culculate and allocate memory//
```

```
    printf("Enter the recording time(sec) : ");
    gets(string);
    number = atoi(string);
    count = (number*5000);

    if ((data = farmalloc(count)) == NULL)
     {   printf("\nCAN'T ALLOCATE MEMORY (data)");
         exit(0);
     }

//record data//
    record(data,count);
    printf("\nNOW RECORD COMPLETE AND PREPARE TO SEND MESSAGE");
    printf("\nIF WANT TO TO TERMINATE -> PRESS ANY KEY");

//request to connect with receiver//
    setup(con_buff.connection_num[0],sender_name);

//send data until complete//
    while(send(&ecb_send,&head_send,net_buff,imme_add,data,count)!=1)
     {
        printf("\nSENDING NOT COMPLETE -> SEND AGAIN");
     }

    printf("\nEnter filename(sec) : ");
    gets(filename);
 strcat(filename,tail);

    //save source file//
    if((stream = fopen(filename, "wb")) == NULL)
    {
      printf("\nCAN'T OPEN THIS FILE : %s",filename);
      exit(0);
    }

n=fwrite(data,5000,(count/5000),stream);
```

```
    printf("\nnumber of item : %d",n);
        fclose(stream);

/*      if (_dos_creat("1.voc", _A_NORMAL, &handle) != 0)
          {  printf("\nUnable to create 1.voc");
             exit(0);
          }

        if (_dos_write(handle,data,count, &c) != 0)
          {
             printf("Unable to write to 1.voc");
             exit(0);
           }

        _dos_close(handle);*/

       farfree(data);

}
```

```
/* Source code ของโปรแกรมด้านรับ(s_wait.c) */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <mem.h>
#include <conio.h>
#include <io.h>
#include <bios.h>
#include <fcntl.h>
#include <alloc.h>
#define FALSE 0
#define TRUE  1
#define DWORD  unsigned long
#define WORD   unsigned int
#define BYTE   unsigned char
#define VOCPTR char far*
typedef struct
        { unsigned int  checksum;
          unsigned int  length;
          unsigned char transport_control;
          unsigned char packet_type;
          unsigned char dest_network_number[4];
          unsigned char dest_node_address[6];
          unsigned int  dest_socket;
          unsigned char source_network_number[4];
          unsigned char source_node_address[6];
          unsigned int  source_socket;
        }IPXHEADER;

typedef struct
        { void far      *link_address;
          void far      (*esr)(void);
          unsigned char in_use;
          unsigned char completion_code;
          unsigned int  socket_number;
```

```
        unsigned char workspace[4];
        unsigned char driver_workspace[12];
        unsigned char immediate_address[6];
        unsigned int  packet_count;
        struct
          {  void far *address;
             unsigned int length;
          } packet[2];
       }ECB;

typedef struct
       {  unsigned char network_num[4];
          unsigned char node_address[6];
       } NET_ADDRESS;

typedef struct
       {  unsigned int  len;
          unsigned char count;
          unsigned char connection_num[100];
       } CON_ADDRESS;

void ipx_send( ECB*,IPXHEADER*,NET_ADDRESS*,unsigned char*,unsigned char*);
void ipx_listen( ECB *,IPXHEADER *,unsigned char *) ;

  WORD    voc_status_word      = 0;
  WORD    voc_err_stat         = 0;
  BYTE    vocfile_headerlength = 0;
  BYTE    vocdriver_installed  = 0;
  WORD    vocdriver_version    = 0;
  VOCPTR  voc_ptr_to_driver    = 0;

  IPXHEADER           head_read,head_send;
  ECB                 ecb_read,ecb_send;
  unsigned char       packet[500];
  unsigned char       flag=0;
  unsigned char huge  *data;
```

```
void _print_voc_errmessage(void)
{
  switch (voc_err_stat) {
    case 100 : printf(" CT-VOICE.DRV driver file not loaded");break;
    case 110 : printf(" No memory free for driver file ");break;
    case 120 : printf(" Wrong driver file ");break;
    case 200 : printf(" VOC file not found ");break;
    case 210 : printf(" No memory free for VOC file ");break;
    case 220 : printf(" File is not in VOC format ");break;
    case 300 : printf(" Memory allocation error occurred ");break;
    case 400 : printf(" No Sound Blaster card found ");break;
    case 410 : printf(" Wrong port address used ");break;
    case 420 : printf(" Wrong interrupt used ");break;
     }
}
VOCPTR _voc_get_buffer(char *voicefile)
/* INPUT : Filename as string
   OUTPUT: Pointer to buffer with VOC data
   PURPOSE: Load a file into memory and returns the value pointer to
            the data upon being successfully loaded. */
{
      int       filehandle;
      long      filesize;
      WORD      blocksize,byte_read;
      char huge *filepointer;
      VOCPTR    bufferaddress;
      BYTE      check;
      WORD      segment;
/* VOC file not found */
   if(_dos_open(voicefile,O_RDONLY,&filehandle) != 0)
     {
      voc_err_stat = 200;
      return(FALSE);
     }
```

```
   filesize = filelength(filehandle);
   blocksize = (filesize + 15L)/16;
/*Insufficient memory for the VOC file*/
   check = _dos_allocmem(blocksize,&segment);
   if (check != 0)
      {
       voc_err_stat = 210;
       return(FALSE);
      }
   FP_SEG(bufferaddress) = segment;
   FP_OFF(bufferaddress) = 0;
   filepointer = (char huge*)bufferaddress;
/*Load the sample file*/
   do
     {
      _dos_read(filehandle,filepointer,0x8000,&byte_read);
      filepointer += byte_read;
     }while (byte_read == 0x8000);
   _dos_close(filehandle);
   vocfile_headerlength = (BYTE)bufferaddress[20];
   printf("\nvocfile_headerlength : %x",vocfile_headerlength);
   return(bufferaddress);
}
WORD _voc_getversion(void)
{
   WORD    vdummy;
   asm {
      mov      bx,0
      call     voc_ptr_to_driver
      mov      vdummy, ax
       }
   return(vdummy);
}

WORD _voc_init_driver(void)
{
```

```
    int             filehandle;
    unsigned long   filesize;
    WORD            blocksize;
    char huge       *filepointer;
    WORD            byte_read;
    BYTE            check;
    WORD            segment;
    WORD            vseg;
    WORD            vofs;
    WORD            vout;
  vocdriver_installed = FALSE;
/* Driver file not found */
  if (_dos_open("CT-VOICE.DRV",O_RDONLY,&filehandle) != 0)
    {
     voc_err_stat = 100;
     return(FALSE);
    }
  filesize = filelength(filehandle);
  blocksize = (filesize + 15L) /16;
/* Insufficient memory */
  check = _dos_allocmem(blocksize,&segment);
  if (check != 0)
    {
     voc_err_stat = 110;
     return(FALSE);
    }
  FP_SEG(voc_ptr_to_driver) = segment;
  FP_OFF(voc_ptr_to_driver) = 0;
  filepointer = (char huge*)voc_ptr_to_driver;
/* Load driver file */
  do
   {
    _dos_read(filehandle,filepointer,0x8000,&byte_read);
    filepointer += byte_read ;
   } while (byte_read == 0x8000) ;
  _dos_close(filehandle) ;
```

```
/* Determine driver version */
   vocdriver_version = _voc_getversion();
/* Start driver */
   vseg = FP_SEG(&voc_status_word);
   vofs = FP_OFF(&voc_status_word);
   asm {
      mov      bx,1          /*set port*/
      mov      ax,0x220
      call     voc_ptr_to_driver
      mov      bx,2          /*set interrupt*/
      mov      ax,7
      call     voc_ptr_to_driver
      mov      bx,3
      call     voc_ptr_to_driver
      mov      vout,ax
      mov      bx,5
      mov      es,vseg
      mov      di,vofs
      call     voc_ptr_to_driver
       }
/* No Sound Blaster card found */
   if (vout == 1)
      {
       voc_err_stat = 400;
       return(FALSE);
      }
/* Wrong port address used */
   if (vout == 2)
      {
       voc_err_stat = 410;
       return(FALSE);
      }
/* Wrong interrupt number used */
   if (vout == 3)
      {
       voc_err_stat = 420;
```

```
        return(FALSE);
       }
    vocdriver_installed = TRUE;
    return(TRUE);
}
void _voc_deinstall_driver(void)
/* INPUT: None; OUTPUT: None; PURPOSE: Switches off driver, releases memory. */
{
    if (vocdriver_installed)
       {
        asm {
             mov   bx,9
             call  voc_ptr_to_driver
              }
        _dos_freemem(FP_SEG(voc_ptr_to_driver));
       }
}
void _voc_set_speaker(BYTE on_off)
/* INPUT: TRUE for speaker on, FALSE for speaker off; OUTPUT: None;
 * PURPOSE: Toggles SB card output. */
{
    asm {
       mov       bx,4
       mov       al,on_off
      call      voc_ptr_to_driver
        }
}
void _voc_set_filter_p(void)
{
    asm {
       mov       bx,22
       mov       ax,1
       mov       cx,0
       call      voc_ptr_to_driver
        }
}
```

```
void _voc_stop(void)
/* INPUT: None; OUTPUT: None; PURPOSE: Stops a sample. */
{
  asm {
    mov     bx,8
    call    voc_ptr_to_driver
     }
}
void _voc_output(unsigned char huge *bufferaddress)
{
  WORD  vseg,vofs;

  _voc_set_speaker(TRUE);
  vseg = FP_SEG(bufferaddress);
  vofs = FP_OFF(bufferaddress);
  //+vocfile_headerlength;
  asm {
     mov     bx,6
     mov     es,vseg
     mov     di,vofs
   call    voc_ptr_to_driver
     }
}
void textnumerror(void)
{
  printf("Error # %d: ",voc_err_stat);
  _print_voc_errmessage();
   exit(voc_err_stat);
}
void  play_back(unsigned char huge *buf)
{
  char far* voc_buffer;
  char far* old_voc_ptr;
  char ch;

   if (_voc_init_driver() == FALSE) textnumerror();
```

```
    printf("\nPress p to playback sample : ");
    ch = getche();
    if (ch == 'p')
     {
       _voc_set_speaker(TRUE);
       _voc_set_filter_p();
       _voc_output(buf);
     do{}while (voc_status_word != 0); //&& (kbhit() == 0));
     }
    _voc_deinstall_driver();
}

void ipx_install(void)
{
     asm mov ax,0x7a00;
     asm int 0x2f;
     if(_AL != 0xff)
       {  printf("\n IPX not installed.");
            exit(0);
       }
     //ipx_spx = MP_FP(_ES,_IS);
     printf("\n IPX is installed now.");
}

void create_socket(unsigned int socket)
{
     asm mov al,0x00;
     asm mov bx,0x00;
     asm mov dx,socket;
     asm int 0x7a;
     if (_AL != 0x00)
          {  printf("\n Can't open socket ");
             exit(0);
          }
     printf("\n socket complte ");
}
```

```
void get_connection_number(char *name,CON_ADDRESS *con_buff_ptr)
{  struct
     {  unsigned int  len;
        unsigned char buff_type;
        unsigned int  object_type;
        unsigned char object_name_len;
        unsigned char name[47];
     } request_buff;

     request_buff.len = 51;
     request_buff.buff_type        = 0x15;
     request_buff.object_type      = 0x0100;
     request_buff.object_name_len = (unsigned char) strlen(name);
     strcpy(request_buff.name,name);
     con_buff_ptr->len = 101;

     _SI = FP_OFF ((void far*)&request_buff);
     _DS = FP_SEG ((void far*)&request_buff);
     _DI = FP_OFF ((void far*)con_buff_ptr);
     _ES = FP_SEG ((void far*)con_buff_ptr);

     asm mov ah,0xe3;
     asm int 0x21;

     if (_AL != 0)
        {  printf("\n Unable get connetion number.");
           exit(0);
        }
     if (con_buff_ptr->count == 0)
        {  printf("\n User ID not  available");
           exit(0);
        }
     if (con_buff_ptr->count > 1)
        {  printf("\n There are user id logging in more than one");
           exit(0);
```

```
        }
}

unsigned char get_local_con_number(void)
{
    asm mov ah,0xdc;
    asm mov al,0x00;
    asm int 0x21;
    return _AL;
}

void get_internet_add(unsigned char number,NET_ADDRESS *net_buff_ptr)
{  struct
    {  unsigned int  len;
        unsigned char type;
        unsigned char con_number;
    } request_buff;
  struct
    {  unsigned int  len;
        unsigned char network_num[4];
        unsigned char node_address[6];
        unsigned int  socket;
    } reply_buff;

    request_buff.len        = 2;
    request_buff.type       = 0x13;
    request_buff.con_number = number;
    reply_buff.len          = 12;

    _SI = FP_OFF ((void far*)&request_buff);
    _DS = FP_SEG ((void far*)&request_buff);
    _DI = FP_OFF ((void far*)&reply_buff);
    _ES = FP_SEG ((void far*)&reply_buff);

    asm mov ah,0xe3;
    asm int 0x21;
```

```
    if (_AL != 0)
        {  printf("\n Unable get internet address.");
           exit(0);
        }
    memcpy(net_buff_ptr->network_num,reply_buff.network_num,4);
    memcpy(net_buff_ptr->node_address,reply_buff.node_address,6);
}

void get_local_target(NET_ADDRESS net_buff,unsigned char *imme_add_ptr)
{   struct
    {  unsigned char network_num[4];
       unsigned char node_address[6];
       unsigned int  d_socket;
    } request_buff;

    memcpy(request_buff.network_num,net_buff.network_num,4);
    memcpy(request_buff.node_address,net_buff.node_address,6);
    request_buff.d_socket  = 0x4444;

    _SI = FP_OFF ((void far*)&request_buff);
    _DS = FP_SEG ((void far*)&request_buff);
    _DI = FP_OFF ((void far*)imme_add_ptr);
    _ES = FP_SEG ((void far*)imme_add_ptr);

    asm mov bx,0x02;
    asm int 0x7a;

    if (_AL != 0)
        {  printf("\n Unable get local target.");
           exit(0);
        }
}

void ipx_breath(void)
{   asm mov bx,0x000a;
```

```
    asm int 0x7a;
}

void ipx_send( ECB              *ecb_send_ptr,
               IPXHEADER        *head_send_ptr,
               NET_ADDRESS      *net_buff_ptr,
               unsigned char    *imme_add_ptr,
               unsigned char    *packet_ptr
             )
{
  ecb_send_ptr->in_use=1;


    memset(ecb_send_ptr,0,sizeof(ECB));
    ecb_send_ptr->socket_number     = 0x5555;
    memcpy(ecb_send_ptr->immediate_address,imme_add_ptr,6);
    ecb_send_ptr->packet_count      = 2;
    ecb_send_ptr->packet[0].address = head_send_ptr;
    ecb_send_ptr->packet[0].length  = sizeof(IPXHEADER);
    ecb_send_ptr->packet[1].address = packet_ptr;
    ecb_send_ptr->packet[1].length  = 500;


    head_send_ptr->packet_type = 4;
    memcpy(head_send_ptr->dest_network_number,net_buff_ptr->network_num,4);
    memcpy(head_send_ptr->dest_node_address,net_buff_ptr->node_address,6);
    head_send_ptr->dest_socket  = 0x4444;


        _ES = FP_SEG ((void far*) ecb_send_ptr);
        _SI = FP_OFF ((void far*) ecb_send_ptr);
        asm mov bx,0x03;
        asm int 0x7a;


   printf("\n completion code: %x  ",ecb_send_ptr->completion_code);
   ipx_breath();
}
```

```
void ipx_listen( ECB                *ecb_read_ptr,
                 IPXHEADER          *head_read_ptr,
                 unsigned char      *packet_ptr
               )
{
    memset(ecb_read_ptr,0,sizeof(ECB));
    memset(head_read_ptr,0,sizeof(IPXHEADER));

    ecb_read_ptr->socket_number     = 0x5555;
    ecb_read_ptr->packet_count      = 2;
    ecb_read_ptr->packet[0].address = head_read_ptr;
    ecb_read_ptr->packet[0].length  = sizeof(IPXHEADER);
    ecb_read_ptr->packet[1].address = packet_ptr;
    ecb_read_ptr->packet[1].length  = 500;

    _ES = FP_SEG ((void far*) ecb_read_ptr);
    _SI = FP_OFF ((void far*) ecb_read_ptr);

     asm mov bx,0x0004;
     asm int 0x7a;

        while(ecb_read_ptr->in_use)
        {   ipx_breath();
        }
     if(ecb_read_ptr->completion_code != 0)
        {  printf("\n Listen error!! ");
           printf("\n completion code : %x",ecb_read_ptr->completion_code);
           exit(0);
        }

}

void listen( ECB                *ecb_read_ptr,
             IPXHEADER          *head_read_ptr,
             NET_ADDRESS        *net_buff_ptr,
             unsigned char      *imme_add_ptr,
```

```
                unsigned char       *filename,
                unsigned long int   *count_ptr)
{
  unsigned char       end[8],start[8],num_pac[20];
  int                 i,j,handle,number=0,get_pac=0;
  unsigned int        bytes;
  unsigned            c;
  unsigned long int   count=0;

    i=1;
    memset(end,0x1b,8);
    memset(start,0x01,8);

    get_pac=0;
    while(i)
        {  memset(packet,0,500);
           ipx_listen(ecb_read_ptr,head_read_ptr,packet);
            if(strncmp(start,packet,8))
              {   i = strncmp(end,packet,8);
                  if(i)
                    {   //data packet then save for file//
                        for(j=0;j<500;j++)
                        { data[count] = packet[j];//*(pac+j);    //packet[j];
                          count++;
                        }
                        get_pac++;
                    }

              }
            else
                {     /*staet packet then create  file */
                    printf("\nstart packet: %s",packet);
                    *count_ptr=atol(packet+8);
                    printf("\nmemory : %ld",*count_ptr);

                    //load data from filename//
```

```
                if ((data = farmalloc(*count_ptr)) == NULL)
                  {   printf("\nNot enough memory to allocate buffer");
                        exit(0);  // terminate program if out of memory
                  }


            }

      }
 /* stop packet then close file */

 printf("\nend packet: %s",packet);
 number=atoi(packet+8);
 printf("\nnumber of packet is %d",number);
 printf("\nreceived  packet  : %d",get_pac);
 printf("\ncount form start packet : %ld",*count_ptr);
 printf("\ncount from receiver     : %ld",count);


 //check for complete receive//
 if((number==get_pac) && (*count_ptr==count))
   {  //sending complete signal//
        memset(packet,0,500);
        memset(packet,0x06,8);
        ipx_send(&ecb_send,&head_send,net_buff_ptr,imme_add_ptr,packet);
   }
else
   {  //sending  uncomplete signal//
        memset(packet,0,500);
        memset(packet,0x00,8);
        ipx_send(&ecb_send,&head_send,net_buff_ptr,imme_add_ptr,packet);
   }

 play_back(data);

 farfree(data);
```

```
}

void main(int argc, char *argv[])
{
  unsigned int        listen_socket,bytes;
  unsigned long int   count;
  int                 handle,i,n;
  unsigned            c;
  unsigned char       imme_add[6],
                      *filename="5.voc",
                      *tail=".voc";
  CON_ADDRESS         con_buff;
  NET_ADDRESS         net_buff;
  FILE *              stream;


    clrscr();
    ipx_install();

    listen_socket=0x5555;
    create_socket(listen_socket);

    get_connection_number(argv[1],&con_buff);
    printf("\ncon_number of %s is %d ",argv[1],con_buff.connection_num[0]);

    get_internet_add(con_buff.connection_num[0],&net_buff);

    get_local_target(net_buff,imme_add);

    //get output filename//
    printf("\nEnter filename(sec) : ");
```

```
    gets(filename);
    strcat(filename,tail);

    //send for connection response//
    memset(packet,0,500);
    memset(packet,0x06,8);
    ipx_send(&ecb_send,&head_send,&net_buff,imme_add,packet);

    //wait for data//
    printf("\nlisten");
    listen(&ecb_read,&head_read,&net_buff,imme_add,filename,&count);

    //load data from filename//
    if ((data = farmalloc(count)) == NULL)
        {   printf("\nNot enough memory to allocate buffer");
            exit(0);  // terminate program if out of memory
        }

    memset(data,0,count);
    printf("\n count : %ld",count);

   if((stream = fopen(filename, "wb")) == NULL)
        {
          printf("\nCAN'T OPEN THIS FILE : %s",filename);
          exit(0);
        }
     n=fwrite(data,5000,(count/5000),stream);
     printf("\nnumber of item : %d",n);
     fclose(stream);
     _dos_open(filename,O_RDONLY,&handle);
     _dos_read(handle,data,count,&bytes);
     _dos_close(handle);

   play_back(data);
   farfree(data);
}
```