บริการส่งโทรสารผ่านเครือข่ายอินเทอร์เน็ต

FAX SERVER ON INTERNET

โดย

นางสาวจิราภรณ์ จุติสุขสันติ์ รหัส 36014080

นางสาวฐิตินันท์ ลิมปาภินันท์ รหัส 36014130

อาจารย์ที่ปรึกษา

ผศ.ดร.สุวิพล สิทธิชีวภาค

อ.เกรียงไกร วงศ์โรจนภรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

ปริญญานิพนธ์ปีการศึกษา 2539

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง บริการส่งโทรสารผ่านเครือข่ายอินเทอร์เน็ต

Fax Server on Internet

ผู้จัดทำ

1. นางสาวจิราภรณ์ จุติสุขสันติ์ 36014080

2. นางสาวฐิติ นันท์ ลิมปาภินันท์ 36014130

..... อาจารย์ที่ปรึกษา

(ผศ.ดร.สุวิพล สิทธิชีวภาค)

..... อาจารย์ที่ปรึกษา

(อ.เกรียงไกร วงศ์โรจนภรณ์)

บริการส่งโทรสารผ่านเครือข่ายอินเทอร์เน็ต
FAX SERVER ON INTERNET

โดย จิราภรณ์ จุติสุชสันติ์ 36014080
ฐิตินันท์ ลิมปาภินันท์ 36014130

อาจารย์ที่ปรึกษา ผศ.ดร.สุวิพล สิทธิชีวภาค
อ.เกรียงไกร วงศ์โรจนภรณ์

บทคัดย่อ

โครงงานนี้เป็นการเขียนแอพพลิเคชั่นบนวินโดวส์ เพื่อรับ-ส่งโทรสารผ่านเครือข่ายอินเทอร์เน็ต โดยใช้โปรแกรมเดลไฟล์ (Delphi) เป็นเครื่องมือสำหรับพัฒนาโปรแกรม หลักการทำงานของโปรแกรมจะประกอบด้วย 2 ส่วนหลัก คือ ส่วนที่ทำการรับ-ส่งข้อมูลระหว่างเครื่องโทรสารกับเครื่องให้บริการ และส่วนที่ทำการรับ-ส่งไฟล์ข้อมูลระหว่างเครื่องให้บริการผ่านเครือข่ายอินเทอร์เน็ต โดยในส่วนที่สองจะให้บริการแก่ยูสเซอร์ในการส่งข่าวสารด้วย ดังนั้นปลายทางที่ต้องการ อาจเป็นได้ทั้งเครื่องโทรสารหรือเครื่องยูสเซอร์

ABSTRACT

This project presents the application for sending-receiving fax on internet by using Delphi for program development. The main idea of this program has two important tasks, the first is receiving-sending on telephone network. The other is receiving-sending files between hosts on internet. By the rear part also supports users for arrangment (send-receive) their files and they can select the terminal as facsimile as user.

สารบัญ

สารบัญรูปภาพ

ลำดับภาพ **หน้า**

สารบัญตาราง

ลำดับตาราง **หน้า**

บทที่ 1
บทนำ

ปัจจุบันการติดต่อสื่อสารทางด้านข้อมูล เป็นสิ่งที่สำคัญอย่างมากในโลกของการแข่งขันทางธุรกิจ การศึกษา และรวมไปถึงการดำเนินชีวิตประจำวัน อินเทอร์เน็ต (Internet) ก็เป็นอีกเทคโนโลยีหนึ่งที่ถูกนำมาใช้อย่างแพร่หลายเพื่อรองรับการเจริญเติบโตทางด้านการสื่อสารข้อมูล ทั้งนี้เนื่องจากระบบอินเทอร์เน็ตเป็นระบบเครือข่ายคอมพิวเตอร์ที่มีผู้ใช้งานมากที่สุดในโลก โดยเฉพาะในประเทศไทยได้มีหลายบริษัท หลายหน่วยงาน ต่างให้ความสนใจและทำการศึกษาระบบเครือข่ายนี้เป็นจำนวนมาก เพราะระบบอินเทอร์เน็ตเป็นระบบเครือข่ายคอมพิวเตอร์สากลที่มีการเชื่อมโยงกันทั่วทุกมุมโลก ซึ่งทำให้บุคคลหรือหน่วยงานทั้งหลายสามารถใช้ประโยชน์จากระบบเครือข่ายนี้ได้อย่างมหาศาล

การพัฒนาของอินเทอร์เน็ตอยู่บนรากฐานของโปรโตคอล TCP/IP ซึ่งเป็นหนทางของการที่จะทำให้ระบบเชื่อมโยงคอมพิวเตอร์เป็นระบบเปิด เชื่อมต่อได้หลากหลายผลิตภัณฑ์ หนทางของการพัฒนาแอพพลิเคชั่นบนอินเทอร์เน็ตในปัจจุบันนี้มีมากขึ้น และกำลังก้าวเข้าสู่การใช้ประโยชน์ในองค์กร เช่น ใช้เป็นวีดีโอคอนเฟอเรนซ์, ใช้ในการประชุม, การจัดส่งเอกสาร, การเรียกค้นข้อมูลข่าวสารในองค์กร เป็นต้น ปัจจุบันการเชื่อมต่ออินเทอร์เน็ตกับคอมพิวเตอร์ขององค์กรนั้น ส่วนใหญ่จะเป็นการใช้งานส่วนบุคคล โดยใช้เครื่องคอมพิวเตอร์ โมเด็ม และสายโทรศัพท์ จำนวนหนึ่งชุดต่อบุคลากรหนึ่งคน ซึ่งถ้าหากบุคลากรทุกคนในองค์กรต้องการใช้งานอินเทอร์เน็ตพร้อมๆ กัน เครื่องคอมพิวเตอร์ทุกเครื่องในองค์กรก็ต้องทำการติดตั้งโมเด็มและสายโทรศัพท์ทั้งหมดซึ่งถือเป็นการสิ้นเปลืองทรัพยากรเป็นอย่างยิ่ง ในความเป็นจริงแล้วเราสามารถต่อเครื่องคอมพิวเตอร์ทุกเครื่องในองค์กรกับระบบอินเทอร์เน็ตโดยใช้โมเด็มเพียงชุดเดียวได้ และพนักงานทุกคนก็ไม่ต้องหมุนโมเด็มทุกครั้งเมื่อต้องการใช้งานอินเทอร์เน็ต โดยการขอใช้บริการแบบนิติบุคคลหรือองค์กร (Corporation User) โดยหน่วยงานหรือองค์กรนั้นต้องมีระบบเครือข่ายที่ทำงานด้วยโปรโตคอล TCP/IP ได้

นอกจากนี้พบว่า การติดต่อสื่อสารข้อมูลในปัจจุบันนอกจากจะใช้โทรศัพท์แล้ว ยังมีการใช้เครื่องโทรสารอย่างแพร่หลาย ซึ่งค่าบริการในการติดต่อนั้นจะขึ้นอยู่กับขนาดของข้อมูลและระยะทางของการส่งทำให้เกิดปัญหาว่า ถ้าเราทำการส่งข้อมูลที่มีขนาดใหญ่ไปยังเครื่องโทรสารปลายทางที่อยู่ไกล จะทำให้เสียค่าใช้จ่ายเป็นจำนวนมาก

ดังนั้นโครงงานนี้จึงจัดทำขึ้นเพื่อเป็นการแก้ปัญหาดังที่กล่าวข้างต้น โดยการใช้ประโยชน์ของระบบเครือข่ายภายในองค์กรและมีการเชื่อมต่อคอมพิวเตอร์ทั้งองค์กรสู่ระบบอินเทอร์เน็ตผ่านเครื่องคอมพิวเตอร์เพียงเครื่องเดียว โดยการเขียนแอพพลิเคชั่นขึ้นซึ่งมุ่งเน้นการใช้ประโยชน์ในองค์กรที่ต้องมีการจัดส่งเอกสารผ่านเครื่องโทรสารไปยังหน่วยงานย่อยที่อยู่ไกลออกไปอยู่เป็นประจำ

แอพพลิเคชั่นที่สร้างขึ้นนี้มีโปรแกรมสำหรับรับข้อมูลที่ต้องการส่งจากเครื่องโทรสารมาเก็บเป็นแฟ้มข้อมูลที่เครื่องให้บริการที่หน่วยงานผู้ส่งเอกสาร ซึ่งเครื่องให้บริการต้องต่ออยู่กับเครือข่ายอินเทอร์เน็ต แล้วทำการส่งข้อมูลนั้นผ่านเครือข่ายอินเทอร์เน็ตไปยังเครื่องให้บริการที่ปลายทาง จากนั้นจึงส่งต่อไปยังเครื่องโทรสารปลายทางได้ นอกจากนี้แอพพลิเคชั่นดังกล่าวยังได้รองรับการให้บริการแก่ยูสเซอร์ต่างๆ ในการรับส่งไฟล์

ข้อมูล ที่สามารถส่งไฟล์ที่มีรหัสภาษาไทย (รหัส สมอ.) จากโปรแกรมราชวิถี (RW) และโปรแกรมเวิร์ดจุฬาฯ (CW) รวมทั้งไฟล์ข้อความจากระบบปฏิบัติการดอส (DOS) โดยจะมีโปรแกรมที่ทำหน้าที่ในการเข้ารหัสเพื่อให้มีรูปแบบที่เหมาะสมสำหรับการส่งออกไปยังเครื่องโทรสาร ดังนั้นเทอร์มินัลปลายทางที่ต้องการ จึงสามารถเป็นได้ทั้งยูสเซอร์หรือเครื่องโทรสารอื่นๆ

บทที่ 2
ทฤษฎีหรือหลักการ

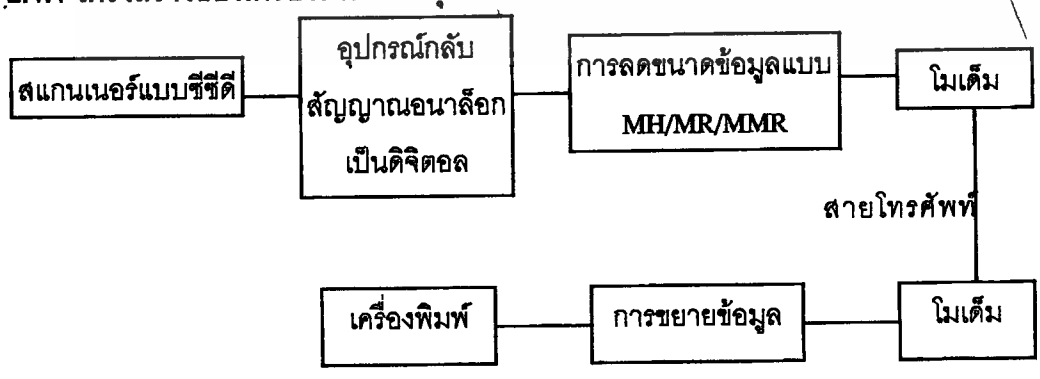
2.1 คุณสมบัติของเครื่องโทรสารกลุ่ม 3

คุณสมบัติของเครื่องโทรสารกลุ่ม 3 ตามมาตรฐานของ CCITT แสดงดังต่อไปนี้

Item	Standard	Options		Small Copy (A5&A6)			
Scan Width							
in	8.46	10	11.9	4.2	5.9	5.9	4.2
mm	215	255	303	107	151	151	107
Pels/line	1728	2048	2432	854	1216	1728	1728
H /in	203			203	203	290	406
/mm	8			8	8	11.4	16
v /in	97.8	196		196/392	138/176	138/176	196/392
/mm	385	7.7		7.7/15.4	5.44/10.9	5.44/10.9	7.7/15.4
ms/line	20	0,5,10,40					
Coding	Modified	Modified Read, Modified-Modified Read					
Modem							
Fax Signal	V.27ter	V.29		V.17			
Bits/s	2400/4800	9600/7200		14400, 12000, 9600, 7200			
Handshake	V.21 (Ch2)	V.27ter					
Bits/s	300	2400					
Error	None	Error Correction Mode					

ตารางที่ 2.1 คุณสมบัติของเครื่องโทรสารกลุ่ม 3 ตามมาตรฐานของ CCITT

2.1.1 โครงสร้างของเครื่องโทรสารกลุ่ม 3



รูปที่ 2.1 โครงสร้างของเครื่องโทรสารกลุ่ม 3

เครื่องโทรสารกลุ่ม 3 มีการทำงานดังต่อไปนี้

1.สแกนเนอร์แบบซีซีดี (CCD:Charge Coupled Device) ทำหน้าที่ในการเปลี่ยนภาพให้เป็นไฟฟ้าโดยอาศัยโฟโต้เซนเซอร์ (photosensors) เป็นตัวตรวจจับความเข้มของจุดแต่ละจุดแล้วสร้างพัลส์ (pulse) ขึ้นมาแทนจุดแต่ละจุดนั้น ซึ่งใน 1 เส้นจะทำการตรวจจับ 1728 จุด ทำให้ได้พัลส์จำนวน 1728 ลูก

2.ตัวแปลงสัญญาณอนาล็อกเป็นดิจิตอล (A/D converter) จะทำการเปลี่ยนสัญญาณอนาล็อกให้เป็นดิจิตอล

3.การลดขนาดข้อมูล อาจเป็นแบบโมดิฟายด์ฮัฟฟ์แมน (HM:Modified Huffman), โมดิฟายดรีดดิ้ง (MR:Modified Reading) หรือโมดิฟายด์-โมดิฟายดรีดดิ้ง (MMR:Modified Modified Reading) ซึ่งจะทำการลดจำนวนบิทของข้อมูลให้น้อยลง โดยการใช้รหัสตัวใหม่แทนข้อมูลเดิมทำให้ข้อมูลมีขนาดเล็กลงกว่าเดิม ช่วยให้สามารถส่งข้อมูลได้เร็วขึ้นด้วย

4.โมเด็ม (Modem) จะทำการเปลี่ยนสัญญาณที่ถูกเข้ารหัสเป็นดิจิตอลเรียบร้อยแล้ว ให้เป็นสัญญาณอนาล็อกอีกครั้งหนึ่ง เพื่อให้สามารถส่งไปบนสายโทรศัพท์

ส่วนทางด้านรับจะมีโมเด็มทำการเปลี่ยนสัญญาณอนาล็อกที่รับมาจากทางด้านส่ง ให้เป็นสัญญาณดิจิตอลแล้วส่งไปยังส่วนขยายข้อมูล ซึ่งจะทำการเปลี่ยนรหัสต่างๆ ให้อยู่ในรูปของจุดขาว,ดำ เพื่อให้เครื่องพิมพ์สามารถแสดงผลออกมาได้เหมือนต้นฉบับเดิม

2.1.2 ขบวนการรับส่งข้อมูลของเครื่องโทรสารกลุ่ม 3

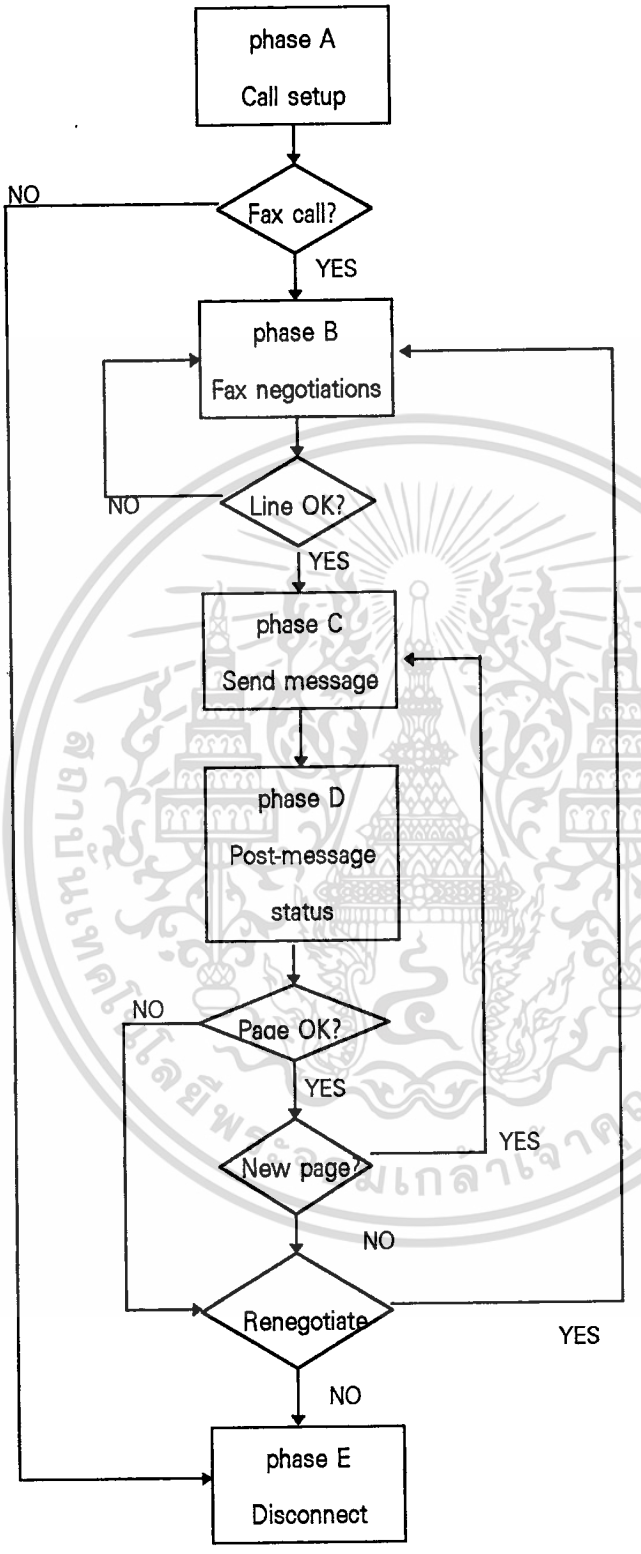
ตามมาตรฐานของ CCITT T.30 ได้แบ่งขั้นตอนในการรับส่งข้อมูลของเครื่องโทรสารกลุ่ม 3 ออกเป็น 5 เฟส แสดงได้ดังรูปที่ 2.2

ช่วงการทำงานตั้งแต่เฟส A-E เรียกว่า แฟกซิไมล์เซชชั่น (Facsimile session)

ช่วงการทำงานตั้งแต่เฟส B-D เรียกว่า แฟกซ์ไมล์โพรซีเดอร์ (Facsimile prodcedure)

<u>เฟส A ขั้นตอนการติดต่อ</u> (Call Establishment) หรือเรียกว่าการเริ่มต้นการเรียก เป็นขั้นตอนที่จะเชื่อมโยงเครื่องโทรสารต้นทางเข้ากับเครื่องโทรสารปลายทาง โดยผ่านทางข่ายสายโทรศัพท์ อาจเป็นแบบควบคุมโดยโอเปอเรเตอร์ ให้โอเปอเรเตอร์ติดต่อกันได้แล้วจึงทำการส่งข้อมูล หรืออาจเป็นแบบอัตโนมัติคือรับหรือส่งเอกสารโดยอัตโนมัติที่ด้านใดด้านหนึ่งหรือสามารถรับอัตโนมัติและส่งอัตโนมัติได้ทั้งสองด้าน

ทางด้านผู้เรียกจะทำการเรียกไปยังเครื่องโทรสารปลายทางที่ต้องการติดต่อด้วยสัญญาณไดออลโทน (dial tone) และส่งสัญญาณ CNG (Calling tone) ไปเพื่อเริ่มการติดต่อ ส่วนทางด้านถูกเรียกจะตอบสนองต่อการเรียกนั้น โดยส่งสัญญาณ CED (Called station identification) ซึ่งเป็นสัญญาณความถี่ 2100 Hz กลับไปยังเครื่องโทรสารด้านผู้เรียกทุกๆ 3 วินาที



รูปที่ 2.2 ขบวนการในการรับส่งข้อมูลของเครื่องโทรสารกลุ่ม 3

เฟส B ขั้นตอนการส่งข่าวสาร (Pre-message procedure) เป็นขั้นตอนที่เครื่องโทรสารต้นทางเชื่อมโยงกับเครื่องโทรสารปลายทางแล้วต้องทำการตรวจสอบคุณสมบัติและความสามารถในการรับและส่งให้ตรงกัน ก่อนที่จะเริ่มส่งข่าวสาร มี 2 ขั้นตอนคือ

1.ส่วนการตรวจสอบสัญญาณ ทำการส่งสัญญาณเพื่อบอกให้ทราบว่าเป็นเครื่องโทรสารอยู่ในกลุ่มใด, พร้อมที่จะรับข้อมูลหรือไม่, หมายเลขโทรศัพท์ของเครื่องโทรสารต้นทาง รวมถึงความสามารถอื่นๆ ที่ไม่ใช่มาตรฐานของ CCITT เป็นทางเลือก

2.ส่วนคำสั่ง ส่งสัญญาณเพื่อตรวจสอบระบบสื่อสัญญาณ DIS (Digital identification signal) ไปยังด้านผู้เรียกเพื่อถึงแสดงคุณสมบัติและความสามารถต่างๆ ของเครื่อง ซึ่งเมื่อทางด้านผู้เรียกได้รับสัญญาณนี้แล้วก็จะส่งสัญญาณ DCS (Digital Command Signal) มายังด้านผู้ถูกเรียกเพื่อแจ้งให้ทราบว่าได้เลือกใช้คุณสมบัติและความสามารถใดของเครื่องที่ถูกเรียกเป็นข้อกำหนดในการรับส่งข้อมูลครั้งนี้ ซึ่งเครื่องโทรสารด้านถูกเรียกก็จะต้องเลือกโหมดของตนเองให้เป็นไปตามข้อกำหนดนั้น จากนั้นเครื่องโทรสารด้านผู้เรียกจะส่งสัญญาณความเร็วสูงที่เรียกว่า เทรนนิ่ง (training) เพื่อตรวจสอบระบบสื่อสาร ซึ่งหลังจากเครื่องโทรสารด้านถูกเรียกได้รับสัญญาณเทรนนิ่งและเมื่อตรวจสอบเรียบร้อยแล้ว ก็จะส่งสัญญาณ CFR (Confirmation to Receive) กลับไปยังเครื่องโทรสารต้นทางเพื่อยืนยันความพร้อมในการรับข้อมูล

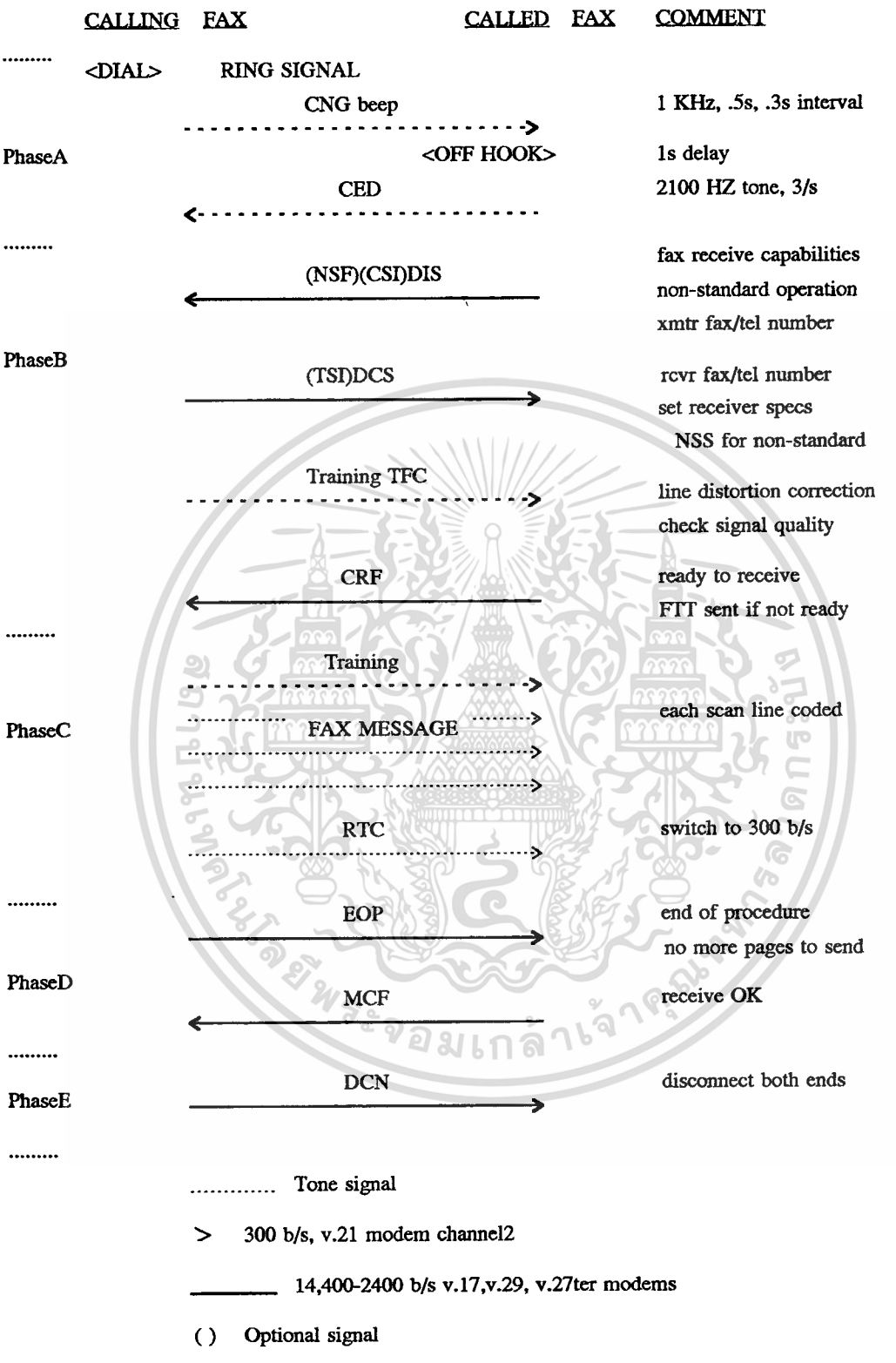
เฟส C ขั้นตอนการส่งข้อมูล (Message transmission) ประกอบด้วย

1.เฟส C1 เป็นขั้นตอนส่งสัญญาณควบคุมเพื่อการซิงโครไนซ์ การตรวจสอบข้อผิดพลาด การแก้ไขข้อผิดพลาด และตรวจสอบสถานะของระบบสื่อสัญญาณในขณะที่กำลังส่งข่าวสาร

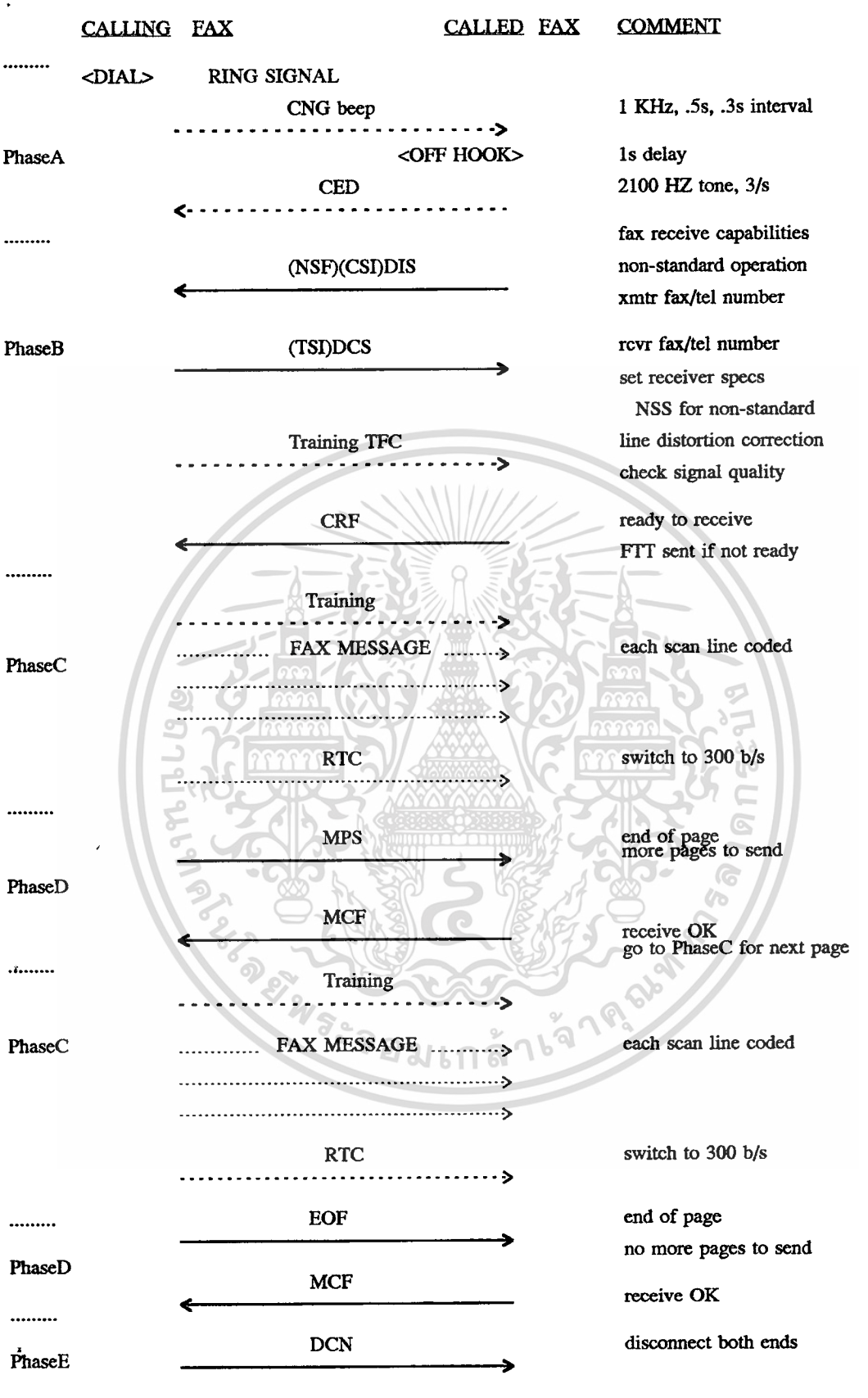
2.เฟส C2 เป็นขั้นตอนการส่งข่าวสาร โดยมีรูปแบบตามแต่ชนิดของเครื่องโทรสารแต่ละกลุ่ม

เฟส D ขั้นตอนหลังการส่งข่าวสาร (Post-message procedure) เครื่องโทรสารด้านผู้เรียกจะส่งสัญญาณ RTC (Return to Control) ไปยังด้านถูกเรียกเพื่อทำการปรับโมเด็มให้กลับไปอยู่ที่ความเร็ว 300 บิท/วินาที จากนั้นจึงจะส่งสัญญาณ EOP (End of Procedure) ซึ่งทางด้านถูกเรียกก็จะส่งสัญญาณ MCF (Message Confirmation) กลับมาให้ทางด้านต้นทางเพื่อยืนยันการรับข้อมูลที่ถูกต้องเรียบร้อย ในกรณีที่มีการส่งข้อมูลหลายๆ หน้า เมื่อจบหน้าแรกจะมีการส่งสัญญาณอื่นๆ ที่แสดงว่ามีข้อมูลที่จะส่งต่อไปอีก แทนการส่งสัญญาณ EOP เช่น EOM (End of Message) หรือ MPS (Multi Page Signal)

เฟส E ปลดสาย (Call release) เป็นขั้นตอนเลิกการติดต่อระหว่างเครื่องโทรสารต้นทาง และเครื่องโทรสารปลายทาง อาจเป็นแบบใช้อัตโนมัติหรือไม่ก็ได้ โดยเครื่องโทรสารด้านผู้เรียกจะส่งสัญญาณ DCN (Disconnect) ไปยังด้านปลายทางเพื่อยกเลิกการติดต่อ



รูปที่ 2.3 แสดงขั้นตอนการรับส่งข้อมูลระหว่างเครื่องโทรสารจำนวน 1 หน้า



รูปที่ 2.4 แสดงขั้นตอนการรับส่งข้อมูลระหว่างเครื่องโทรสารจำนวน มากกว่า 1 หน้า

2.2 แฟกซ์โมเด็ม (Fax Modem)

โมเด็มที่ใช้ในเครื่องโทรสารกลุ่ม 3 ได้กำหนดไว้ตามมาตรฐานของ CCITT ดังตารางต่อไปนี้

Bits per Second	Baud Rate	Bits per Sample	Type	Carrier Frequency	Bandwidth in Hertz
14400	2400	6	V.17	1800	550-3050
12000	2400	5	V.17	1800	550–3050
9600	2400	4	V.29	1700	450-2950
7200	2400	3	V.29	1700	450-2950
4800	1600	3	V.27ter	1800	950-2650
2400	1200	2	V.27ter	1800	1150-2450

ตารางที่ 2.2 มาตรฐานของโมเด็มที่ใช้ในเครื่องโทรสารกลุ่ม 3 ตามข้อกำหนดของ CCITT

โมเด็มที่ใช้ในเครื่องโทรสารกลุ่ม 3 จะตรวจสอบการติดต่อระหว่างเครื่องโทรสารทั้ง 2 ด้านก่อนการส่งข้อมูล และโมเด็มจะพยายามส่งข้อมูลด้วยความเร็วสูงสุดเท่าที่จะเป็นไปได้ ส่วนใหญ่โมเด็มในเครื่องโทรสารได้ถูกกำหนดไว้ตามมาตรฐาน V.27ter และ V.29 โดยส่วนใหญ่ในการส่งข้อมูลของเครื่องโทรสารกลุ่ม 3 นั้น จะเริ่มต้นที่ความเร็ว 9600 บิทต่อวินาที แต่ถ้าหากไม่สามารถส่งข้อมูลด้วยความเร็วนี้ได้ โมเด็มจะลดความเร็วในการส่งลงเป็น 7200,4800 หรือ 2400 บิทต่อวินาที แทนตามความเหมาะสม

2.2.1 การแบ่งระดับโมเด็มโดย EIA

สมาคม EIA ได้พัฒนามาตรฐานซึ่งกำหนดขั้นตอนและคำสั่งที่ใช้ระหว่างคอมพิวเตอร์ (DTE) และโมเด็ม (DCE) ขึ้น โดยแบ่งโมเด็มออกเป็น 3 ระดับตามความสามารถของโมเด็มในการนำไปสู่การส่งโทรสาร ดังนี้

1.แฟกซ์โมเด็มระดับ 1 หมายถึงแฟกซ์โมเด็มที่มีบริการเท่าที่จำเป็นต่อการรองรับขั้นตอนของโทรสารกลุ่ม 3 เท่านั้น การส่งโทรสารที่แฟกซ์โมเด็มระดับ 1 จะต้องใช้ภายใต้การควบคุมของซอฟต์แวร์ ต่างจากการส่งแบบใช้ดาต้าโมเด็ม ซึ่งการกำหนดเวลา การเข้ารหัส และการลำดับการส่งต้องอาศัยมาตรฐาน T.30 ทำให้ไม่สามารถควบคุมการส่งโทรสารโดยใช้คำสั่งจากผู้ควบคุมเครื่องได้

2.แฟกซ์โมเด็มระดับ 2 มีความสามารถสูงกว่าแฟกซ์โมเด็มระดับ 1 ซึ่งขั้นตอนที่ยุ่งยากในการส่งโทรสารโดยใช้มาตรฐาน T.30 จะถูกตัดออกจากคอมพิวเตอร์ไปเป็นหน้าที่ของโมเด็มแทน ซึ่งหน้าที่ของโมเด็มระดับ 2 คือการเริ่มต้นการเรียก การจัดการขบวนการติดต่อสื่อสาร การส่งข้อมูลภาพและหน้าที่อื่นๆ ระหว่างรูปแบบของมาตรฐาน T.4 (เครื่องโทรสารกลุ่ม 3) และมาตรฐาน T.6 (เครื่องโทรสารกลุ่ม 4) เครื่องคอมพิวเตอร์จะทำหน้าที่ในการจัดเตรียมและลดขนาดข้อมูลภาพสำหรับการสื่อสารและตีความข้อมูลที่ถูกลดขนาดมาแล้วในทางด้านรับ

3.แฟกซ์โมเด็มระดับ 3 ยังคงอยู่ในระหว่างการศึกษาเพื่อกำหนดมาตรฐาน

2.2.2 กลุ่มคำสั่งที่ใช้ควบคุมแฟกซ์โมเด็ม

คำสั่งสำหรับควบคุมแฟกซ์โมเด็มระดับ 1 และ 2 โดย ELA ถูกออกแบบเพิ่มเติมจากลุ่มคำสั่ง AT ซึ่งสายของอักขระที่ถูกส่งไปยังโมเด็มจะเรียกว่า แถวคำสั่ง (command line) และจะต้องขึ้นต้นด้วยอักษร AT หรือ at แถวคำสั่งประกอบด้วยสัญลักษณ์ใน รหัสแอสกี้ (ascii) และลงท้ายด้วยเครื่องหมายแสดงการสิ้นสุดคำสั่งที่เรียกว่า รหัสแอสกี้ 13 (carriage return)

มีเพียง บิทต่ำ 7 บิทของแต่ละอักษรเท่านั้นที่จะถูกตรวจสอบในขณะที่โมเด็มแปลความหมายของคำสั่ง คำสั่งที่เขียนด้วยอักษรตัวใหญ่หรือตัวเล็กจะมีความหมายเหมือนกัน ช่องว่างและเครื่องหมายควบคุมอื่นๆ ที่นอกเหนือไปจาก รหัสแอสกี้ 13 และ รหัสแอสกี้ 8 (ช่องว่าง) ที่ปรากฏในคำสั่งจะถูกละเลยไป และจะถูกแสดงค่าด้วย 0 ด้วยเหตุนี้แฟกซ์โมเด็มทุกเครื่องจึงต้องมี การควบคุมการไหล (flow control) แบบ XON/XOFF และอาจมีการควบคุมการไหลอื่นๆ อีกก็ได้

<u>คำสั่งสำหรับแฟกซ์โมเด็มระดับ 1</u>

แฟกซ์ของ EIA แต่ละคำสั่งจะขึ้นต้นด้วย +F ซึ่งมีรูปแบบทั่วไปอยู่ 3 รูปแบบตามการใช้งานได้แก่ ความสามารถ (capabilities), สถานะ (status) และการกำหนด (set)

ถ้าต้องการกำหนดขอบเขตความสามารถของโมเด็ม จะใช้รูปแบบดังนี้

+ F *command=?*

ซึ่ง command จะแสดงคำสั่งแฟกซ์ที่ใช้งานได้ โมเด็มจะตอบสนองโดยการเลือกค่าหรือช่วงของค่าที่มันสามารถรับได้ เช่นโมเด็มระดับ 1 อาจจะตอบสนองต่อแถวคำสั่งซึ่งรูปแบบ AT+FCLASS=? (เป็นการบอกว่าการติดต่อสื่อสารในครั้งนี้ใช้ระดับใด) ด้วยค่า 0,1 ผลตอบสนองนี้แสดงให้เห็นว่าโมเด็มสามารถถูกกำหนดให้เป็นระดับ 1 ได้

คำสั่งชุดที่ 2 คือคำสั่งเกี่ยวกับสถานะ จะเป็นการกำหนดค่าขององค์ประกอบหรือทางเลือกในโครงสร้าง คำสั่งนี้มีรูปแบบดังนี้

+ F *command?*

ตัวอย่างการกำหนดรูปแบบการทำงานเช่น กำหนดสั่งว่า AT+FCLASS? โมเด็มซึ่งทำงานเหมือนเป็นแฟกซ์โมเด็มระดับ 1 จะตอบสนองต่อคำสั่งด้วย 1

คำสั่งชุดสุดท้ายใช้สำหรับกำหนดค่าขององค์ประกอบหรือผ่านค่าองค์ประกอบที่ใช้ควบคุมการทำงานของโมเด็ม มีรูปแบบดังนี้

+ F *command=val*

ซึ่ง command จะแสดงถึงองค์ประกอบที่ต้องการกำหนดและ val แสดงถึงค่าขององค์ประกอบที่ต้องการ ทั้งนี้ขึ้นอยู่กับแต่ละคำสั่ง val อาจจะเป็นตัวเลขหรือตัวอักษรก็ได้ ซึ่งการกำหนดเป็นตัวเลขทั้งในคำสั่งมาตรฐาน AT ดาต้าโมเด็ม (data modem) และในคำสั่งแฟกซ์โมเด็มระดับ 1 จะแสดงด้วยเลขฐานสิบ

ค่าเริ่มต้นของแถวคำสั่งตัวสุดท้ายคือ รหัสแอสกี้ 13 (carriage return) หรือแถวคำสั่งอาจจะจบด้วยเครื่องหมาย ; ก็ได้ ยกเว้นคำสั่ง +FTS และ +FRS คำสั่งของโมเด็มระดับ 1 จะต้องเป็นคำสั่งสุดท้ายในบรรทัด

(แม้ว่าจะได้กำหนดไว้อย่างชัดเจนใน EIA/TIA-578 แต่แฟกซ์โมเด็มระดับ 1 จำนวนมากก็ยอมรับคำสั่งหลายๆ คำสั่งในบรรทัดเดียวกันโดยไม่มีการแบ่งแยกเป็นส่วน)

แฟกซ์โมเด็มอาจจะถูกกำหนดให้ตอบสนองต่อตัวอักษรหรือตัวเลขอย่างใดอย่างหนึ่งก็ได้ การตอบสนองต่อตัวอักษรจะตามด้วย รหัสแอสกี้ 13 (carriage return) และเครื่องหมายแสดงการขึ้นบรรทัดใหม่ ส่วนการตอบสนองต่อตัวเลขจะตามด้วย รหัสแอสกี้ 13 (carriage return) เท่านั้น และมีรหัสที่ใช้แสดงผลดังนี้คือ ตกลง(0), ติดต่อ(1), ไม่มีสัญญาณ(3), ผิดพลาด(4)

<u>คำสั่งสำหรับแฟกซ์โมเด็มระดับ 2</u>

คำสั่ง EIA ระดับ 2 นี้มีความจำเป็นต้องใช้รูปแบบตามข้อกำหนดเช่นเดียวกับระดับ 1 และจะขึ้นต้นคำสั่งด้วย +F เสมอ รูปแบบของคำสั่งมี 3 แบบเช่นเดียวกับระดับ 1 ดังกล่าวข้างต้น สำหรับระดับ 2 คำสั่งที่ใช้สำหรับกำหนดค่าขององค์ประกอบหรือผ่านองค์ประกอบที่ใช้ในการควบคุมการทำงานของโมเด็มอาจจะใช้ได้กับค่าเดียวหรือหลายค่าก็ได้ ซึ่งต่างจากระดับ 1 อย่างไรก็ตาม ในคำสั่งระดับ 2 ค่าคงที่จะต้องเป็นเลขฐาน 16 ค่าคงที่ต่างๆ นี้สร้างจากขึ้นมาจาก 0 ถึง 9 (รหัสแอสกี้ 30h ถึง 39h) และ A ถึง F (รหัสแอสกี้ 41h ถึง 46h) เช่นค่าคงที่ 255 ในเลขฐานสิบมีค่าเท่ากับ FFh ในฐาน 16 จะถูกส่งตัวอักษร 2 ตัวคือ FF (หมายเหตุ:อักษร h ซึ่งแสดงให้ทราบว่าเป็นเลขฐาน 16 จะไม่ถูกส่งออกไปด้วย)

กลุ่มของค่าคงที่ซึ่งประกอบด้วยสัญลักษณ์ในแอสกี้ จะต้องอยู่ระหว่างเครื่องหมาย “และ” ส่วนสตริงว่าง (null-string) จะมีเครื่องหมาย “”

สำหรับการกำหนดค่าที่มีหลายๆ ค่า แต่ละค่าจะถูกแยกด้วยเครื่องหมาย “,” เช่นผลตอบสนองคือ (0,2,4,8) ส่วนการกำหนดค่าเป็นช่วงจะแสดงด้วยเครื่องหมาย “-” เช่น กำหนดค่าอยู่ในช่วง 0.255 (ฐานสิบ) สามารถเขียนได้ว่า 0-FF เป็นต้น และคำสั่งในระดับ 2 ยังยอมรับการกำหนดค่าหลายแบบผสมกัน ประกอบด้วยค่าที่เป็นลำดับซึ่งแต่ละค่าจะอยู่ในวงเล็บและแบ่งด้วย “,” โดยจะไม่สนใจช่องว่างระหว่างค่าเช่น (0,1,2), (),(0-3)

คำสั่งระดับ 2 จะถูกทำจากซ้ายไปขวาและแต่ละคำสั่งจะถูกทำแยกจากกัน โดยไม่สนใจสิ่งที่ตามหลังแถวคำสั่ง ถ้าทุกคำสั่งสามารถทำได้ถูกต้อง จะแสดงด้วยรหัสที่บอกสถานะของคำสั่งสุดท้ายในบรรทัด แต่ถ้าเกิดความผิดพลาดขึ้นหรือพบคำสั่งที่ไม่ถูกต้องตามข้อกำหนด การประมวลผลคำสั่งจะหยุดลงและคำสั่งอื่นๆ ที่ยังไม่ได้ทำจะยกเลิกไป รหัสแสดงผลของแฟกซ์โมเด็มระดับ 2 มีดังนี้

0 : OK

1 : CONNECT

2 : RING

3 : NO CARRIER

4 : ERROR

5 : NO DIALTONE

6 : BUSY

7 : NO ANSWER

2.3 การเข้ารหัส

ในเครื่องโทรสารกลุ่ม 3 กำหนดให้มีการเข้ารหัสเพื่อให้ข้อมูลที่ต้องส่งผ่านระบบสื่อสัญญาณมีจำนวนน้อยลง ช่วยให้การส่งเร็วขึ้น การเข้ารหัสมีให้เลือก 2 แบบคือ 1 มิติ, 2 มิติ แต่จะแสดงเฉพาะรายละเอียดของการเข้ารหัสแบบ 1 มิติ ซึ่งใช้หลักการเข้ารหัสแบบโมดิฟายด์ฮัฟฟ์แมน ซึ่งข้อมูลที่มีค่าความน่าจะเป็นสูง (เกิดขึ้นบ่อย) จะถูกแทนด้วยรหัสที่มีความยาวน้อย ส่วนข้อมูลที่มีความน่าจะเป็นต่ำ (เกิดขึ้นน้อย) จะถูกแทนด้วยรหัสที่มีความยาวมาก ซึ่งในมาตรฐานของ CCITT ได้กำหนดรหัสที่ใช้ในการแทนมาอย่างแน่นอนแล้ว ดังในตารางที่ 2.3 และ 2.4

ข้อกำหนดในการเข้ารหัสดังต่อไปนี้

1.การเข้ารหัสจะเริ่มขึ้นด้วยจุดขาวก่อนเสมอ กรณีที่จุดแรกเป็นจุดดำ รหัสของจุดขาวที่มีความยาวเท่ากับ 0 จะถูกส่งออกไป

2.ข้อมูลจุดขาวหรือจุดดำสามารถมีความยาวได้ถึง 1728 พิคเซล ซึ่งเป็นความยาวสูงสุดสำหรับเส้นสแกนมาตรฐาน 1 เส้น

3.รหัสที่ใช้แทนข้อมูลมี 2 ชนิดคือ

- เทอร์มิเนติ้งโค้ด (Terminating Code) ใช้แทนค่าพิคเซลที่มีความยาวตั้งแต่ 0 ถึง 63
- เมคอัพโค้ด (Make-up Code) ใช้แทนค่าพิคเซลที่มีความยาวเป็นจำนวนเท่าของ 64 จนถึง 1728 นั่นคือมีเมคอัพโค้ดแทนรหัสได้ 27 ชุด

4.กรณีของพิคเซลซึ่งมีความยาวตั้งแต่ 64-1728 จะแทนด้วยเมคอัพโค้ดซึ่งมีค่าเท่ากับหรือน้อยกว่าความยาวของพิคเซลก่อน โดยความยาวของพิคเซลที่เหลือจากการแทนด้วยเมคอัพโค้ด จะมีค่าไม่เกิน 63 พิคเซล ซึ่งสามารถแทนค่าที่เหลือด้วยเทอร์มิเนติ้งโค้ดได้

5.จุดสิ้นสุดของเส้นสแกน (EOL:End-Of-Line) เมื่อสิ้นสุดการสแกนข้อมูลครบ 1 เส้น ข้อมูลจะต้องตามด้วยรหัส EOL เพื่อแสดงจุดสิ้นสุดของเส้นสแกน นอกจากนี้รหัส EOL ยังใช้นำหน้าเส้นสแกนเส้นแรกของแต่ละหน้าด้วย

รูปแบบของ EOL คือ 0000 0000 0001

6.บิทเติม (Fill) เป็นบิทที่เติมขึ้นระหว่างบิทของข้อมูลกับบิทสิ้นสุด เพื่อให้เวลาในการส่งข้อมูลในเส้นสแกนนั้นไม่ต่ำกว่าเวลาต่ำสุดที่เครื่องพิมพ์ด้านรับสามารถทำงานได้ทัน

รูปแบบของบิทเติมคือ 0 ซึ่งจำนวนบิทสามารถแปรค่าได้

7.เมื่อจบการส่งข้อมูลแต่ละแผ่นจะต้องตามด้วยรหัสรีเทิร์นทูคอนโทรล (RTC:Return to Control) เพื่อเป็นการกลับเข้าสู่โหมดการควบคุมเครื่องโทรสาร

รูปแบบของรหัส RTC คือ รหัส EOL ติดต่อกันจำนวน 6 ชุด

run	white	black	run	white	black
0	00110101	0000110111	32	00011011	000001101010
1	000111	010	33	00010010	000001101011
2	0111	11	34	00010011	000011010010
3	10000	10	35	00010100	000011010011
4	1011	011	36	00010101	000011010100
5	1100	0011	37	00010110	000011010101
6	1110	0010	38	00010111	000011010110
7	1111	00011	29	00101000	000011010111
8	10011	000101	40	00101001	000001101101
9	10100	000100	41	00101010	000001101101
10	00111	0000100	42	00101011	000011011010
11	01000	0000101	43	00101100	000011011011
12	001000	0000111	44	00101101	000001010100
13	000011	00000100	45	00000100	000001010101
14	110100	00000111	46	00000101	000001010110
15	110101	000011000	47	00001010	000001010111
16	101010	0000010111	48	00001011	000001100100
17	101011	0000011000	49	01010010	000001100101
18	0100111	0000011000	50	010101011	000001010010
19	0001100	00001100111	51	01010011	000001010011
20	0001000	00001101000	52	01010101	000000100100
21	0010111	00001101100	53	00100100	000000110111
22	0000011	00000110111	54	00100101	000000111000
23	0000100	00000101000	55	01011000	000000100111
24	0101000	00000010111	56	01011001	000000101000
25	0101011	00000011000	57	01011010	000001011000
26	0010011	000011001010	58	01011011	000001011001
27	0100100	000011001011	59	01001010	000000101011
28	0011000	000011001100	60	01001011	000000101100
29	00000010	000011001101	61	00110010	000001011010
30	00000011	000001101000	62	00110011	000001100110
31	00011010	000001101001	63	00110100	000001100111

ตารางที่ 2.3 เทอร์มิเนติ้งโค้ด

run	white	black	run	white	black
64	11011	0000001111	960	011010100	0000001110011
128	10010	000011001000	1024	011010101	0000001110100
192	010111	000011001001	1088	011010110	000000111010
256	010111	000001011011	1152	011010111	0000001110110
320	0110111	000000110011	1216	011011000	0000001110111
384	00110110	000000110100	1280	011011001	0000001010010
448	01100100	000000110101	1344	011011010	0000001010011
512	01100101	0000001101100	1408	011011011	0000001010100
576	01101000	0000001101101	1472	010011000	0000001010101
640	01100111	0000001101010	1536	01001101	0000001011010
704	011001100	0000001001011	1600	010011010	0000001011011
768	011001101	0000001001100	1664	011000	0000001100100
832	011010010	0000001001101	1728	010011011	0000001100101
896	011010011	0000001110010			

ตารางที่ 2.4 เมคอัพโค้ด

2.4 ระบบเครือข่าย และการโปรแกรม

การพัฒนาโปรแกรมประยุกต์บนระบบเครือข่ายจำเป็นต้องมีความรู้พื้นฐานทางด้านระบบเครือข่าย พอสมควรในหัวข้อนี้จะอธิบายคำศัพท์พื้นฐาน และหลักการของระบบเครือข่ายที่จำเป็นสำหรับการทำความเข้าใจในการพัฒนาโปรแกรมประยุกต์บนระบบเครือข่ายดังจะกล่าวต่อไป

2.4.1 ความหมายของระบบเครือข่ายคอมพิวเตอร์ และอินเทอร์เน็ตเวิร์คกิ้ง (Internet working)

ระบบเครือข่ายคอมพิวเตอร์ (Computer Network) คือระบบการเชื่อมต่อระหว่างระบบปลายทาง (End-System) ซึ่งระบบปลายทางเป็นเป็นระบบอิสระจากกัน (Autonomous) ระบบปลายทางสามารถเป็นได้ตั้งแต่ไมโครคอมพิวเตอร์ (Microcomputer) ไปจนกระทั่งซุปเปอร์คอมพิวเตอร์ (Supercomputer) ขนาดใหญ่ เพื่อจุดมุ่งหมายในการแลกเปลี่ยนข้อมูล และการแบ่งข้อมูลในการแบ่งปันทรัพยากรของระบบเช่น ไฟล์ (File) ข้อมูล, เครื่องพิมพ์ (Printer), โมเด็ม (Modem), ตลอดจนการให้บริการฐานข้อมูลร่วม (Sharing database)

อินเทอร์เน็ตเวิร์คกิ้ง หรืออินเทอร์เน็ต (Internet) คือการเชื่อมต่อของระบบเครือข่าย 2 เครือข่ายขึ้นไป ดังนั้นคอมพิวเตอร์บนระบบเครือข่ายหนึ่งก็สามารถติดต่อกับคอมพิวเตอร์บนระบบเครือข่ายอื่นๆ ได้

2.4.2 องค์ประกอบของอินเทอร์เน็ต

อินเทอร์เน็ตเป็นเครือข่ายคอมพิวเตอร์ชนิดหนึ่งที่ใช้โปรโตคอล TCP/IP(Transmission Control Protocol/Internet Protocol) เป็นมาตรฐานการทำงานของระบบ ดังนั้นถ้ามีเครือข่ายคอมพิวเตอร์ที่ใช้โปรโตคอล TCP/IP อยู่แล้วก็จะเป็นการสะดวกและง่ายต่อการเชื่อมต่อเข้ากับระบบอินเทอร์เน็ต ระบบการทำงานของเครือข่ายโปรโตคอล TCP/IP โดยเฉพาะสำหรับเครือข่ายอินเทอร์เน็ตนั้นจะแบ่งกลุ่มของแฟกเตอร์หรือฟังก์ชั่นการทำงานออกเป็น 6 กลุ่มใหญ่ๆ ซึ่งการติดตั้งอุปกรณ์ต่างๆ เองก็ต้องคำนึงถึงแฟกเตอร์ 6 กลุ่มเช่นเดียวกันคือ

1.ชนิดของสถานี

ระบบเครือข่ายโปรโตคอล TCP/IP ส่วนใหญ่จะประกอบด้วยเครื่องคอมพิวเตอร์ (สถานี) ที่ทำหน้าที่แตกต่างกันอยู่ 2 ชนิด คือเครื่องที่เป็นสถานีที่ให้บริการที่เรียกว่าโฮสต์ (Host) หรือเซอร์ฟเวอร์ (Server) และเครื่องคอมพิวเตอร์สำหรับผู้ใช้ทั่วไปซึ่งเรียกว่าเทอร์มินัล (Terminal) หรือไคลเอ็นต์ (Client) โดยที่เครื่องคอมพิวเตอร์ที่เป็นสถานีให้บริการนั้นจะเป็นเครื่องที่คอยให้บริการแก่ผู้ใช้ในด้านต่างๆ ไม่ว่าจะเป็นแหล่งเก็บรวบรวมข้อมูล (Data Sharing) การให้บริการโปรแกรมประยุกต์ต่างๆ (Application) หรือการให้บริการการใช้งานระบบประมวลผลกลาง (CPU Time Sharing) เป็นต้น ดังนั้นคุณสมบัติโดยทั่วไปทั้งด้านฮาร์ดแวร์และด้านซอฟต์แวร์ของเครื่องโฮสต์หรือเซอร์ฟเวอร์จึงมีคุณสมบัติที่ดีกว่าเครื่องคอมพิวเตอร์?สำหรับผู้ใช้มาก

2.ระบบไอพีแอดเดรส (IP Address)

การสื่อสารข้อมูลในระบบเครือข่ายคอมพิวเตอร์ เป็นการสื่อสารในลักษณะที่เฟรมข้อมูลของแต่ละการสื่อสารเป็นคนกำหนดเส้นทางที่สื่อสารเอง คือเมื่อมีการขอติดต่อสื่อสารข้อมูลของเครื่องคอมพิวเตอร์คู่ใดเกิดขึ้น เครื่องคอมพิวเตอร์เครื่องนั้นก็จะทำการสร้างเฟรมข้อมูลขึ้นมาแล้วค่อยส่งออกไปในระบบเครือข่าย โดยที่เฟรมข้อมูลนั้นจะมีส่วนของแอดเดรสที่อยู่ในส่วนอ้างอิงกำกับการสื่อสาร (Header) ที่จะบอกว่า เฟรมข้อมูลนี้เป็นของเครื่องคอมพิวเตอร์เครื่องใดที่กำลังส่ง และจะส่งไปยังเครื่องใด

ดังนั้นการที่เครื่องคอมพิวเตอร์ต่างๆ จะติดต่อสื่อสารกันในระบบเครือข่ายโปรโตคอล TCP/IP จะต้องมีการกำหนดค่าไอพีแอดเดรสให้แต่ละสถานีที่จะสื่อสารกันด้วย (นอกเหนือจากค่า MAC-Address ที่มีอยู่ในแต่ละเครื่อง) เพราะค่าไอพีแอดเดรสนั้นจะเป็นค่าอ้างอิงในเฟรมข้อมูลที่สื่อสารในเครือข่าย ซึ่งจะมี 2 ชนิด คือ ไอพีแอดเดรสต้นทาง (Source IP Address) และไอพีแอดเดรสปลายทาง (Destination IP Address)

3.ระบบโปรโตคอลหาเส้นทาง (IP Routing Protocals)

ลักษณะการติดต่อสื่อสารกันระหว่างเครื่องคอมพิวเตอร์ใดๆ ในระบบเครือข่ายอินเตอร์เทนั้นโดยทั่วไปแล้วมี 2 ลักษณะคือ

-การเชื่อมต่อภายในเครือข่ายท้องถิ่น (LAN)

-การเชื่อมต่อระหว่างเครือข่ายท้องถิ่นหนึ่งกับอีกเครือข่ายท้องถิ่นหนึ่ง โดยอาจมีการบริการของระบบเครือข่ายระยะไกล (WAN) เข้ามาเกี่ยวข้องด้วย

ในการเชื่อมต่อจำเป็นต้องใช้อุปกรณ์ที่เรียกว่าเราเตอร์ (Router) โดยเราเตอร์จะเกี่ยวข้องกับระบบทำงานที่เรียกว่า โปรโตคอลหาเส้นทาง (Routing Protocal) ซึ่งจะทำหน้าที่ตรวจสอบและจัดการเกี่ยวกับเส้นทางในการสื่อสารข้อมูลทั้งหมดของระบบ

4.ระบบชื่อกลุ่ม (Domain Name System)

มีการออกแบบระบบชื่อของสถานีบริการต่างๆบนเครือข่ายอินเทอร์เน็ตในรูปลักษณะตัวอักษร เพื่ออำนวยความสะดวกต่อการใช้งานของยูสเซอร์ ระบบ DNS (Domain Name System) เป็นระบบซอฟแวร์ที่ทำหน้าที่ในการจัดสรรและบริการในส่วนการเปรียบเทียบค่าระหว่างชื่อตัวอักษรกับค่าไอพีแอดเดรสของเครื่องสถานีต่างๆ บนอินเทอร์เน็ต

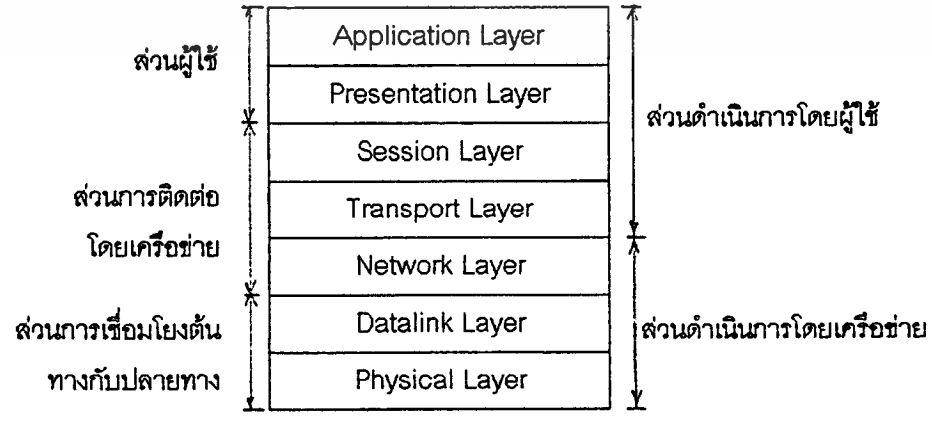
5.โปรแกรมประยุกต์บนเครือข่ายอินเทอร์เน็ต (Application)

ระบบเครือข่ายอินเทอร์เน็ตเป็นระบบเครือข่ายขนาดใหญ่ที่มีการเชื่อมโยงกันทั่วโลก ดังนั้นการใช้งานโปรแกรมประยุกต์ต่างๆ บนระบบอินเทอร์เน็ตจึงมีลักษณะพิเศษแตกต่างจากการใช้งานบนระบบเครือข่ายท้องถิ่นทั่วไป คือจะมีโปรแกรมประยุกต์มากมายหลายชนิด เช่น ระบบจดหมายอิเล็กโทรนิกส์ (E-Mail :Electronic Mail) ระบบข่าวสารร่วม (Usenet) ระบบอุโมงค์อินเทอร์เน็ต (Gopher) และระบบเครือข่ายใยแมงมุม (World-Wide-Web) เป็นต้น โดยที่แต่ละชนิดมีลักษณะการใช้งานที่แตกต่างกันมาก

6.ระบบความปลอดภัย (Security) มีหน้าที่ป้องกันไม่ให้มีการลักลอบเข้ามาใช้หรือทำลายข้อมูลที่สำคัญ

2.5 โมเดลอ้างอิง OSI

เพื่อลดปัญหาในความยุ่งยากสับสนในการจัดการติดต่อสื่อสารข้อมูล โครงสร้างของการสื่อสารข้อมูลภายในอุปกรณ์คอมพิวเตอร์ส่วนใหญ่จะถูกแบ่งเป็นชั้นๆ (Level) เรียกว่าซึ่งแต่ละชั้นจะมีขบวนการลำดับการทำงานของตนเอง และมีหมายเลขลำดับ (N) ชื่อ รายละเอียด และหน้าที่การทำงานแตกต่างกันออกไป



รูปที่ 2.5 OSI โมเดล

OSI เป็นคำย่อที่มาจากคำว่า Open System Interconnection โดยที่เป็นมาตรฐานที่ถูกเสนอขึ้นโดย International Standard Organization ซึ่งเป็นองค์กรที่จัดตั้งขึ้นมาเพื่อดูแลและส่งเสริมตลอดจนกำหนดมาตรฐานของการติดต่อสื่อสารของระบบเครือข่ายคอมพิวเตอร์ โดยโมเดล OSI นี้มีลักษณะเป็นสถาปัตยกรรมแบบระบบเปิด (Open System) เพราะมุ่งที่จะให้ระบบคอมพิวเตอร์ในหลายๆ รูปแบบที่แตกต่างกันสามารถเชื่อมต่อกันได้ OSI โมเดลได้แบ่งโปรโตคอล (protocal) ในการสื่อสารออกเป็น 7 เลเยอร์ (layer) ซึ่งโปรโตคอล คือชุดหรือข้อตกลงในการติดต่อ ข้อสังเกตโมเดล OSI เป็นเพียงข้อเสนอแนะ มิใช่ข้อกำหนด และควรรู้ว่ายังไงไม่มีระบบการเชื่อมต่อที่สร้างเหมือนกับโมเดล OSI จริง

1.ชั้นฟิสิคัล (Physical) เป็นชั้นล่างสุดของการติดต่อสื่อสาร ทำหน้าที่ส่ง-รับข้อมูลจริงๆ จากช่องทางการสื่อสาร (สื่อกลาง) ระหว่างคอมพิวเตอร์เครื่องหนึ่งกับคอมพิวเตอร์เครื่องอื่นๆ มาตรฐานสำหรับเลเยอร์ชั้นนี้จะกำหนดว่าแต่ละคอนเนคเตอร์ (Connector) เช่น RS-232-C มีกี่พิน (PIN) แต่ละพินทำหน้าที่อะไรบ้าง ใช้สัญญาณไฟกี่โวลต์ เทคนิคการมัลติเพลกซ์แบบต่างๆ ก็จะถูกกำหนดอยู่ในชั้นเลเยอร์นี้

2.ชั้นดาต้าลิ้งค์ (Data Link) จะเป็นเสมือนผู้ตรวจสอบ หรือควบคุมความผิดพลาดในข้อมูลโดยจะแบ่งข้อมูลที่ส่งออกเป็นแพ็กเกจหรือเฟรม ถ้าผู้รับได้รับข้อมูลถูกต้องก็จะส่งสัญญาณยืนยันกลับมาว่าได้รับข้อมูลแล้ว เรียกว่า สัญญาณ ACK (Acknowledge) ให้กับผู้ส่ง แต่ถ้าผู้ส่งไม่ได้รับสัญญาณ ACK หรือได้รับสัญญาณ NAK (Negative Acknowledge) กลับมา ผู้ส่งอาจจะทำการส่งข้อมูลไปให้ใหม่ อีกหน้าที่หนึ่งของชั้นนี้ก็คือป้องกันไม่ให้เครื่องส่งทำการส่งข้อมูลเร็วจนเกินขีดความสามารถของเครื่องผู้รับข้อมูลได้

3.ชั้นเน็ตเวอร์ค (Network) เป็นชั้นที่ออกแบบหรือกำหนดเส้นทางการเดินทางข้อมูลที่ส่ง-รับในการส่งผ่านข้อมูลระหว่างต้นทางและปลายทาง ซึ่งเป็นที่แน่นอนว่าในการสื่อสารข้อมูลผ่านเครือข่ายการสื่อสารจะต้องมีเส้นทางการรับ-ส่งข้อมูลมากกว่า 1 เส้นทาง ดังนั้นในชั้นนี้จะมีหน้าที่เลือกเส้นทางที่ใช้เวลาในการสื่อสารน้อยที่สุดและระยะทางสั้นที่สุดด้วย ข่าวสารที่รับมาจากเลเยอร์ชั้นที่ 4 จะถูกแบ่งออกเป็นแพ็กเกจ (packet) ในชั้นที่ 3 นี้

4.ชั้นทรานสปอร์ต (Transport) บางครั้งเรียกว่าชั้น Host-to-Host หรือเครื่องต่อเครื่องและจากเลเยอร์ชั้นที่ 4 ถึงชั้นที่ 7 นี้รวมเรียกว่า เลเยอร์ End-to End ในเลเยอร์ชั้น Transprot นี้จะเป็นการสื่อสารระหว่างต้นทางและปลายทาง (คอมพิวเตอร์กับคอมพิวเตอร์) กันจริงๆ เลเยอร์ชั้น Transport จะทำหน้าที่ตรวจสอบว่าข้อมูลที่ส่งมาจากเลเยอร์ชั้นที่ 5 นั้นไปถึงปลายทางจริงๆ หรือไม่ ดังนั้นการกำหนดตำแหน่งของข้อมูล (Address) จึงเป็นเรื่องสำคัญในชั้นนี้ เนื่องจากจะต้องรู้ว่าใครคือผู้ส่งและใครคือผู้รับข้อมูลนั้น

5.ชั้นเซสชั่น (Session) ทำหน้าที่เชื่อมโยงระหว่างผู้ใช้คอมพิวเตอร์เครื่องอื่นๆ โดยผู้ใช้จะใช้คำสั่งหรือข้อความที่กำหนดไว้ป้อนเข้าไปในระบบ ในการสร้างการเชื่อมโยงนี้ผู้ใช้จะต้องกำหนดรหัสตำแหน่งของจุดหมายปลายทางที่ต้องการติดต่อสื่อสารด้วย ในบางครั้งเครือข่ายอาจมีชั้นเซสชั่นและชั้นทรานสปอร์ตรวมเป็นชั้นเลเยอร์ชั้นเดียวกัน

6.ชั้นพรีเซนเตชั่น (Presentation) ทำหน้าที่เหมือนบรรณารักษ์ กล่าวคือคอยรวบรวมข้อความ(Text) และแปลงรหัสหรือแปลงรูปแบบของข้อมูลให้เป็นรูปแบบการสื่อสารเดียวกัน เพื่อช่วยลดปัญหาต่างๆ ที่อาจจะเกิดขึ้นกับผู้ใช้งานในระบบ

7.ชั้นแอพพลิเคชั่น (Application) เป็นรูปเลเยอร์ชั้นบนสุดของรูปแบบ OSI ซึ่งเป็นชั้นที่ใช้ติดต่อระหว่างผู้ใช้โดยตรงซึ่งได้แก่ โฮสต์คอมพิวเตอร์ เทอร์มินัลหรือคอมพิวเตอร์ PC เป็นต้น แอพพลิเคชั่นในเลเยอร์นี้สามารถนำเข้าหรือออกจากระบบเครือข่ายได้โดยไม่ต้องสนใจว่าจะมีขั้นตอนการทำงานอย่างไร เพราะจะมีชั้นพรีเซนเตชั่นเป็นผู้รับผิดชอบอยู่แล้ว ในรูปแบบ OSI นี้ชั้นแอพพลิเคชั่นจะทำการติดต่อกับชั้นพรีเซนเตชั่นโดยตรง

ในการสื่อสารกันระหว่างระบบหรือระหว่างคอมพิวเตอร์กับคอมพิวเตอร์ โปรโตคอลแบบหนึ่งก็จะถูกใช้สำหรับการสื่อสารระหว่างชั้น N ของคอมพิวเตอร์เครื่องหนึ่งกับชั้น N เดียวกันของคอมพิวเตอร์อีกเครื่องหนึ่ง ซึ่งในระหว่างการสื่อสารข้อมูลกันจริงๆ นั้น โปรโตคอลชั้น N ของอุปกรณ์ทั้งสองเครื่องจะสื่อสารกันผ่านชั้นล่างซึ่งเป็นกายภาพหรือวัตถุ (คือสื่อกลางการสื่อสาร) แต่โดยแนวความคิดของการสื่อสารข้อมูลแล้วถือว่าชั้นที่ N ของเครื่องหนึ่งกำลังติดต่อโดยตรงกับชั้นที่ N ของอีกเครื่องหนึ่ง ซึ่งเรียกการสื่อสารแบบนี้ว่า การสื่อสารแบบเสมือนจริง (Virtual Communication)

โปรโตคอลของในเลเยอร์แต่ละชั้นจะแตกต่างกันออกไป แต่อย่างไรก็ตามการที่เครื่องคอมพิวเตอร์หลายๆ เครื่องจะติดต่อสื่อสารกันได้ ในแต่ละเลเยอร์ของแต่ละเครื่องจะต้องใช้โปรโตคอลแบบเดียวกัน หรือถ้าใช้โปรโตคอลต่างชนิดกันก็ต้องมีอุปกรณ์ หรือซอฟแวร์ที่สามารถแปลงโปรโตคอลที่ต่างกันนั้นให้มีรูปแบบอย่างเดียวกันเพื่อเชื่อมโยงให้คอมพิวเตอร์ทั้ง 2 เครื่องสามารถติดต่อกันได้

ในหนึ่งชั้นของเลเยอร์ไม่ได้กำหนดว่าจะต้องมีเพียงหนึ่งโปรโตคอลเท่านั้นที่อยู่ระดับเลเยอร์เดียวกัน และในทางตรงข้าม ชุดของโปรโตคอลใดๆ อาจจะมีมากกว่าหนึ่งเลเยอร์ประกอบกันเป็นข้อกำหนดของระบบเครือข่ายเรียกว่าชุดโปรโตคอล (Protocal Suite) เช่น ชุดโปรโตคอล TCP/IP (tTransmission Control Protocal/Internet Protocal) เป็นต้น ประโยชน์ในการแบ่งเป็นเลเยอร์ คือกำหนดการติดต่อระหว่างเลเยอร์ทำได้โดยไม่ต้องคำนึงถึงการเปลี่ยนในเลเยอร์ใดๆ ที่ติดกัน

2.6 ความรู้เบื้องต้นเกี่ยวกับโปรโตคอล TCP/IP

2.6.1 ข้อแตกต่างระหว่างชุดโปรโตคอล TCP/IP และรูปแบบ OSI

1.ลำดับการติดต่อสื่อสารของชั้นเลเยอร์ ในรูปแบบ OSI นั้นจะกำหนดลำดับชั้นการสื่อสารที่เป็นลำดับขั้นตอนการติดต่อที่แน่นอน โดยเฉพาะการอินเตอร์เฟซระหว่างชั้นเลเยอร์ ซึ่งทำให้รูปแบบ OSI สามารถเป็นระบบเปิดสำหรับระบบเครือข่ายคอมพิวเตอร์ทั่วไป เพราะไม่ว่าจะมีการเปลี่ยนแปลงโปรโตคอลในเลเยอร์ชั้นใดก็ตามจะไม่มีผลกระทบต่อการสื่อสารเลเยอร์ชั้นถัดไป ในขณะที่ชุดโปรโตคอล TCP/IP จะไม่มีการกำหนดรูปแบบการติดต่อที่ตายตัว เพื่อให้ผู้ออกแบบเครือข่ายมีอิสระสามารถเปลี่ยนแปลงโครงสร้างของเครือข่ายได้ง่ายขึ้น

2.การติดต่อสื่อสารระหว่างเครือข่ายหรือการอินเทอร์เน็ต คือ การติดต่อสื่อสารข้อมูลระหว่างระบบคอมพิวเตอร์ 2 ระบบที่ไม่สามารถติดต่อสื่อสารกันได้โดยผ่านทางเครือข่ายการสื่อสารข้อมูลเพียงเครือข่ายเดียวได้

ต้องอาศัยเครือข่ายตั้งแต่ 2 เครือข่ายขึ้นไปในการติดต่อสื่อสารกัน และเครือข่ายเหล่านี้อาจจะมีลักษณะของเครือข่ายที่ต่างกันก็ได้

ความแตกต่างในเรื่องของอินเทอร์เน็ตระหว่างชุดโปรโตคอล TCP/IP กับรูปแบบ OSI ก็คือในชุดโปรโตคอล TCP/IP จะใช้โปรโตคอลสำหรับอินเทอร์เน็ตที่เรียกว่า โปรโตคอล IP (Internet Protocal) ซึ่งในรูปแบบ OSI จะเรียกว่าโปรโตคอลสำหรับการอินเทอร์เน็ตว่า โปรโตคอลเน็ตเวอร์ค

3.การบริการการเชื่อมต่อการสื่อสาร (Connection Service) ในชุดโปรโตคอล TCP/IP นั้นจะมีการบริการการเชื่อมต่อการสื่อสารระหว่างต้นทางและปลายทาง 2 แบบ คือการบริการแบบ Connectionless และแบบ Connection-Oriented ส่วนในรูปแบบ OSI จะให้ความสำคัญเฉพาะกับการบริการแบบ Connection-Oriented เท่านั้น

4.โปรโตคอลควบคุมการจัดการสื่อสาร ในชุดโปรโตคอล TCP/IP จะใช้โปรโตคอล TCP (Transmission Control Protocal) เป็นโปรโตคอลสำหรับควบคุมการสื่อสาร กำหนดตำแหน่งต้นทางและปลายทาง และอื่นๆ กับข้อมูล ซึ่งในรูปแบบ OSI นั้นจะแบ่งแยกการควบคุมการสื่อสารออกจากกันโดยใช้โปรโตคอลเซสชั่นและโปรโตคอล ทรานสปอร์ตตามลำดับ

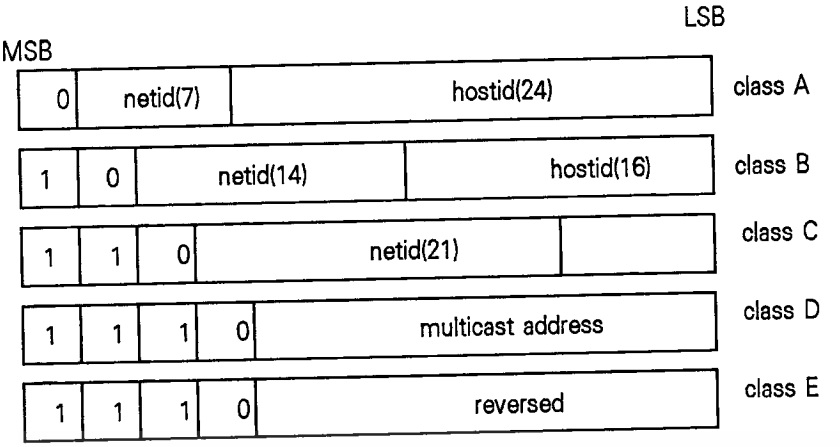
2.6.2 ลักษณะของการติดต่อ แบ่งออกเป็น 2 ชนิดคือ

Connection-Oriented คือการติดต่อที่ต้องมีการเชื่อมโปรเซสที่จะทำการติดต่อก่อนที่จะมีการส่งหรือรับข้อมูล ซึ่งสามารถใช้คำว่าวงจรเสมือน(Virtual Circuit) เพราะว่าจะทำงานเสมือนมีวงจรต่ออยู่ระหว่างโปรเซส ถึงแม้ว่าข้อมูลนี้อาจจะผ่าน Pocket-Switching Network บริการชนิดนี้ส่วนมากจะใช้ในกรณีที่มีข่าวสารต้องการมากกว่าหนึ่งข่าวสาร ดังนั้นสามารถแบ่งขั้นการทำงานออกเป็น

- ขั้นการสร้างการติดต่อ (Connection establishment)
- ขั้นการส่งผ่านข้อมูล (Data tranfer)
- ขั้นยกเลิกการติดต่อ (Connection termination)

Connectionless หรือดาต้าแกรม (Datagram) คือจะไม่มีขั้นการสร้างการติดต่อ และขั้นการยกเลิกการติดต่อ แต่จะมีขั้นการส่งผ่านข้อมูลเพียงเดียว โดยข้อมูลซึ่งเรียกว่าดาต้าแกรมจะถูกส่งจากระบบหนึ่งไปสู่ระบบหนึ่งอย่างเป็นอิสระโดยไม่ขึ้นอยู่กับดาต้าแกรมอื่น

2.6.3 ไอพีแอดเดรส (IP Address) เป็นหัวใจสำคัญของโปรโตคอล IP เพราะเป็นแอดเดรสที่บ่งบอกถึงสถานีปลายทางจริงๆ ภายในระบบเครือข่ายขนาดใหญ่ที่มีการเชื่อมต่อทั้ง LAN และ WAN ทำให้ผู้ใช้งานมองระบบเครือข่ายเสมือนเป็นระบบเครือข่ายเดียวกันได้ และมีวัตถุประสงค์ให้ผู้ใช้งานสามารถกำหนดแอดเดรสของสถานีได้ตามต้องการ



รูปที่ 2.6 แสดงการแบ่งประเภทของไอพีแอดเดรส

ไอพีแอดเดรสเป็นแอดเดคสขนาด 32 บิต แบ่งออกเป็น 5 ประเภทคือประเภท A, B,C, D และ E ดังแสดงในรูปที่ 2.6 แอดเดรสที่ใช้งานโดยทั่วไปคือ 3 ประเภทแรก ประเภทที่ 4 ใช้ในกรณีพิเศษ ส่วนประเภทสุดท้ายสำรองไว้ใช้ในอนาคต ช่วงของไอพีแอดเดรสแต่ละประเภทแสดงดังรูปที่ 2.7

0000000 ..... 00000 0111111 11111	class A
1000000 ..... 00000 1011111 11111	class B
1100000 ..... 00000 1101111 11111	class C
1110000 ..... 00000 1110111 11111	class D
1111000 ..... 00000 1111011 11111	class E

รูปที่ 2.7 ช่วงของไอพีแอดเดรสแต่ละประเภท

ไอพีแอดเดรสแต่ละประเภทมีหมายเลขที่ไม่ซ้ำกัน ซึ่งก็หมายความว่า ถ้ากำหนดสถานีใดๆ ด้วยไอพีแอดเดรส หมายเลขของสถานีก็ไม่ซ้ำด้วยเช่นกัน เมื่อมองแค่ 3 ประเภทแรกจะเห็นได้ว่าภายในแอดเดรสขนาด 32 บิตแบ่งออกเป็น 2 ส่วนย่อยคือ หมายเลขเครือข่าย (network identification (netid)) และหมายเลขสถานี (host identification (hostid)) แต่ละประเภทมีขนาด netid และ hostid ไม่เท่ากันดังในรูปที่ 2.6

ในการอ่าน/เขียนไอพีแอดเดรส แทนที่จะมองเป็นเลขฐานสอง กลับมองเป็นเลขฐานสิบ โดยการแบ่ง 32 บิตออกเป็น 4 ไบต์ย่อย แต่ละไบต์แทนด้วยเลขฐานสิบ 1 ตัว เรียกวิธีการเช่นนี้ว่า "dotted decimal" ดังนั้น

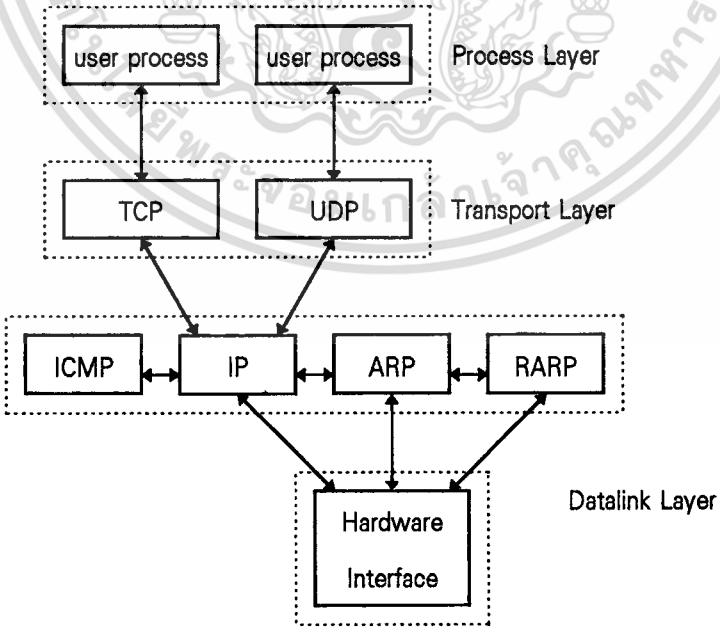
เลขแต่ละตัวของไอพีแอดเดรสมีค่า 0-255 เช่น $(10011110011011000000010011100010)_B$ เขียนในรูป "dotted decimal" ได้เป็น 158.108.4.226 เมื่อกล่าวถึงเฉพาะ netid ของไอพีแอดเดรสประเภท A, B และ C จะใช้เลขฐานสิบจำนวน 1,2 และ 3 หลักในการอ้างถึงตามลำดับ และปล่อยให้ส่วนที่เป็น hostid มีค่าเป็น "0" เช่น 1.0.0.0, 158.108.0.0 และ 195.200.10.0 เป็น netid ประเภท A, B และ C ตามลำดับ

การนำไอพีแอดเดรสมากำหนดเป็นหมายเลขของเครือข่ายหรือสถานีนั้นมีข้อกำหนดปลีกย่อยอยู่หลายประการ ซึ่งผู้วางระบบจำเป็นต้องเข้าใจในส่วนนี้ จึงจะกำหนดแอดเดรสในการใช้งานได้ไม่ผิดพลาด

2.6.4 โครงสร้างของชุดโปรโตคอล TCP/IP

โครงสร้างของสถาปัตยกรรมของชุดโปรโตคอล TCP/IP นั้นแบ่งออกได้เป็น 3 ส่วนหลักๆ คือ ส่วนของกรรมวิธีปฏิบัติการหรือโปรเซส (Process) โฮสต์ (Host) และเครือข่าย (Network) ในส่วนของโปรเซสก็ได้แก่ เอนทิลิตี้หรือแอปพลิเคชั่นที่ต้องการติดต่อสื่อสาร ทุกโปรเซสจะกระทำในเครื่องของโฮสต์ (หรือสเตชั่น) ซึ่งในแต่ละโฮสต์สามารถจะมีหลายๆ เอนทิตี้ (หมายถึงแอพพลิเคชั่น เช่นโปรแกรมระบบจัดการฐานข้อมูล ไฟล์ข้อมูล) ได้พร้อมกัน การสื่อสารกันระหว่างเอนทิตี้ของโฮสต์เครื่องหนึ่งกับเอนทิตี้ของโฮสต์อีกเครื่องหนึ่ง หรือหลายเครื่องจะกระทำโดยผ่านทางเครือข่ายที่โฮสต์เชื่อมต่ออยู่

การทำงานที่สัมพันธ์กันระหว่างโปรเซส โฮสต์ และเครือข่ายของสถาปัตยกรรม TCP/IP ทำให้สามารถจัดรูปแบบของสถาปัตยกรรม TCP/IP ได้เป็น 4 ชั้น และสามารถกำหนดชนิดของโปรโตคอลที่ทำงานในแต่ละชั้นได้เป็น 4 แบบโปรโตคอลเช่นกัน ดังที่ได้กล่าวมาแล้วว่าในชุดโปรโตคอล TCP/IP นั้นเอนทิตี้แต่ละชั้นอาจจะติดต่อสื่อสารข้อมูลโดยผ่านเอนทิตี้ในชั้นเดียวกัน หรือเอนทิตี้ในชั้นล่างลงไปซึ่งไม่จำเป็นต้องเป็นชั้นที่ติดกัน



รูปที่ 2.8 ความสัมพันธ์ระหว่างชุดโปรโตคอล TCP/IP

ชั้นแอพพลิเคชั่น ในชั้นนี้ประกอบด้วยโปรแกรมประยุกต์ที่ใช้ในเครือข่าย เช่น โปรแกรมส่งถ่ายข้อมูล (file-transfer programe) และอาจกล่าวได้ว่าโปรโตคอล TCP/IP ก็คือ โปรโตคอลในชั้นแอพพลิเคชั่นรวมกับชั้นพรีเซนเตชั่นของ OSI โมเดลนั้นเอง

ชั้นทรานสปอร์ต (Transport Layer) ในชั้นนี้เป็นชั้นที่ให้การส่งข้อมูลจากจุดปลายถึงจุดปลาย เปรียบเทียบได้กับชั้นเซสชั่นรวมกับทรานสปอร์ตเลเยอร์นั่นเอง โดยโปรโตคอล TCP/IP มีซอกเก็ต (socket) เป็นจุดปลาย (end-point) ในการสื่อสาร ซึ่งซอกเก็ตนี้ประกอบไปด้วยหมายเลขของคอมพิวเตอร์ และหมายเลขพอร์ต (port) ของเครื่องที่ต้องการส่งข้อมูลไปถึง ในชั้นนี้มีการรับรองให้ถึงที่หมาย และลำดับของข้อมูลที่ส่งโดยไม่ซ้ำและความผิดพลาดของข้อมูล

ชั้นอินเทอร์เน็ต (Internet layer) ในชั้นนี้มีการกำหนดดาต้าแกรม และทำการหาเส้นทางการส่ง หน้าที่ของเลเยอร์นี้เทียบเท่ากับชั้นเน็ตเวอร์คเลเยอร์และดาต้าลิ้งค์ของ OSI โมเดล

ชั้นฟิสิคัล โปรโตคอล TCP/IP ไม่ได้กำหนดรูปแบบของการเชื่อมต่อในระดับนี้ขึ้นใหม่ แต่ใช้มาตรฐานที่มีอยู่เดิมที่กำหนดไว้ก่อน เช่น RS232, อีเทอร์เน็ต (Ethernet) เป็นต้น

บทที่ 3
หลักการทำงาน

โปรแกรมที่สร้างขึ้นในโครงงานนี้ สร้างขึ้นโดยใช้เดลไฟ 1.0 เป็นเครื่องมือในการสร้าง ซึ่งแต่ละโปรแกรมที่สร้างขึ้นโดยเดลไฟจะเรียกว่าโปรเจค(project) ในโครงงานนี้ได้สร้างโปรเจคชื่อ 'FAXTOOL.DPR' ซึ่งเป็นโปรเจคหลักสำหรับเรียกใช้โปรเจคย่อยต่างๆ ซึ่งจะมีหน้าที่ต่างๆ กัน ในบทนี้จะอธิบายลำดับขั้นตอนการทำงานของโปรเจคย่อยๆ ที่สำคัญที่เกี่ยวข้องกับโครงงานนี้ทั้งหมด พร้อมทั้งภาพประกอบ สำหรับผังงาน(Flow Chart) และโค้ดของแต่ละโปรเจคย่อย ได้แสดงไว้ในภาคผนวก ก

สำหรับโปรเจค 'FAXTOOL.DPR' (โปรแกรม 'FAXTASK.PAS') เป็นโปรเจคหลักสำหรับรวมเอาโปรเจคย่อยซึ่งมีหน้าที่การทำงานต่างๆ กันเข้าด้วยกัน ดังต่อไปนี้

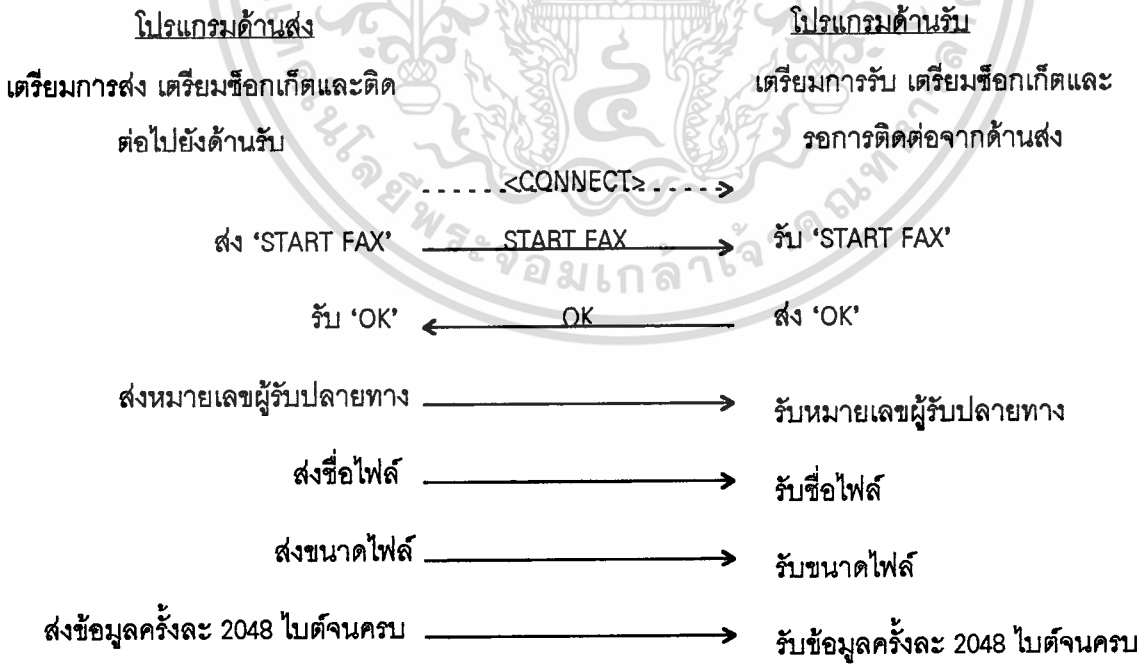
- โปรเจค 'USERSEND.DPR' ทำหน้าที่ส่งไฟล์จากผู้ใช้ต้นทางไปยังเครื่องให้บริการต้นทาง (SERVE1.DPR)
- โปรเจค 'SERVE1.DPR' ทำหน้าที่รับไฟล์จากผู้ใช้ต้นทาง (USERSEND.DPR)
- โปรเจค 'FAXRECV.DPR' ทำหน้าที่รับข้อมูลโทรสารจากเครื่องโทรสารต้นทาง
- โปรเจค 'CLIFUNC.DPR' ทำหน้าที่ส่งไฟล์ข้อมูลไปยังเครื่องให้บริการปลายทาง (SERVFUNC.DPR)
- โปรเจค 'SERVFUNC.DPR' ทำหน้าที่รับไฟล์ข้อมูลจากเครื่องให้บริการต้นทาง (CLIFUNC.DPR)
- โปรเจค 'SENDFAX.DPR' ทำหน้าที่ส่งข้อมูลโทรสารไปยังเครื่องโทรสารปลายทาง
- โปรเจค 'SENDU2.DPR' ทำหน้าที่ส่งไฟล์ไปยังเครื่องผู้ใช้ปลายทาง (USERECV.DPR) พร้อมทั้งเรียกใช้โปรเจค 'SENDFAX.DPR' เมื่อปลายทางของข้อมูลเป็นเครื่องโทรสาร
- โปรเจค 'USERECV.DPR' ทำหน้าที่รับไฟล์จากเครื่องให้บริการปลายทาง (SENDU2.DPR)

นอกจากนี้ ยังมีโปรเจคและโปรแกรมอื่นๆ ซึ่งต้องใช้ประกอบกับโปรเจคย่อยข้างต้นดังนี้

- โปรเจค 'SETTING.DPR' ใช้สำหรับการกำหนดพารามิเตอร์ให้แก่พอร์ตสื่อสารอนุกรมของเครื่องให้บริการ เช่น หมายเลขพอร์ต, อัตราเร็วในการส่ง เป็นต้น
- โปรแกรม 'ABOUT.PAS' และไฟล์ 'ABOUT.TXT' ถูกเรียกใช้โดยโปรเจค FAXTOOL.DPR เพื่อแสดงฟังก์ชันการทำงานของโปรเจคนี้
- โปรแกรม 'FUNCTN1.PAS' ถูกเรียกใช้โดยโปรเจค FAXTOOL.DPR เมื่อต้องการให้โปรแกรมทำงานเป็นเครื่องให้บริการต้นทาง
- โปรแกรม 'FUNCTN2.PAS' ถูกเรียกใช้โดยโปรเจค FAXTOOL.DPR เมื่อต้องการให้โปรแกรมทำงานเป็นเครื่องให้บริการปลายทาง
- โปรแกรม 'COOKIE.PAS' เก็บบันทึกโพรซีเยอร์และฟังก์ชันต่างๆ สำหรับโปรเจค 'SENDFAX.DPR'
- โปรแกรม 'CHEESE.PAS' เก็บบันทึกโพรซีเยอร์และฟังก์ชันต่างๆ สำหรับโปรเจค 'FAXRECV.DPR'
- ไฟล์ชื่อ 'DIAL.NO' ถูกเรียกใช้โดยโปรเจค 'SENDFAX.DPR' ใช้สำหรับเก็บหมายเลขโทรศัพท์ของเครื่องโทรสาร

- ไฟล์ชื่อ 'FAX.INI' ถูกเรียกใช้โดยโปรเจค 'SENDFAX.DPR' และ 'FAXRECV.DPR' ใช้สำหรับเก็บสถานะสุดท้ายของกำหนดพารามิเตอร์ในการสื่อสารผ่านพอร์ตอนุกรม ได้แก่ เลขพอร์ต, อัตราเร็วของข้อมูล, ช่วงเวลาที่ให้รอก่อนที่จะหมุนหมายเลขซ้ำ, และ รีโซลูชันของเครื่องโทรสาร
- ไฟล์ชื่อ 'NORMAL2.FON' เป็นไฟล์ที่เก็บตัวลักษณะของตัวอักษรของโปรแกรมเวิร์ดจุฬาฯ และเวิร์ดราชวิถี แต่ละตัวอักษรกว้าง 16×สูง 20 พิกเซล ใช้ในการแปลงข้อมูลจากไฟล์แบบตัวอักษร (text file) เป็นข้อมูลแบบบิทแม็พ (bitmap)
- ไฟล์ชื่อ 'CODEFILE.PAS' ใช้สำหรับการเข้ารหัสข้อมูลแบบตัวอักษรให้เป็นรหัสฮัฟฟ์แมนและทำการกลับบิทสูง-บิทต่ำของทุกๆ ไบท์ข้อมูล ผลจากการแปลงจะบันทึกเป็นไฟล์ที่เปลี่ยนนามสกุลเป็น '.HUF'
- ไฟล์ชื่อ 'VIEW.PAS' ใช้สำหรับถอดรหัสไฟล์ฮัฟฟ์แมนที่ได้จาก 'CODEFILE.PAS' เพื่อแสดงผลบนฟอร์ม ผลที่ได้จะเป็นข้อมูลแบบบิทแม็พ ซึ่งบันทึกไว้ในไฟล์ชั่วคราวชื่อ 'SEEME.PAS'
- ไฟล์ชื่อ 'DECLARE.PAS' สำหรับการกำหนดตัวแปรบางตัวที่มีการเรียกใช้เหมือนกันมากกว่า 1 โปรแกรม เพื่อลดจำนวนตัวแปรรวมในโปรแกรมทั้งหมด

เนื่องจากการทำงานของโปรแกรมที่เกี่ยวข้องกับการรับ/ส่งไฟล์ผ่านเครือข่ายอินเทอร์เน็ตจะต้องมีการโต้ตอบกันระหว่างทางด้านส่งกับทางด้านรับ ดังนั้นจะอธิบายขั้นตอนการทำงานของโปรแกรมด้านส่งและโปรแกรมด้านรับควบคู่กันไป



รูปที่ 3.1 แสดงการส่งสัญญาณโต้ตอบระหว่างโปรแกรมผู้ส่งและโปรแกรมผู้รับ

หลักการทำงานเมื่อผู้ใช้ต้นทางเป็นผู้ส่งและเครื่องให้บริการต้นทางเป็นผู้รับ

การโต้ตอบระหว่างโปรแกรมทั้งสองเป็นดังรูปที่ 3.1 ผู้ใช้ต้นทางเรียกใช้โปรเจค USERSEND.DPR และผู้ให้บริการต้นทางเรียกใช้โปรแกรม SERVE1.DPR

โปรเจค USERSEND.DPR โปรแกรมชื่อ USER1.PAS

ทำหน้าที่ส่งไฟล์ข้อมูลแบบตัวอักษร(ASCII) ซึ่งเป็นไฟล์จากโปรแกรมเวิร์ดจุฬาฯ หรือ เวิร์ดราชวิถี หรือไฟล์แอสกี้จากโปรแกรมดอส โดยมีพารามิเตอร์ต่างๆ ที่ผู้ใช้ต้องป้อนเข้าไปในโปรแกรมได้แก่

- หมายเลข IP ของเครื่องให้บริการต้นทาง
- หมายเลข IP ของเครื่องผู้ใช้บริการปลายทาง หรือ เบอร์โทรศัพท์ของเครื่องโทรสารปลายทาง
- ชื่อไฟล์ที่ต้องการส่ง ผู้ใช้อาจป้อนโดยตรงหรือใช้ปุ่ม Browse เพื่อเลือกไฟล์ก็ได้

หลังจากที่ผู้ใช้ป้อนพารามิเตอร์ต่างๆ เรียบร้อยแล้วก็เริ่มการส่งโดยกดปุ่ม OK โปรแกรมจะเรียกโปรแกรมย่อยชื่อ initClient และ BeginSendFile เพื่อทำการกำหนดพารามิเตอร์ตามที่ผู้ใช้กำหนดไว้ รวมทั้งพารามิเตอร์อื่นๆ ที่จำเป็น ได้แก่ โหมดการทำงานของซ็อกเก็ตเป็นแบบ non-blocking ซึ่งกำหนดเวลาสำหรับรอการติดต่อเป็น 30 วินาที หมายเลขพอร์ตบริการ(service port) ของโปรแกรมประยุกต์นี้ซึ่งกำหนดเป็น 1407 จากนั้นทำการติดต่อไปยังเครื่องให้บริการตามหมายเลข IP นั้น

เมื่อสามารถติดต่อกับเครื่องให้บริการได้ภายในเวลาที่กำหนดไว้ โปรแกรมจะเรียกใช้โปรแกรมย่อยชื่อ ClientHandshake เพื่อตรวจสอบกับเครื่องให้บริการ (โดยใช้โปรแกรมย่อยของเครื่องให้บริการชื่อ ServerHandshake) ว่ากำลังใช้โปรแกรมเดียวกันหรือไม่ซึ่งมีขั้นตอนการตรวจสอบดังนี้ ClientHandshake ส่งข้อความ 'START FAX' และรอรับ 'OK' จาก ServerHandShake ถ้าไม่มีข้อผิดพลาดแสดงว่าโปรแกรมทางด้านผู้ใช้สามารถเริ่มส่งข้อมูลได้ โดยเรียกใช้โปรแกรมย่อยชื่อ SendFile

โปรแกรมจะเริ่มส่งหมายเลข IP ของเครื่องผู้ใช้ปลายทาง หรือเบอร์โทรศัพท์ของเครื่องโทรสารปลายทางตามที่ผู้ใช้กำหนดไว้ แล้วส่งชื่อไฟล์ ตามด้วยขนาดของไฟล์นั้น จากนั้นจะเป็นส่วนของข้อมูลจริง ซึ่งในที่นี้โปรแกรมจะทำการส่งข้อมูลครั้งละ 2048 ไบต์ จนจบไฟล์ จึงจบการทำงานของโปรแกรม

โปรเจค SERVE1.DPR โปรแกรมชื่อ SERVER1.PAS

ติดตั้งไว้ที่เครื่องให้บริการต้นทาง ผู้ดูแลเครื่องให้บริการต้องกำหนดไดเรกทอรีสำหรับเก็บบันทึกไฟล์ที่รับเข้ามาก่อนที่จะเริ่มทำงาน หรือโปรแกรมกำหนดค่าเริ่มต้นมาให้เป็นไดเรกทอรี 'c:\faxsend'

โปรแกรมจะเริ่มด้วยการเรียกโปรแกรมย่อยชื่อ initServer และ BeginListening เพื่อกำหนดค่าพารามิเตอร์ต่างๆ ที่จำเป็น เช่น โหมดการทำงานของซ็อกเก็ตเป็น non-blocking โดยกำหนดช่วงเวลาที่รอเป็น 0 วินาที หมายถึงให้ซ็อกเก็ตรอการติดต่อจากผู้เรียกได้ไม่จำกัดเวลา หมายเลขพอร์ตบริการของโปรแกรมเป็น 1407 ซึ่งต้องตรงกับการกำหนดใน USERSEND.DPR จากนั้นจะเรียกใช้โปรแกรมย่อยชื่อ DoServer เพื่อรอให้มีการติดต่อจากผู้ใช้ เมื่อสามารถติดต่อได้แล้วโปรแกรมจะเรียกโปรแกรมย่อยชื่อ ServerHandshake เพื่อตรวจ

สอบกับโปรแกรมผู้ใช้ (โปรแกรมย่อยชื่อ ClientHandshake) ว่าเป็นโปรแกรมเดียวกันหรือไม่โดยการรอรับข้อความ 'START FAX' จาก ClientHandshake จากนั้นจะส่ง 'OK' ตอบกลับไป

หลังจากนั้นโปรแกรมจะเรียกโปรแกรมย่อยชื่อ GetFile เพื่อรับข้อมูลโดยรับเอาหมายเลข IP ของผู้ใช้ปลายทางหรือเบอร์โทรศัพท์ของเครื่องโทรสารปลายทาง, รับชื่อไฟล์, และขนาดของไฟล์ที่ส่งมาแล้ว จากนั้นให้บันทึกหมายเลขปลายทางที่รับได้ และชื่อไฟล์ที่รับได้ ลงในไฟล์ที่ให้มีชื่อเป็น fax0X.zzz ในไดเร็กทอรีที่กำหนด ซึ่งไฟล์ที่รับเข้ามาจะเป็น fax00.zzz, fax01.zzz, fax02.zzz ...ตามลำดับ เมื่อบันทึกหมายเลขผู้รับและชื่อไฟล์แล้ว ให้เริ่มรับข้อมูล และบันทึกลงในไฟล์ครั้งละ 2048 ไบต์ จนกระทั่งขนาดของข้อมูลที่รับได้เท่ากับขนาดของไฟล์ แสดงว่าจบไฟล์

ไฟล์ข้อมูลที่รับได้จะมีถูกเพิ่มส่วนหัวไฟล์ ซึ่งประกอบด้วย

- ไอพีแอดเดรส หรือ เบอร์โทรศัพท์ของเครื่องโทรสารปลายทาง
- ชื่อไฟล์ ซึ่งถูกส่งมาจากด้านผู้ใช้ต้นทาง

เมื่อโปรแกรมรับข้อมูลครบ 1 ไฟล์แล้ว โปรแกรมจะถามว่าต้องการรอการติดต่อจากผู้ใช้คนอื่นอีกหรือไม่ ถ้าตอบตกลง ก็จะเริ่มการรอให้มีการติดต่อมาจากผู้ใช้อื่นต่อไป

หลักการทำงานของโปรแกรมเมื่อผู้ให้บริการต้นทางเป็นผู้ส่งและผู้ให้บริการปลายทางเป็นผู้รับ

การโต้ตอบระหว่างโปรแกรมทั้งสองเหมือนในรูปที่ 3.1 แต่ไม่มีการส่งหมายเลขผู้รับปลายทาง ผู้ให้บริการต้นทางเรียกใช้โปรแกรม CLIFUNC.DPR และผู้ให้บริการปลายทางเรียกใช้โปรแกรม SERVFUNC.DPR

โปรเจค CLIFUNC.DPR โปรแกรม UCLIENT.PAS

มีการทำงานคล้ายกับ USERSEND.DPR แต่จะเป็นการส่งข้อมูลที่รับโดย SERVE1.DPR ซึ่งมีชื่อไฟล์เป็น fax00.zzz, fax01.zzz, fax02.zzz,... แล้วส่งไฟล์เหล่านี้ไปให้แก่เครื่องให้บริการปลายทาง

เมื่อเริ่มการทำงานของโปรแกรมต้องพยายามติดต่อไปยังเครื่องให้บริการปลายทาง เมื่อสามารถติดต่อได้สำเร็จแล้ว ก็เรียกโปรแกรมย่อยชื่อ ClientHandshake เพื่อทำการตรวจสอบกับโปรแกรมของเครื่องให้บริการปลายทางเช่นเดียวกับใน USERSEND.DPR

เมื่อส่งไฟล์เสร็จแล้ว ให้ลบไฟล์นั้นทิ้ง จากนั้นโปรแกรมจะถามว่าต้องการส่งไฟล์อื่นต่อไปหรือไม่ ถ้าตอบตกลงโปรแกรมก็จะกลับไปตรวจสอบไฟล์ที่มีลำดับต่อไปอีกครั้งหนึ่งเพื่อส่งไป

หลังจากที่ส่งไฟล์ที่มีนามสกุล '.ZZZ' ซึ่งเป็นไฟล์ที่บันทึกข้อมูลไว้แบบตัวอักษร มีต้นทางคือผู้ใช้ โปรแกรมจะอ่านไฟล์ที่มีนามสกุล '.AAA' ซึ่งเป็นไฟล์ที่รับมาจากเครื่องโทรสารต้นทาง ซึ่งจะกล่าวไว้ในโปรเจค FAXRECV.DPR ในภายหลัง

โปรเจค SERVFUNC.DPR โปรแกรม USERVR2.PAS

มีการทำงานคล้ายโปรเจค SERVER1.PAS แต่ทุกไฟล์ถูกส่งมาจากเครื่องให้บริการต้นทาง (โปรเจค CLIFUNC.DPR) ทั้งสิ้น โดยก่อนที่จะให้เริ่มทำงาน ผู้ดูแลเครื่องให้บริการต้องกำหนดไดเรกทอรีที่ต้องการให้เก็บข้อมูลที่รับมา หรือโปรแกรมกำหนดค่าเริ่มต้นให้เป็นไดเรกทอรี c:\recvfax

โปรแกรมเริ่มทำงานด้วยการรอให้มีการติดต่อมาจากเครื่องให้บริการต้นทาง เมื่อมีการติดต่อสำเร็จแล้ว โปรแกรมจะเรียกโปรแกรมย่อยชื่อ ServerHandshake เพื่อทำการตรวจสอบกับโปรแกรมของเครื่องให้บริการต้นทางเช่นเดียวกับการตรวจสอบในโปรแกรม SERVER1.PAS

หลังจากที่การตรวจสอบถูกต้องแล้ว โปรแกรมจะรับไฟล์จากเครื่องให้บริการต้นทางครั้งละ 1 ไฟล์จนกว่าจะตอบปฏิเสธการรับไฟล์ต่อไป ไฟล์ที่บันทึกได้จะมีชื่อตามที่ส่งมาจาก CLIFUNC.DPR ซึ่งก็คือ fax00.zzz, fax01.zzz, fax02.zzz,.. และ fax00.aaa, fax01.aaa, fax02.aaa,..

หลังจากบันทึกข้อมูลแล้ว ข้อมูลภายในไฟล์นั้นจะเหมือนกับทางด้านส่ง(เครื่องให้บริการต้นทาง) ทุกประการ

หลักการทำงานเมื่อผู้ให้บริการปลายทางเป็นผู้ส่งและผู้ใช้ปลายทางเป็นผู้รับ

การโต้ตอบระหว่างโปรแกรมทั้งสองเหมือนรูปที่ 3.1 แต่ไม่มีการส่งหมายเลขผู้รับปลายทาง ผู้ให้บริการปลายทางเรียกใช้โปรเจค SENDU2.DPR และผู้ใช้ปลายทางเรียกใช้โปรเจค USERECV.DPR

โปรเจค SENDU2.DPR โปรแกรม TOUSER2.PAS

มีการทำงานคล้ายกับโปรเจค CLIFUNC.DPR แต่โปรเจคนี้เป็นการส่งข้อมูลที่รับมาจากเครื่องให้บริการต้นทาง เมื่อเริ่มการทำงานโปรแกรมจะไปอ่านไฟล์ซึ่งมีรูปแบบชื่อเป็น FAX00.ZZZ, FAX01.ZZZ,.. ซึ่งเก็บไว้ในไดเรกทอรีที่กำหนดไว้ในโปรแกรม USERVER2.PAS ซึ่งเป็นไฟล์ที่ส่งมาจากผู้ใช้ต้นทาง

โปรแกรมจะอ่านส่วนหัวของไฟล์ ซึ่งจะได้ค่าของหมายเลขปลายทางแล้วทำการวิเคราะห์ว่าเป็นไอพีแอดเดรสของเครื่องผู้ใช้ปลายทาง หรือเบอร์โทรศัพท์ของเครื่องโทรสารปลายทาง ถ้าพบว่าเป็นไอพีแอดเดรสให้ทำการติดต่อไปยังเครื่องผู้ใช้ตามเลขไอพี นั้นโดยใช้โปรแกรม TOUSER2.PAS นี้ในการติดต่อ แต่ถ้าเป็นเบอร์โทรศัพท์ของเครื่องโทรสารต้องเรียกโปรแกรม USENDFAX.PAS แทน แล้วโปรแกรมจะส่งชื่อไฟล์ faxoX.zzz ไปให้กับตัวแปร passFile:String ในโปรแกรม USENDFAX.PAS เพื่อส่งข้อมูลไปยังเครื่องโทรสาร สำหรับการส่งข้อมูลไปยังเครื่องโทรสารปลายทางจะกล่าวถึงในโปรเจค SENDFAX.DPR ในภายหลัง

นอกจากนี้ ที่ส่วนหัวของไฟล์ยังประกอบด้วยชื่อของไฟล์ต้นฉบับอีกด้วย ถ้าโปรแกรมตรวจพบว่าปลายทางควรเป็นเครื่องผู้ใช้ปลายทาง โปรแกรมจะส่งชื่อไฟล์ที่อ่านได้ไปให้ปลายทางด้วย

หลังจากที่โปรแกรมสามารถติดต่อไปยังเครื่องผู้ใช้ปลายทางตามเลขไอพีที่อ่านได้แล้ว โปรแกรมจะเรียกโปรแกรมย่อยเพื่อทำการตรวจสอบกับโปรแกรมของเครื่องให้บริการปลายทางในลักษณะเดียวกันกับในโปรแกรม UCLIENT.PAS ก่อนที่จะส่งชื่อไฟล์ และขนาดของไฟล์ ออกไป

จากนั้นโปรแกรมจะส่งข้อมูลครั้งละ 2048 ไบต์จนจบไฟล์ เมื่อส่งครบแล้วโปรแกรมจะถามว่าต้องการส่งไฟล์ลำดับต่อไปหรือไม่ ถ้าตอบตกลงโปรแกรมก็จะกลับไปตรวจสอบไฟล์ที่มีลำดับต่อไปอีกครั้งหนึ่งเพื่อส่งไป

หลังจากที่ส่งไฟล์ที่มีนามสกุล '.ZZZ' เรียบร้อยแล้ว โปรแกรมจะไปอ่านไฟล์ที่มีนามสกุล '.AAA' เพื่อส่งไปยังเครื่องโทรสารปลายทางต่อไป ซึ่งที่ส่วนหัวของไฟล์จะประกอบด้วยเบอร์โทรศัพท์ของเครื่องโทรสารปลายทางเท่านั้น โปรแกรมจะทำการอ่านแล้วส่งหมายเลขนั้นไปให้ USENDFAX.PAS ต่อไป

โปรเจค USERECV.DPR **โปรแกรม** USER2.PAS

เป็นการรับไฟล์ที่ส่งมาจากเครื่องให้บริการปลายทางซึ่งมีการทำงานคล้ายกับโปรเจค SERVE1.DPR และ SERVFUNC.DPR ผู้ใช้ต้องกำหนดไดเรกทอรีสำหรับเก็บไฟล์ หรือโปรแกรมกำหนดค่าเริ่มต้นเป็น 'c:\recvfax\data' และไฟล์ที่เก็บบันทึกจะมีชื่อไฟล์ตามที่ส่งมาจากเครื่องให้บริการปลายทางซึ่งเป็นชื่อไฟล์ที่แท้จริงซึ่งส่งมาจากผู้ใช้ต้นทาง ดังนั้น หลังจากรับไฟล์แล้ว ไฟล์จะมีลักษณะเหมือนไฟล์ต้นทางทุกประการ

หลักการทำงานของโปรแกรมสำหรับรับข้อมูลจากเครื่องโทรสาร

โปรเจค FAXRECV.DPR โปรแกรม UWAITFAX.PAS ทำหน้าที่รับข้อมูลจากเครื่องโทรสารต้นทาง ซึ่งจะถูกเรียกใช้โดยเครื่องให้บริการต้นทาง

โปรแกรมจะให้ผู้ใช้ป้อนพารามิเตอร์ต่างๆ ได้แก่ ไดเรกทอรีสำหรับบันทึกข้อมูลที่รับมา และพารามิเตอร์เพื่อกำหนดค่าเริ่มต้นให้แก่พอร์ตสื่อสารอนุกรมเมื่อผู้ใช้กดปุ่ม Config บนฟอร์ม ซึ่งจะแสดงไว้ในโปรเจค SETTING.DPR ในภายหลัง นอกจากนี้ยังกำหนดให้ใช้จำนวนพาริตี้เป็น none, 1 บิทหยุด, และ 8 บิทข้อมูล และในระหว่างการส่งข้อมูลจากคอมพิวเตอร์ไปยังแฟกซ์โมเด็มนั้นจะมีการควบคุมการไหลของข้อมูลแบบ CTR/RTS หลังจากที่กำหนดค่าต่างๆ เรียบร้อยแล้ว โปรแกรมจะส่งคำสั่งเพื่อส่งค่าที่เซ็ตไว้ไปให้แฟกซ์โมเด็ม และรีเซ็ตแฟกซ์โมเด็มก่อนที่จะทำการส่งไฟล์ ซึ่งคำสั่งต่างๆ ที่ใช้เรียงลำดับต่อไปนี้

ATZ	เพื่อรีเซ็ตแฟกซ์โมเด็ม
ATH &D2 E0 X0	สั่งให้วางแฮนด์เซ็ต, ให้มีการวางแฮนด์เซ็ตเมื่อ DTR ไม่แอคทีฟ, ให้ตอบสนองแบบตัวอักษร และให้แสดงคำสั่งที่สะท้อนกลับจากโมเด็ม
AT+FCLASS=2	เพื่อเซ็ตแฟกซ์โมเด็มให้เป็นแฟกซ์โมเด็มระดับ 2
AT+FLID=""	ไม่มีการส่งค่า ID string ใดๆ
AT+FDCC = 1,5,0,2,0,0,0,0	เพื่อกำหนดพารามิเตอร์เป็นค่าเริ่มต้นในการตกลงกันระหว่างที่มีการส่งโทรสาร
AT+FAA=0,+FCR=1	เพื่อให้แฟกซ์โมเด็มตอบรับอัตโนมัติทันทีเมื่อมีการเรียกเข้าและกำหนดให้แฟกซ์โมเด็มสามารถที่จะรับข้อมูลข่าวสารได้ระหว่างการรับโทรสาร

หลังจากนั้นโปรแกรมจะรอจนกว่าจะมีการโทรมาจากเครื่องโทรสารต้นทาง เมื่อได้รับสัญญาณการเรียก แฟกซ์โมเด็มก็จะส่งสัญญาณ "RING" มาให้คอมพิวเตอร์ ซึ่งเมื่อโปรแกรมตรวจสอบเจอสัญญาณดัง

กล่าวก็จะส่งสัญญาณ "+FCON" ซึ่งแสดงว่าเริ่มมีการติดต่อแลกเปลี่ยนข่าวสารระหว่างผู้ส่งและผู้รับ ต่อมาเมื่อโปรแกรมได้รับ "OK" แล้วก็จะส่งคำสั่ง "AT+FDR" เพื่อให้แฟกซ์โมเด็มเข้าสู่เฟส C ในการรับข้อมูล จนเมื่อได้รับสัญญาณ "CONNECT" แล้ว โปรแกรมจะส่งรหัสค่า 12h เพื่อให้แฟกซ์โมเด็มเริ่มส่งข้อมูลโทรสารมาให้กับคอมพิวเตอร์

หลังจากนั้นโปรแกรมจะรับข้อมูลและบันทึกไว้ในไฟล์ชั่วคราวชื่อ TEMP.FAX จนกระทั่งได้รับรหัส DLE(แอสกี้ 16) และ ETX(แอสกี้ 3) ก็เป็นอันสิ้นสุดการรับข้อมูลจากเครื่องโทรสารต้นทาง

หลังจากนั้นโปรแกรมจะนำข้อมูลที่บันทึกไว้มาทำการเรียงบิทใหม่จากบิท 0-7 เป็น 7-0 เพื่อให้สามารถใช้กับโปรแกรม VIEW.PAS เพื่อแสดงรายละเอียดของโทรสารที่ได้รับมา ซึ่งส่วนของโปรแกรม VIEW.PAS นั้นจะกล่าวถึงต่อไป

ในขณะที่โปรแกรมอ่านข้อมูลจากไฟล์ชั่วคราว ก็จะทำการตรวจสอบรหัส EOL ไปด้วย ถ้าพบรหัสนี้มากกว่า 2 ครั้งติดต่อกัน ก็แสดงว่าเป็นจุดสิ้นสุดของข้อมูล โปรแกรมจะใส่รหัส EOL เพิ่มลงไปจนครบ 6 ชุด แล้วบันทึกไว้เป็นไฟล์ที่มีชื่อเป็น fax0X.huf (fax0.huf, fax01.huf, fax02.huf,...) หลังจากนั้นผู้ควบคุมต้องเปิดไฟล์เพื่อเพิ่มส่วนหัวของไฟล์รวมเข้าไปในข้อมูลพร้อมทั้งเปลี่ยนนามสกุลของไฟล์เป็น '.AAA' เพื่อให้สามารถใช้ในโปรเจค SENDU2.DPR ได้ ดังนั้นทางด้านผู้ส่งโทรสารต้นทางต้องเขียนหมายเลขปลายทางที่ต้องการไว้ที่หัวของเอกสารด้วยเพื่อให้ผู้ควบคุมเครื่องให้บริการสามารถเพิ่มส่วนหัวของไฟล์ลงไปได้

ดังนั้นหลังจากการทำงานสิ้นสุดลง ไฟล์ข้อมูลที่รับได้จะมีรูปแบบเป็นรหัสฮัฟฟ์แมนซึ่งถูกกลับบิททุกๆ ไบท์ข้อมูล และเพิ่มส่วนหัวของไฟล์เอาไว้ว่าปลายทางของเอกสารนี้มีหมายเลขเป็นอะไร

หลักการทำงานของโปรแกรมสำหรับส่งข้อมูลไปยังเครื่องโทรสาร

โปรเจค SENDFAX.DPR โปรแกรม USENDFAX.PAS ถูกเรียกใช้โดยเครื่องให้บริการปลายทางสำหรับส่งไฟล์ที่มีการเข้ารหัสฮัฟฟ์แมน 1 มิติ ไปยังเครื่องโทรสารปลายทาง โดยอาจมีการเรียกโปรแกรมนี้ 2 กรณีคือ ถูกเรียกใช้โดยตรง หรือเมื่อโปรเจค SENDU2.DPR ตรวจสอบพบว่าปลายทางเป็นเครื่องโทรสาร

ถ้าโปรแกรมถูกเรียกใช้โดยตรง จะต้องป้อนพารามิเตอร์ต่างๆ ได้แก่ ชื่อไฟล์ที่ต้องการส่ง, ID string, หมายเลขเครื่องโทรสารปลายทางและพารามิเตอร์เพื่อกำหนดค่าเริ่มต้นให้แก่พอร์ตสื่อสารอนุกรมเช่นเดียวกับในโปรเจค FAXRECV.DPR แต่ถ้าโปรแกรมถูกเรียกใช้โดย SENDU2.DPR ไม่ต้องป้อนชื่อไฟล์ และหมายเลขเครื่องโทรสารปลายทางเพราะพารามิเตอร์ทั้งสองนี้จะถูกส่งมาจากโปรแกรม TOUSER2.PAS

คำสั่งต่างๆ ที่ใช้กำหนดค่าเริ่มต้นและรีเซ็ตแฟกซ์โมเด็ม เรียงลำดับต่อไปนี้

ATZ	เพื่อรีเซ็ตแฟกซ์โมเด็ม
ATE1V1	เพื่อเซ็ตให้มีการตอบสนองกลับเป็นแบบข้อความ (ASCII)
ATM1 X4 &D2 S7=120 S8=2	เพื่อเซ็ตให้ลำโพงของโมเด็มมีเสียงดังจนกว่าจะได้รับคลื่นพาหะ, ให้มีการอีเนเบิ้ล (enable) รหัสตอบสนองทุกอย่าง, ให้มีการวางแฮนด์เซ็ตเมื่อ DTR ไม่แอคทีฟ รวมทั้งเซ็ตเวลาในการรอคลื่นพาหะ(หน่วยวินาที)

และการเซ็ตเวลาที่โมเด็มจะหยุดรอเมื่อเจอรหัส ',' ในขณะที่หมุนหมายเลขปลายทาง

AT+FLID="*IDstring* " กำหนดค่า local ID string

AT+FCR=1 เพื่อให้แฟกซ์โมเด็มพร้อมจะรับข่าวสารในขณะที่มีการส่งแฟกซ์

AT+FCLASS=2 เพื่อเซ็ตแฟกซ์โมเด็มให้เป็นแฟกซ์โมเด็มระดับ 2

AT+FDIS=*IntToStr(ToResol-1)+','+IntToStr(TransRate)+*',0,2,0,0,0,5

เพื่อกำหนดพารามิเตอร์ในการตกลงกันระหว่างที่มีการส่งแฟกซ์ สำหรับความหมายของพารามิเตอร์แต่ละตัวแสดงไว้ในภาคผนวก

ก่อนที่จะส่งข้อมูลโปรแกรมจะตรวจสอบว่าเป็นไฟล์รหัสฮัฟฟ์แมนหรือไม่ ถ้าไม่ใช่ โปรแกรมจะทำการเข้ารหัสไฟล์นั้นก่อน โดยเรียกใช้โปรแกรม 'CODEFILE.PAS' ซึ่งจะกล่าวถึงในภายหลัง โดยที่ไฟล์ที่จะถูกเข้ารหัสก่อนส่งไปได้แก่

- ไฟล์ที่ส่งมาจากโปรแกรม TOUSER2.PAS และมีนามสกุลเป็น '.ZZZ'
- ไฟล์ที่กำหนดโดยตัวโปรแกรมเองและไม่มีนามสกุลเป็น '.HUF'

จากนั้นโปรแกรมจะหมุนหมายเลขเครื่องโทรสารปลายทางตามค่าในตัวแปร PhoneNumber ซึ่งได้จากการอ่านไฟล์ 'DIAL.NO' ซึ่งจะกล่าวถึงในหัวข้อโปรเจค DIAL.DPR ในภายหลัง (หรือค่าที่ส่งมาจากโปรแกรม TOUSER2.PAS ในกรณีที่ตรวจพบว่าปลายทางเป็นเครื่องโทรสาร) จากนั้นจึงส่งไฟล์ไปครั้งละ 1024 ไบท์จนกระทั่งจบไฟล์ ซึ่งทุกครั้งที่ไบท์ข้อมูลตรงกับรหัส DLE(DataLink Escape) ซึ่งมีค่า 10h จะต้องทำการส่งไบท์นั้นซ้ำอีกครั้งหนึ่งเพื่อให้แฟกซ์โมเด็มตีความว่าเป็นไบท์ข้อมูล ไม่ใช่รหัสคำสั่ง และเมื่อส่งข้อมูลจนครบแล้วต้องส่งรหัส DLE และ ETX ไปยังแฟกซ์โมเด็มเพื่อให้รู้ว่าจบข้อมูลแล้ว จากนั้นก็จะวางสายพร้อมทั้งรีเซ็ตพอร์ตอนุกรมและแฟกซ์โมเด็ม

หลักการทำงานของโปรเจคที่ใช้ประกอบอื่นๆ

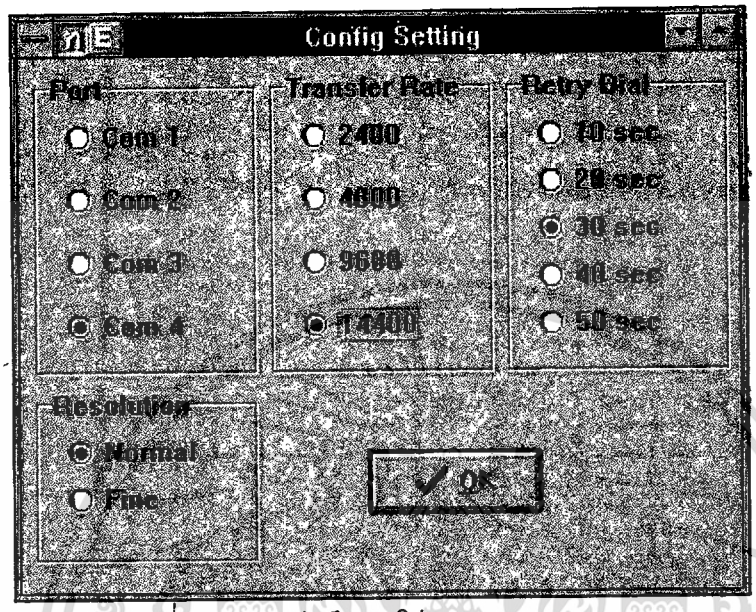
โปรเจค SETTING.DPR โปรแกรม UCONFIG.PAS

ใช้สำหรับกำหนดสถานะการส่งข้อมูลผ่านพอร์ตสื่อสารอนุกรม ได้แก่

- Port ให้ผู้ใช้เลือกหมายเลขพอร์ตอนุกรมที่เชื่อมต่ออยู่กับแฟกซ์โมเด็ม ซึ่งมีค่าให้เลือกเป็น com1, com2, com3 และ com4
- Transfer Rate ให้ผู้ใช้เลือกอัตราเร็วของการส่งข้อมูลผ่านพอร์ตอนุกรม ซึ่งมีค่าให้เลือกเป็น 2400, 4800, 9600 และ 14000 บิตต่อวินาที
- Resolution ให้ผู้ใช้เลือกการแสดงผลข้อมูลของเครื่องโทรสาร ซึ่งมีค่าให้เลือกเป็น
  - Normal Resolution (204 dpi:dots per inch สำหรับการสแกนในแนวนอน และ 98 dpi สำหรับการสแกนในแนวตั้ง) ดังนั้นจะได้ 1728 จุดต่อ 1 เส้นสแกน และ 1143 เส้นสแกน สำหรับ 1 หน้ากระดาษ A4
  - Fine Resolution (204 dpi สำหรับการสแกนในแนวนอน และ 196 dpi สำหรับการสแกนในแนวตั้ง) จะได้ 2287 เส้นสแกน สำหรับ 1 หน้ากระดาษ A4

- Retry Dial ให้ผู้ใช้เลือกช่วงเวลาหยุด ก่อนที่จะทำการหมุนหมายเลขใหม่ถ้าพบว่าการหมุนหมายเลขครั้งแรกไม่สำเร็จ ซึ่งมีค่าให้เลือกเป็น 10, 20, 30, 40 และ 50 วินาที

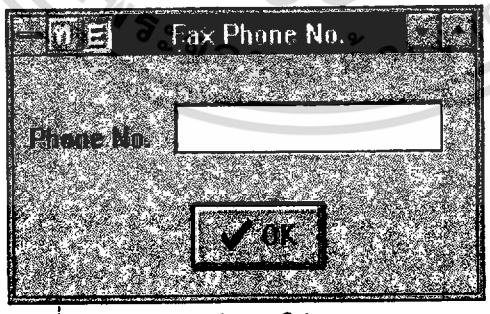
โปรแกรมนี้จะถูกเรียกโดยโปรแกรม USENDFAX.PAS และ UWAITFAX.PAS เมื่อผู้ใช้กดปุ่ม Config.. บนฟอร์ม



รูปที่ 3.2 แสดงฟอร์มของโปรเจค SETTING.DPR

โปรเจค DIAL.DPR โปรแกรม UPHONE.PAS

สำหรับให้ผู้ใช้กำหนดหมายเลขเครื่องโทรสารปลายทาง โดยโปรแกรมนี้ถูกเรียกใช้โดยโปรแกรม USENDFAX.DPR ซึ่งจะมีฟอร์มดังรูปที่ 3.3 เมื่อผู้ใช้กำหนดหมายเลขเครื่องโทรสารปลายทางบนฟอร์มตามรูปที่ 3.3 เรียบร้อยแล้ว โปรแกรมจะบันทึกหมายเลขไว้ในไฟล์ 'DIAL.NO'



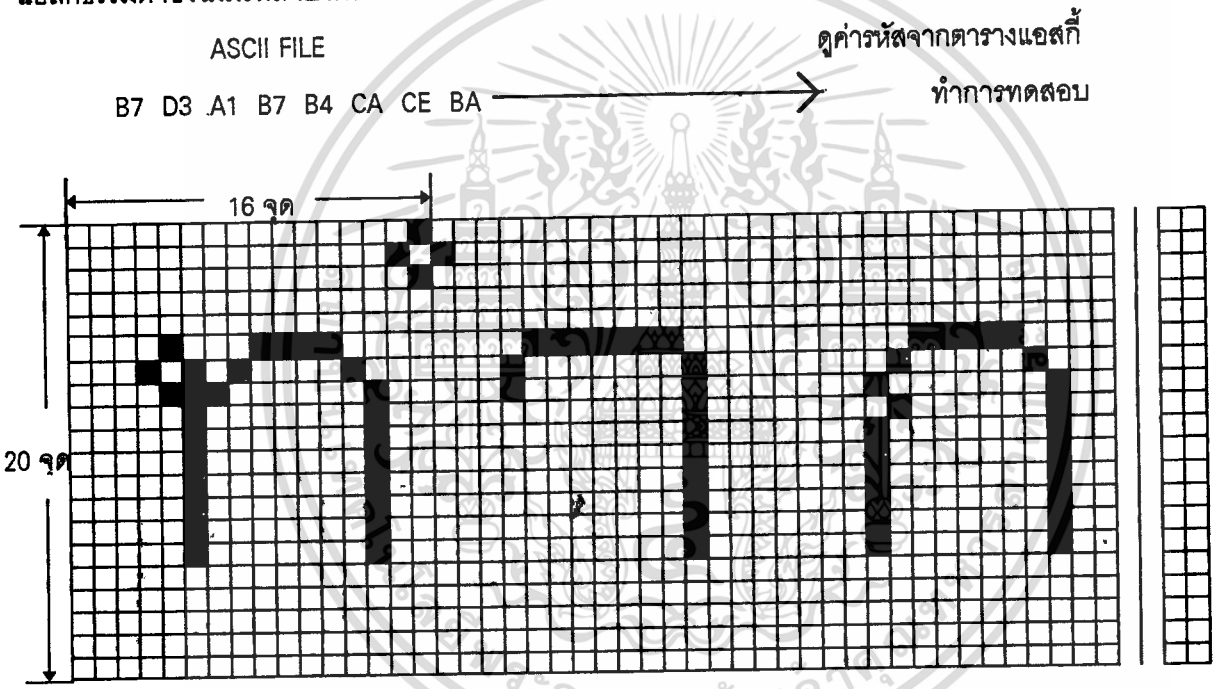
รูปที่ 3.3 แสดงฟอร์มของโปรเจค DIAL.DPR

โปรแกรม CODEFILE.PAS

โปรแกรมนี้ถูกเรียกใช้โดยโปรแกรมย่อย LoadHuffName ในโปรแกรม SENDFAX.PAS ใช้สำหรับแปลงไฟล์แบบตัวอักษร (เก็บข้อมูลเป็นรหัสแอสกี้) จากโปรแกรม CW, RW ซึ่งใช้รหัส สมอ. (ตารางแสดงความสัมพันธ์ของอักษรภาษาไทยกับตารางแอสกี้แสดงไว้ในภาคผนวก) หรือไฟล์แอสกี้ของระบบปฏิบัติการดอส

(DOS) ทั่วๆ ไป ให้เป็นรหัสฮัฟฟ์แมน 1 มิติ ซึ่งโปรแกรมให้บันทึกไฟล์ที่ได้จากการแปลงนี้ในชื่อเดิมแต่นามสกุลของไฟล์เป็น '.HUF'

ในการแปลงจากรหัสแอสกี้เป็นรหัสฮัฟฟ์แมน 1 มิติ โปรแกรมจะอ่านไฟล์แอสกี้ขึ้นมาโดยอ่านจนเจอรหัสขึ้นบรรทัดใหม่ (แอสกี้ 13) ก็นำมาแปลงเป็นรหัสฮัฟฟ์แมนครั้งหนึ่งโดยการเอารหัสแอสกี้มาหาในตารางแอสกี้ที่เป็นบิทแม็พของฟอนต์ไฟล์ชื่อ 'NORMAL2.FON' ซึ่งมีขนาด 16x20 จุด และนำมาแปลงเป็นบิทแม็พต่อๆ กันจนได้ขนาด 1728 จุด จำนวน 20 แถว แล้วนำแต่ละแถวที่มีขนาด 1728 จุด มาเข้ารหัสฮัฟฟ์แมน 1 มิติ จนครบ 20 แถว ดังแสดงไว้ดังรูปที่ 3.4 ก็จะเสร็จการเข้ารหัสแอสกี้ 1 บรรทัด ซึ่งกำหนดไว้ 88 ตัวอักษรต่อบรรทัด ถ้าในบรรทัดมีแอสกี้มากกว่า 88 ตัวอักษร ก็จะปัดไปเป็นบรรทัดใหม่ โปรแกรมจะทำการเข้ารหัสอย่างนี้ไปเรื่อยๆ จนอ่านเจอรหัสแอสกี้ 1Ah ซึ่งเป็นรหัสจบไฟล์ของโปรแกรม CW และ RW แต่ถ้าเป็นไฟล์แอสกี้ธรรมดาซึ่งไม่มีรหัสจบไฟล์ โปรแกรมจะอ่านไปจนหมดไฟล์



รูปที่ 3.4 แสดงการแปลงจากรหัสแอสกี้เป็นบิทแม็พ

จากรูปที่ 3.4 ซึ่งแสดงเป็นบิทแม็พ และเก็บบันทึกไว้ในรูปรหัสไบนารีแล้ว จะถูกแปลงเป็นรหัสฮัฟฟ์แมน 1 มิติ ดังขั้นตอนต่อไปนี้

00000000 00000001 00000000 00000000 00000000 000000...

00000000 00000010 10000000 00000000 00000000 000000...

00000000 00000001 00000000 00000000 00000000 000000...

00000000 00000000 00000000 00000000 00000000 000000...

00000000 00000000 00000000 00000000 00000000 000000...

00001000 11110000 00001111 11100000 00000111 110000...

00010101 00001000 00010000 00010000 00001000 001000...

00001110 00000100 00010000 00010000 00010000 000100...

.....

นำแต่ละแถวมาเข้ารหัสฮัฟฟ์แมน

EOL รหัสบิทขาว 17 ตัว +รหัสบิทดำ 1 ตัว+...จนครบ 1728 ตัว

EOL รหัสบิทขาว 16 ตัว+รหัสบิทดำ 1 ตัว + รหัสบิทขาว 1 ตัว+รหัสบิทดำ 1 ตัว+...จนครบ 1728 ตัว

EOL รหัสบิทขาว 17 ตัว+รหัสบิทดำ 1 ตัว+...จนครบ 1728 ตัว

EOL รหัสบิทขาว 1728 ตัว

EOL รหัสบิทขาว 1728 ตัว

EOL รหัสบิทขาว 4 ตัว+รหัสบิทดำ 1 ตัว+รหัสบิทขาว 3 ตัว+รหัสบิทดำ 4 ตัว+รหัสบิทขาว 8 ตัว+รหัสบิทดำ 5 ตัว+จนครบ 1728 ตัว

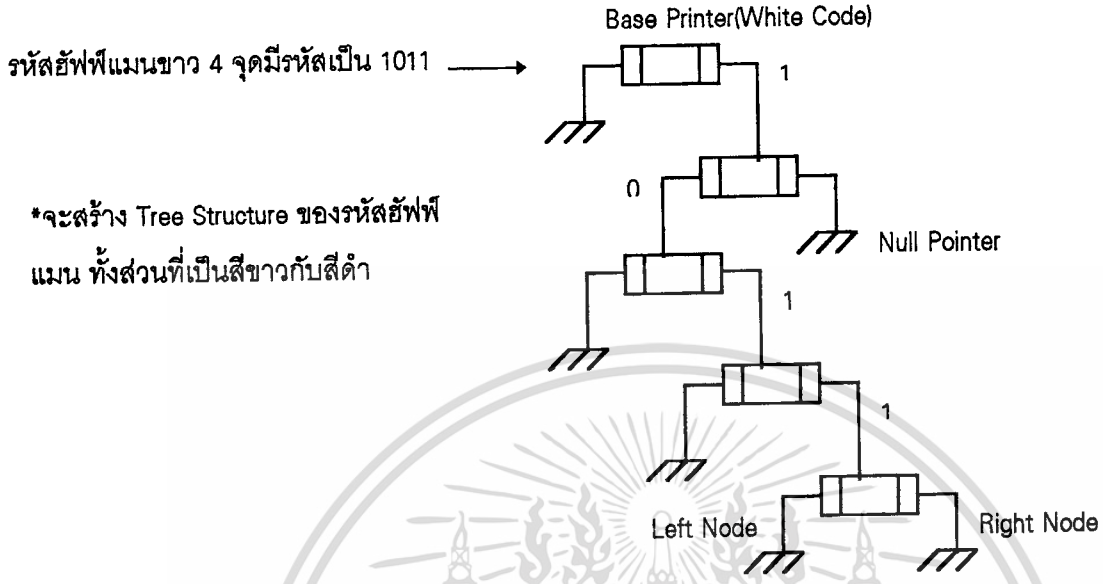
EOL รหัสบิทขาว 3 ตัว+รหัสบิทดำ 1 ตัว+รหัสบิทขาว 1 ตัว+รหัสบิทดำ 1 ตัว+รหัสบิทขาว 1 ตัว+รหัสบิทดำ 1 ตัว+รหัสบิทขาว 4 ตัว+รหัสบิทดำ 1 ตัว+รหัสบิทขาว 5 ตัว+รหัสบิทดำ 1 ตัว+รหัสบิทขาว 7 ตัว+รหัสบิทดำ 1 ตัว+รหัสบิทขาว 8 ตัว+รหัสบิทดำ 1 ตัว+รหัสบิทขาว 5 ตัว+รหัสบิทดำ 1 ตัว+จนครบ 1728 ตัว

ไฟล์รหัสฮัฟฟ์แมนที่ได้จะถูกเรียงสลับบิท จากบิท 0-7 เป็นบิท 7-0 ของทุกๆ ไบท์ข้อมูล เนื่องจากในขณะที่ส่งข้อมูล แฟกซ์โมเด็มจะทำการส่งข้อมูลสลับจากบิทสูงไปยังบิทต่ำก่อน ทำให้เมื่อถึงทางด้านรับ บิทที่รับเข้ามาจะกลับกัน โดยบิทสูงที่ส่งเข้ามาก่อน ทางด้านรับจะรับเข้ามาเป็นบิทต่ำ ดังนั้นเพื่อให้ทางด้านรับสามารถรับข้อมูลได้อย่างถูกต้อง จึงต้องทำการสลับตำแหน่งของบิทดังกล่าว

โปรแกรม VIEW.PAS

เป็นโปรแกรมสำหรับแสดงผลของไฟล์ฮัฟฟ์แมน โดยการแสดงผลจะแสดงเป็นจุดขาว,ดำในลักษณะบิทแม็พตามรูปแบบของรหัสฮัฟฟ์แมนในไฟล์ โดยขั้นแรกจะทำการอ่านข้อมูลในไฟล์รหัสฮัฟฟ์แมน แล้วนำมาทำการสลับเรียงบิทจาก 0-7 เป็นบิท 7-0 จากนั้นนำไปเปรียบเทียบดูว่าตรงกับรหัสฮัฟฟ์แมนตัวไหน ก็จะได้ค่าของจำนวนจุดขาวจุดดำ สำหรับรหัสฮัฟฟ์แมน 1 แถวที่จบด้วยรหัส EOF จะประกอบด้วย 1728 จุด เมื่อแปลงเป็นไบท์ข้อมูลโดยใช้ 1 บิทต่อจุด ก็จะได้ทั้งหมด 216 ไบท์ แต่เนื่องจากในรหัสแอสกี้ 1 บรรทัดจะแทนด้วยรหัสฮัฟฟ์แมน 20 แถว ดังนั้นจะต้องเก็บทั้งหมด 216*20=4320 ไบท์ต่อรหัสแอสกี้ 1 บรรทัด ต่อมาเมื่อแปลงเป็นบิทแม็พทั้งหมดก็จะบันทึกไว้ในไฟล์ชั่วคราวชื่อ 'SEEME.BMP' แล้วจึงนำมาแสดงผลโดยเรียกใช้โปรแกรม UBITMAP.PAS

ในการแปลงกลับจากรหัสฮัฟฟ์แมนเป็นบิทแม็พนั้น โปรแกรมจะทำการสร้างโครงสร้างข้อมูลที่เป็นแบบต้นไม้(Tree Structure) เพื่อใช้ในการถอดรหัสกลับเป็นบิทแม็พ โดยแสดงการสร้างไว้ดังรูป 3.5 และก่อนที่จะทำการถอดรหัสโปรแกรมต้องกลับบิทจาก 0-7 เป็น 7-0 ทุกๆ ไบท์ข้อมูลที่อ่านได้



รูปที่ 3.5 แสดงการสร้างโครงสร้างข้อมูลที่เป็นแบบต้นไม้ของรหัสฮัฟฟ์แมน 1 มิติ

เมื่อได้สร้างโครงสร้างข้อมูลที่เป็นแบบต้นไม้ของรหัสฮัฟฟ์แมนของรหัสสีขาวและสีดำ แล้วก็จะนำเอาไบท์ข้อมูลที่เป็นรหัสฮัฟฟ์แมนมาทำการเช็คไปทีละบิท โดยถ้าบิทใดเป็น 1 ก็จะไปตามโหนดขวา(Right Node) ของโครงสร้างต้นไม้ ส่วนในกรณีที่เป็น 0 ก็จะไปทางโหนดซ้าย(Left Node) ไปอย่างนี้เรื่อยๆ จนกว่าจะได้ค่าจำนวนของจุดขาวหรือดำที่สัมพันธ์กับรหัสฮัฟฟ์แมนดังกล่าว ทำให้ทราบว่าเป็นจุดสีขาวหรือสีดำกี่จุด แล้วจึงบันทึกไว้เป็นไฟล์ชั่วคราวชื่อ 'SEEME.BMP' ก่อนที่จะอ่านไฟล์นั้นเพื่อแสดงผล

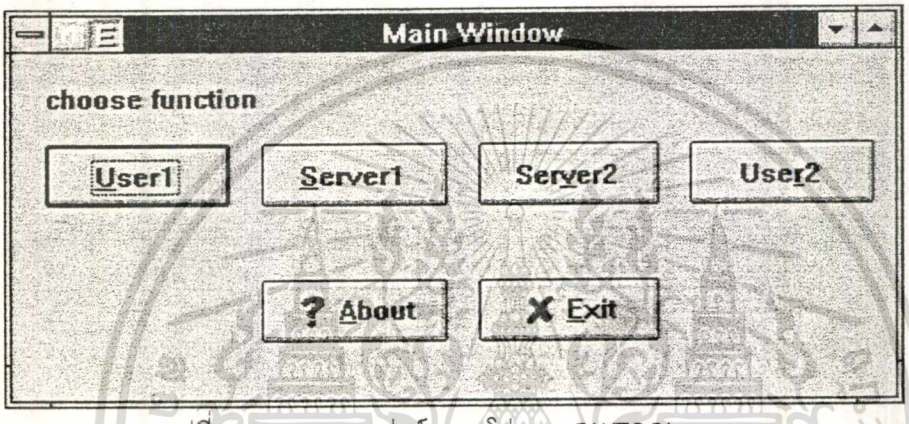
โปรแกรม UBITMAP.PAS

ใช้สำหรับการอ่านไฟล์ที่มีการเข้ารหัสฮัฟฟ์แมนแล้ว โดยไฟล์ที่ต้องการอ่าน ซึ่งมีนามสกุลเป็น '.HUF' จะถูกถอดรหัสฮัฟฟ์แมนโดยโปรแกรม 'VIEW.PAS' และบันทึกไว้ในไฟล์ชื่อ 'SEEME.BMP' ซึ่งข้อมูลที่ได้จะมีลักษณะเป็นบิทแม็พ และโปรแกรม UBITMAP.PAS นี้จะนำไฟล์นั้นมาแสดงผลลงบนฟอร์มทีละจุดจนครบ ซึ่งขนาดของภาพที่ได้จะมีความกว้าง 1728 จุดเท่ากับรีโซลูชั่นของเครื่องโทรสารนั่นเอง

บทที่ 4
การทดลองและผลการทดลอง

ในบทนี้จะกล่าวถึงการทดลองรับ-ส่งข้อมูลเป็นขบวนการทีละขั้นตอน โดยมีจุดมุ่งหมายที่จะรับไฟล์จากเครื่องผู้ใช้ต้นทางและส่งไปยังเครื่องผู้ใช้ปลายทาง, เครื่องโทรสารปลายทาง รวมทั้งรับไฟล์จากเครื่องโทรสารต้นทางเพื่อส่งไปยังเครื่องโทรสารปลายทางได้ โดยมีขั้นตอนต่างๆ ดังต่อไปนี้

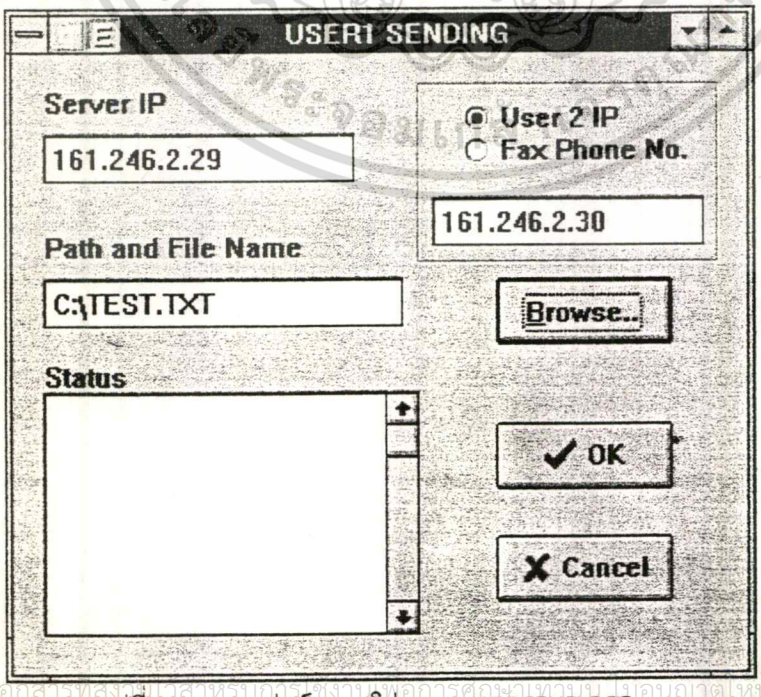
เมื่อทำการคอมไพล์โปรแกรมเป็นไฟล์ FAXTOOL.EXE และทำการรันโปรแกรม จะปรากฏฟอร์มหลัก (Main Window) ดังรูปที่ 4.1



รูปที่ 4.1 แสดงภาพฟอร์มของโปรเจค FAXTOOL

1. การทดลองส่งไฟล์ข้อความจากเครื่องผู้ใช้ต้นทางไปยังเครื่องให้บริการต้นทาง

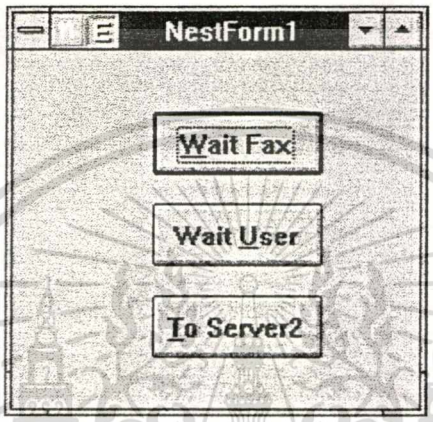
เมื่อผู้ใช้เลือกปุ่ม User1 ซึ่งใช้สำหรับเครื่องผู้ใช้ต้นทาง โปรแกรมจะไปเรียก USERSEND.DPR และได้ฟอร์ม USER1 SENDING ดังรูปที่ 4.2



รูปที่ 4.2 แสดงฟอร์มของโปรเจค USERSEND.DPR

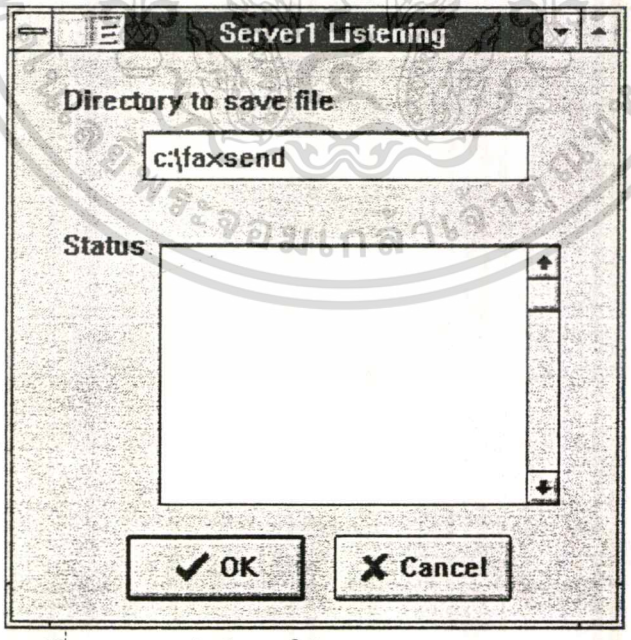
ที่ช่อง Server IP ให้ทำการกำหนดไอพีแอดเดรสของเครื่องให้บริการที่ทำหน้าที่รับไฟล์ โดยในที่นี้ใช้เครื่องที่มีเลขไอพีเป็น 161.246.2.29 เป็นเครื่องให้บริการต้นทาง และที่ช่อง User2 IP/Fax Phone No. ให้เลือกปลายทางของข้อมูลที่ต้องการ ซึ่งในการทดลองเลือกปลายทางเป็นเครื่องคอมพิวเตอร์ที่มีไอพีแอดเดรสเป็น 161.246.2.30 จากนั้นที่ช่อง Path and File Name ให้ใช้ปุ่ม Browse.. เพื่อเลือกไฟล์ที่ต้องการส่ง หรืออาจกำหนดโดยตรงก็ได้ ในการทดลองให้ส่งไฟล์ C:\TEST.TXT

ส่วนทางด้านเครื่องผู้ให้บริการต้นทาง จากรูปที่ 4.1 ให้เลือกฟังก์ชันของ Server1 ซึ่งโปรแกรมภายในจะไปเรียกฟอร์ม NestForm1 ซึ่งภายในประกอบด้วยฟังก์ชันการทำงานของเครื่องให้บริการต้นทาง ดังรูปที่ 4.3



รูปที่ 4.3 แสดงฟอร์มของโปรแกรม FUNCTN1.PAS

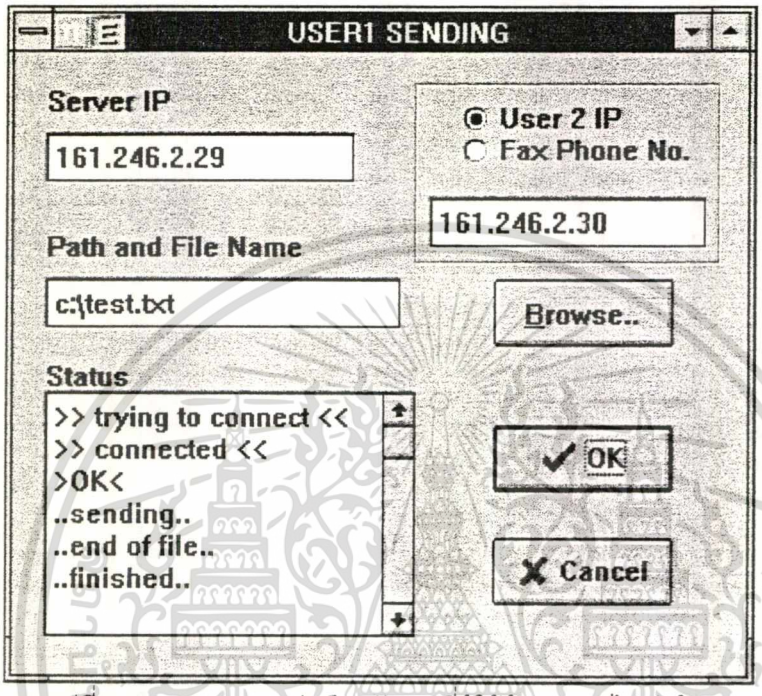
ในการทดลอง เลือกฟังก์ชัน Wait User เพื่อให้เครื่องให้บริการต้นทางรอรับไฟล์ข้อมูลจากผู้ใช้ต้นทาง โดยโปรแกรมจะไปเรียกใช้โปรเจค SERVE1.DPR ซึ่งมีฟอร์มดังรูปที่ 4.4



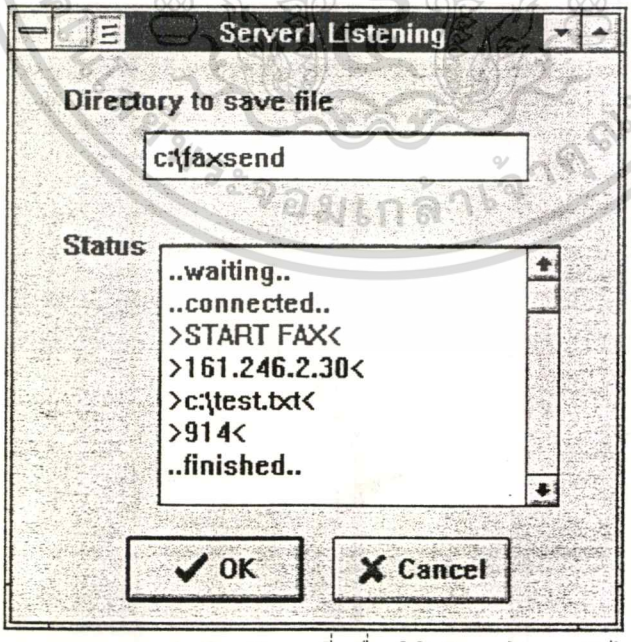
รูปที่ 4.4 แสดงฟอร์มของโปรเจค SERVE1.DPR

ในการทดลองได้กำหนดไดเรกทอรีสำหรับเก็บบันทึกไฟล์ที่รับเข้ามา เป็น C:\FAXSEND จากนั้นคลิ๊กที่ OK เพื่อเริ่มโปรแกรม และรอการติดต่อจากผู้ใช้ต้นทาง

หลังจากที่ผู้ใช้ต้นทางได้กำหนดค่าพารามิเตอร์ต่างๆ และคลิ๊กที่ OK เพื่อเริ่มโปรแกรมและติดต่อมายังเครื่องให้บริการ แล้วหลังจากที่โปรแกรมทั้งสองทำการส่ง/รับไฟล์เรียบร้อยแล้วได้ผลของฟอร์มดังรูปที่ 4.5-4.6



รูปที่ 4.5 แสดงผลของฟอร์มหลังจากที่ผู้ใช้ต้นทางส่งไฟล์แล้ว



รูปที่ 4.6 แสดงผลของฟอร์มหลังจากที่เครื่องให้บริการต้นทางรับไฟล์แล้ว

หลังจากที่เครื่องให้บริการรับไฟล์แล้วจะทำการบันทึกไว้เป็นไฟล์ชั่วคราวชื่อ FAX00.ZZZ ได้ทำการตรวจสอบข้อมูลในไฟล์ชั่วคราวนั้นเปรียบเทียบกับไฟล์ทางด้านผู้ส่ง ได้ผลดังรูปที่ 4.7-4.8

หลังจากนั้นทำการทดลองส่งไฟล์ซ้ำอีกครั้งหนึ่งแต่กำหนดปลายทางเป็นเครื่องโทรสารที่ต่อคู่สายโทรศัพท์กับหมายเลข 16 โดยจากรูปที่ 4.2 เปลี่ยนจาก User 2 IP เป็น Fax Phone No. และเปลี่ยนจาก 161.246.2.30 เป็น 16 ไฟล์ที่ได้จะเก็บบันทึกเป็น FAX01.ZZZ ข้อมูลของไฟล์ที่รับได้เป็นดังรูปที่ 4.9

```
This is the file for testing the project 'Faxtool'.
การทดลองส่งไฟล์ระหว่างเครื่องผู้ใช้-เครื่องให้บริการ

aA bB cC dD eE fF gG hH iI jJ
kK lL mM nN oO pP qQ rR
sS tT uU vV wW xX yY zZ
! " # $ % & ' ( ) * + , - . /
```

รูปที่ 4.7 ไฟล์ TEST.TXT ทางด้านผู้ใช้ต้นทาง

```
Destination Fax IP (161.246.2.30)
File Name (TEST.TXT)
This is the file for testing the project 'Faxtool'.
การทดลองส่งไฟล์ระหว่างเครื่องผู้ใช้-เครื่องให้บริการ

aA bB cC dD eE fF gG hH iI jJ
kK lL mM nN oO pP qQ rR
sS tT uU vV wW xX yY zZ
```

รูปที่ 4.8 ไฟล์ FAX00.ZZZ ที่รับได้โดยเครื่องให้บริการต้นทาง

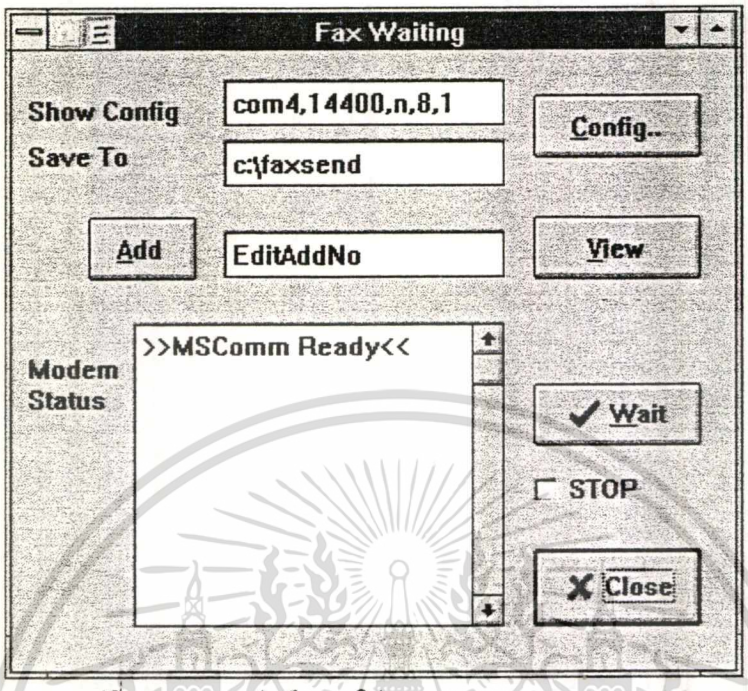
```
Fax Phone No (16)
File Name (TEST.TXT)
This is the file for testing the project 'Faxtool'.
การทดลองส่งไฟล์ระหว่างเครื่องผู้ใช้-เครื่องให้บริการ

aA bB cC dD eE fF gG hH iI jJ
kK lL mM nN oO pP qQ rR
sS tT uU vV wW xX yY zZ
```

รูปที่ 4.9 ไฟล์ FAX01.ZZZ ที่รับได้โดยเครื่องให้บริการต้นทาง

2. การทดลองส่งเอกสารจากเครื่องโทรสารไปยังเครื่องให้บริการต้นทาง

เครื่องให้บริการใช้ฟังก์ชัน Wait Fax จากรูปที่ 4.3 โปรแกรมจะเรียกใช้โปรเจค FAXRECV.DPR ซึ่งมีฟอร์มดังรูปที่ 4.10



รูปที่ 4.10 แสดงฟอร์มของโปรเจค FAXRECV.DPR

ใช้ปุ่ม Config.. ในการกำหนดพารามิเตอร์ของพอร์ตสื่อสารอนุกรม และกำหนดไดเรกทอรีสำหรับเก็บบันทึกไฟล์ที่รับไว้ เป็น C:\FAXSEND เช่นเดียวกับการกำหนดในฟังก์ชัน Wait User

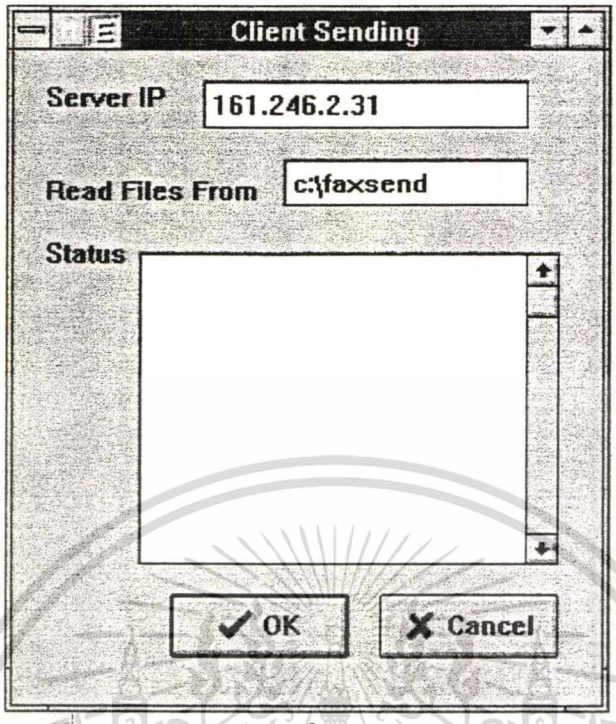
หลังจากทดลองส่งไฟล์จากผู้ใช้ต้นทางและจากเครื่องโทรสาร ตามที่กล่าวมาข้างต้นแล้วจะได้ไฟล์ที่บันทึกในไดเรกทอรี C:\FAXSEND ทั้งหมด 3 ไฟล์คือ

1. FAX00.ZZZ ปลายทางเป็นเครื่องที่มีไอพีแอดเดรสเป็น 161.246.2.30
2. FAX01.ZZZ ปลายทางเป็นเครื่องโทรสารหมายเลข 16
3. FAX00.AAA ปลายทางเป็นเครื่องโทรสารหมายเลข 16

3. ทำการส่งไฟล์จากเครื่องให้บริการต้นทางไปยังเครื่องให้บริการปลายทาง

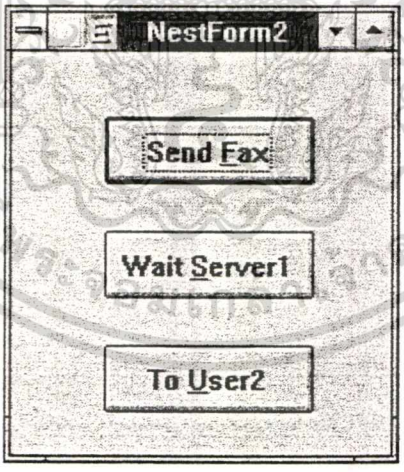
เครื่องให้บริการต้นทางเรียกใช้ฟังก์ชัน To Server2 จากในรูปที่ 4.3 โปรแกรมจะเรียกใช้โปรเจค CLIFUNC.DPR ซึ่งมีฟอร์มเป็นดังรูปที่ 4.11

ทำการกำหนดไดเรกทอรีซึ่งเก็บบันทึกไฟล์สำหรับที่จะส่ง ซึ่งก็คือ C:\FAXSEND ตามที่กำหนดไว้ในโปรเจค SERVE1.DPR และ FAXRECV.DPR ตามรูปที่ 4.4 และ 4.10 ตามลำดับ และในช่อง Server IP ให้กำหนดเป็นไอพีแอดเดรสของเครื่องให้บริการปลายทางซึ่งเป็นผู้รับไฟล์ ซึ่งในการทดลองให้เครื่องที่มีไอพีแอดเดรสเป็น 161.246.2.31 เป็นเครื่องให้บริการปลายทาง



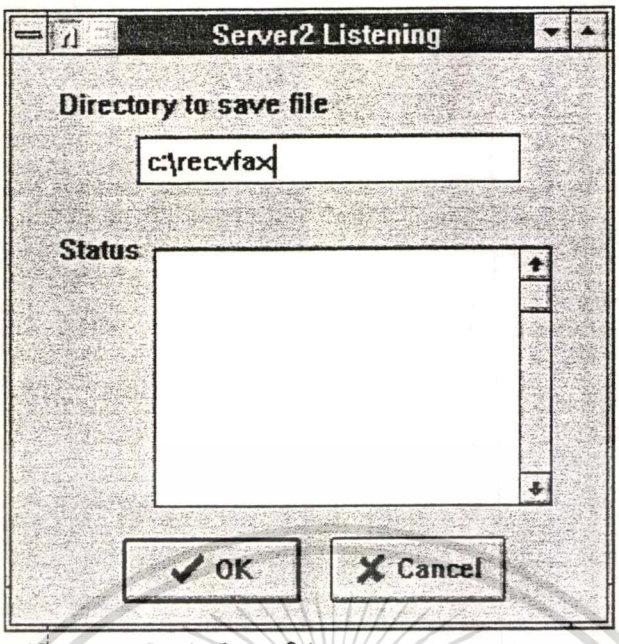
รูปที่ 4.11 แสดงฟอร์มของโปรเจค CLIFUNC.DPR

ทางด้านเครื่องให้บริการปลายทางเรียกใช้ฟังก์ชัน Server2 จากรูปที่ 4.1 ซึ่งโปรแกรมจะเรียกฟอร์ม NestForm2 ซึ่งภายในประกอบด้วยฟังก์ชันการทำงานของเครื่องให้บริการปลายทาง ดังรูปที่ 4.12



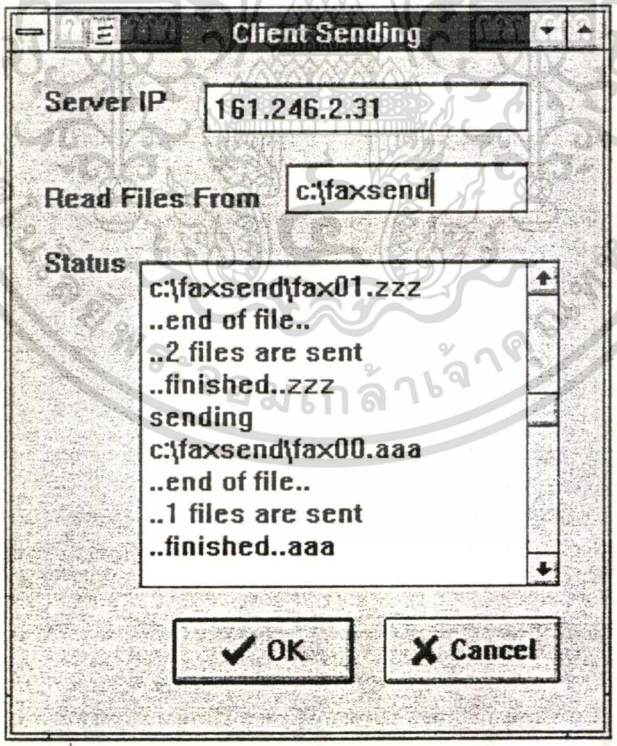
รูปที่ 4.12 แสดงฟอร์มของโปรแกรม FUNCTN2.PAS

เลือกใช้ฟังก์ชัน Wait Server1 เพื่อทำหน้าที่รับไฟล์จากเครื่องให้บริการต้นทาง โปรแกรมจะเรียกใช้โปรเจค SERVFUNC.DPR ซึ่งมีฟอร์มดังรูปที่ 4.13



รูปที่ 4.13 แสดงฟอร์มของโปรเจค SERVFUNC.DPR

ทำการกำหนดไดเรกทอรีสำหรับเก็บบันทึกไฟล์เป็น C:\RECVFAX จากนั้นคลิ๊กที่ปุ่ม OK เพื่อรอการติดต่อจากเครื่องให้บริการต้นทาง และหลังจากที่มีการส่ง/รับไฟล์เรียบร้อยแล้วจะได้ผลการส่ง/รับ ดังรูปที่ 4.14 และ 4.15 ดังนี้



รูปที่ 4.14 แสดงผลหลังจากเครื่องให้บริการต้นทางส่งไฟล์ FAX00.ZZZ, FAX01.ZZZ และ FAX00.AAA

รูปที่ 4.15 แสดงผลหลังจากเครื่องให้บริการปลายทางรับไฟล์ FAX00.ZZZ, FAX01.ZZZ และ FAX00.AAA

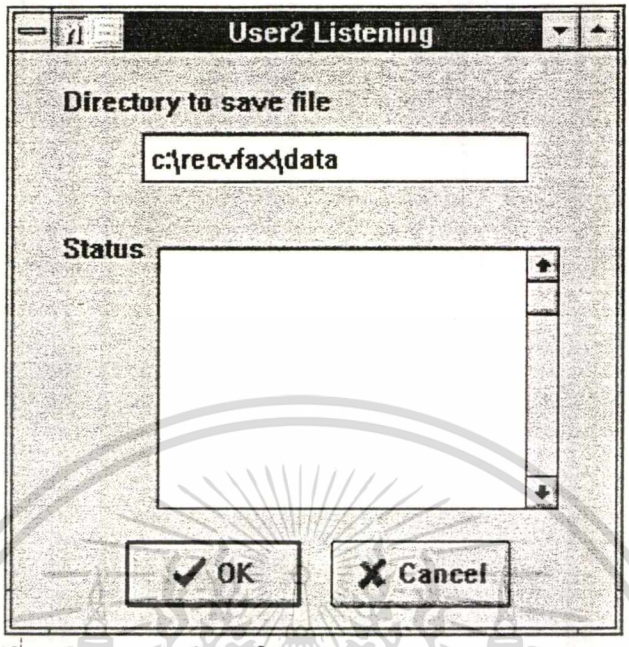
4. การทดลองส่งไฟล์จากเครื่องให้บริการปลายทาง

ไฟล์ต่างๆ อาจถูกส่งไปยังผู้ใช้ปลายทาง หรือเครื่องโทรสารปลายทาง โดยทางด้านส่งซึ่งคือเครื่องให้บริการปลายทางเรียกใช้ฟังก์ชัน To User2 จากรูปที่ 4.12 ซึ่งโปรแกรมจะเรียกใช้โปรเจค SENDU2.DPR ซึ่งมีฟอร์มดังรูปที่ 4.16

รูปที่ 4.16 แสดงฟอร์มของโปรเจค SENDU2.DPR

ภายในไดเรกทอรี C:\RECVFAX จะมีไฟล์อยู่ 3 ไฟล์คือ FAX00.ZZZ, FAX01.ZZZ และ FAX00.AAA ตามที่รับมาโดยโปรแกรม SERVFUNC.DPR เมื่อโปรแกรมอ่านไฟล์ FAX00.ZZZ เป็นอันดับแรกและพบว่าปลายทางของไฟล์เป็นเครื่องผู้ใช้ปลายทางที่มีไอพีแอดเดรสเป็น 161.246.2.30

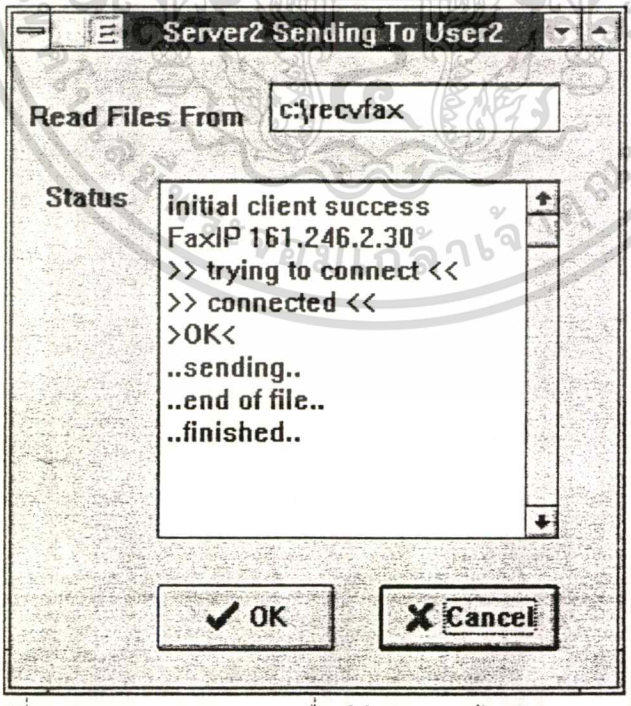
ส่วนทางด้านเครื่องผู้ใช้ปลายทางซึ่งมีไอพีแอดเดรสเป็น 161.246.2.30 เรียกใช้ฟังก์ชัน User2 จากรูปที่ 4.1 ซึ่งโปรแกรมจะเรียกใช้โปรเจค USERECV.DPR ซึ่งมีฟอร์มดังรูปที่ 4.17



รูปที่ 4.17 แสดงฟอร์มของโปรเจค USERECV.DPR

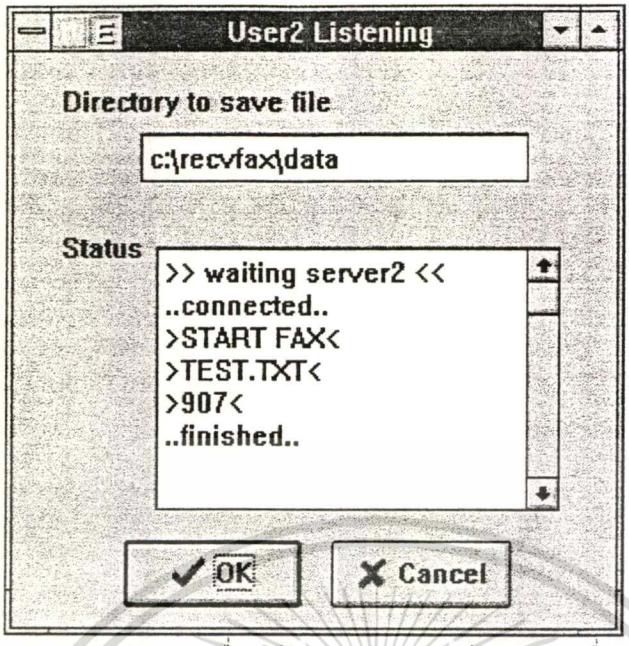
ทำการกำหนดไดเรกทอรีสำหรับเก็บบันทึกไฟล์ที่รับได้เป็น C:\RECVFAX\DATA แล้วคลิ๊กปุ่ม OK เพื่อรอรับไฟล์

หลังจากทำการส่ง/รับไฟล์เรียบร้อยแล้วจะได้ผลการส่ง/รับ ดังรูปที่ 4.18-4.19



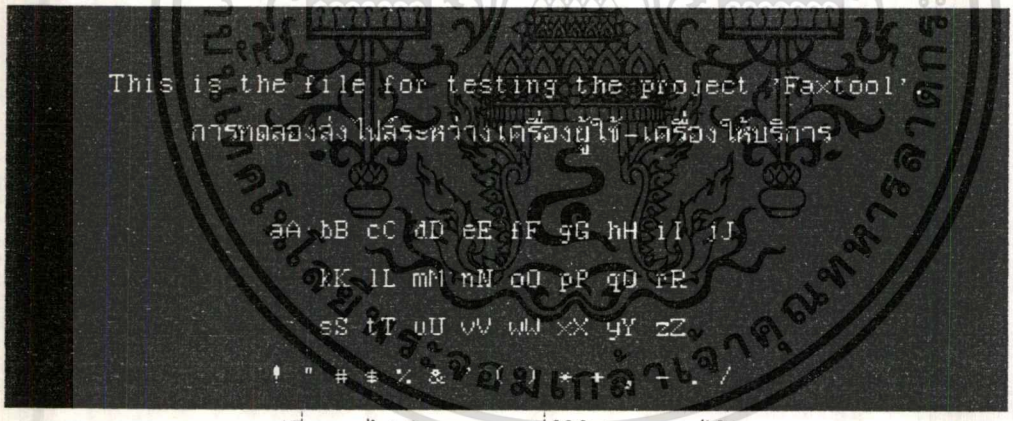
รูปที่ 4.18 แสดงผลหลังจากเครื่องให้บริการส่งไฟล์ FAX00.ZZZ

ไปยังเครื่องที่มีไอพีแอดเดรส 161.246.2.30



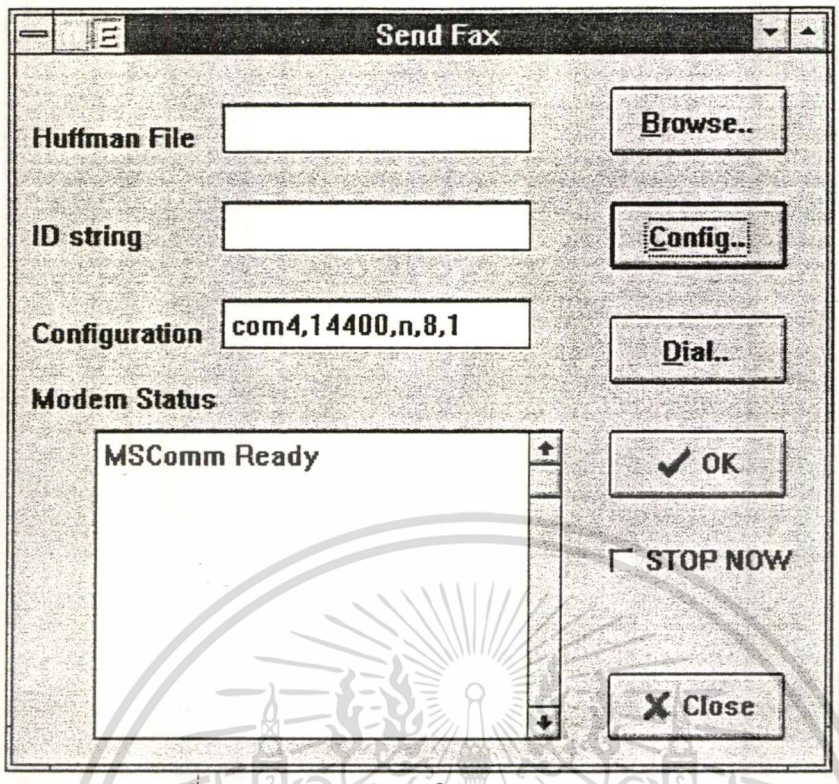
รูปที่ 4.19 แสดงผลหลังจากเครื่องให้บริการต้นทางรับไฟล์จากเครื่องให้บริการ

ชื่อไฟล์ที่ถูกส่งมาจากเครื่องให้บริการคือ TEST.TXT ซึ่งเป็นชื่อไฟล์เดิมที่ส่งมาจากผู้ใช้ต้นทาง ส่วนทางด้านผู้ใช้ปลายทางรับไฟล์และบันทึกไว้เป็นชื่อ TEST.TXT รูปที่ 4.20 แสดงข้อมูลภายในไฟล์ TEST.TXT ที่เครื่องผู้ใช้ปลายทาง



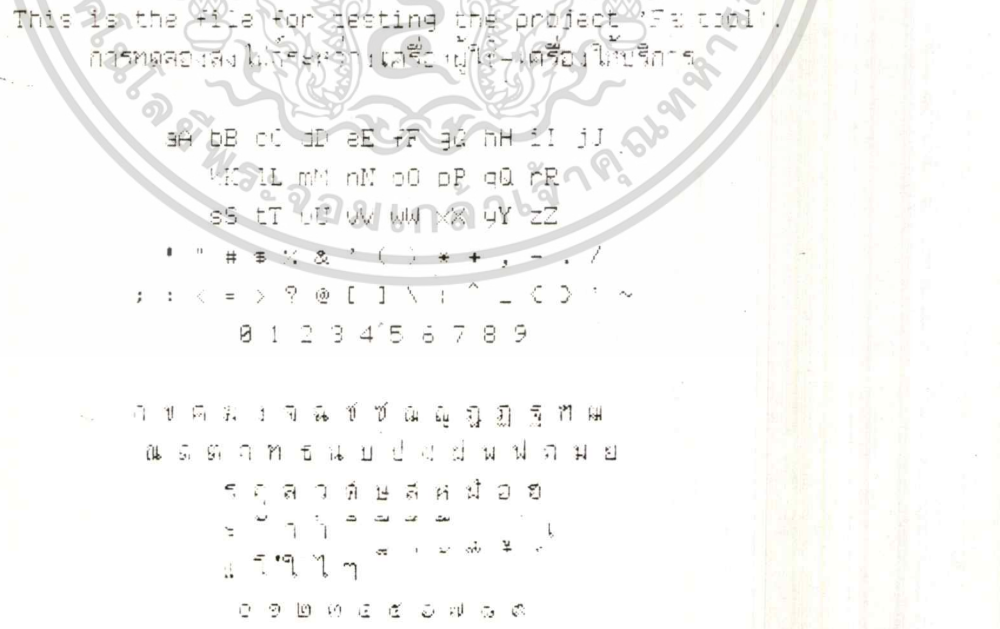
รูปที่ 4.20 ไฟล์ TEST.TXT ที่ผู้ใช้ปลายทางได้รับ

หลังจากที่โปรเจค SENDU2.DPR ส่งไฟล์ FAX00.ZZZ ซึ่งมีปลายทางเป็นเครื่องผู้ใช้เสร็จเรียบร้อยแล้ว โปรแกรมจะอ่านไฟล์ FAX01.ZZZ เป็นลำดับต่อไป ซึ่งปลายทางเป็นเครื่องโทรสารหมายเลย 16 โปรแกรมจะเรียกใช้โปรเจค SENDFAX.DPR ซึ่งมีฟอร์มดังรูปที่ 4.21



รูปที่ 4.21 แสดงฟอร์มของโปรเจค SENDFAX.DPR

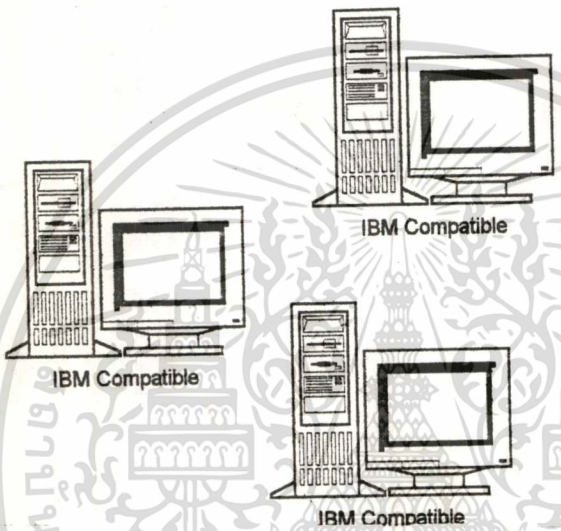
ทำการคลิ๊กปุ่ม OK โปรแกรมจะทำการเข้ารหัสข้อมูลเป็นฮัฟฟ์แมนก่อน แล้วจึงส่งไฟล์ไปยังเครื่องโทรสาร และหลังจากส่งไฟล์เรียบร้อยแล้วจะได้ผลการส่งดังรูปที่ 4.22



รูปที่ 4.22 เอกสารแฟกซ์ที่ส่งจากเครื่องให้บริการมายังเครื่องโทรสาร

หลังจากนั้น โปรเจค SENDU2.DPR จะอ่านไฟล์ FAX00.AAA เป็นลำดับต่อไป ซึ่งโปรแกรมตรวจพบว่าไฟล์มีนามสกุลเป็น .AAA จึงเรียกใช้โปรแกรม SENDFAX.DPR อีกครั้งหนึ่ง

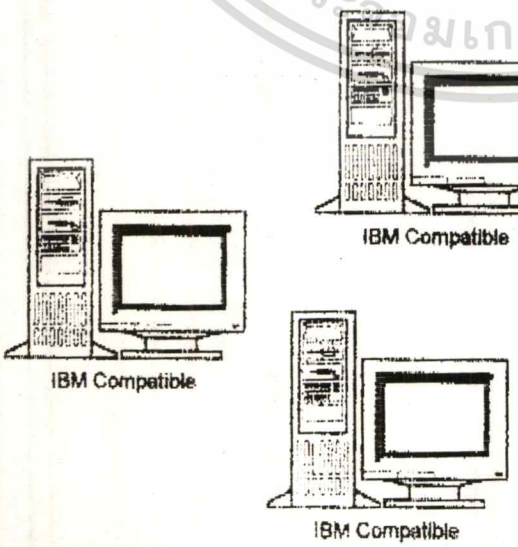
ทำการทดลองส่งแฟกซ์รูปภาพ ดังรูปที่ 4.23 และใช้โปรแกรม FAXRECV เพื่อรับข้อมูลและเพิ่ม header เป็นหมายเลข 17 จะได้ไฟล์สุดท้ายเป็น FAX00.AAA จากนั้นทำการส่งจากเครื่องให้บริการต้นทางไปยังเครื่องให้บริการปลายทาง และเครื่องให้บริการปลายทางทำการอ่านพบเบอร์โทรศัพท์ปลายทางเป็น 17 จึงทำการเรียกโปรแกรม SENDFAX และส่งข้อมูลออกไป เครื่องโทรสารที่ต่ออยู่กับหมายเลข 17 รับเอกสารแฟกซ์ได้ดังรูปที่ 4.24



รูปที่ 4.23 เอกสารแฟกซ์ที่ส่งจากเครื่องโทรสารต้นทาง

From :

P01



รูปที่ 4.24 เอกสารแฟกซ์ที่รับได้ทางเครื่องโทรสารปลายทาง

บทที่ 5
สรุปผลการทดลอง ปัญหาและแนวทางในการพัฒนา

สรุปผลการทดลอง

จากการทดลองส่งไฟล์จากผู้ใช้ต้นทางไปยังเครื่องให้บริการต้นทาง ทั้งกรณีที่กำหนดปลายทางเป็นเครื่องผู้ใช้หรือเป็นเครื่องโทรสาร พบว่าโปรแกรมบริการปลายทางสามารถวิเคราะห์ปลายทางได้อย่างถูกต้องและสามารถส่งข้อมูลไปยังเครื่องผู้ใช้ หรือเครื่องโทรสารได้อย่างถูกต้อง

จากการทดลองส่งเอกสารแฟกซ์จากเครื่องโทรสารทั้งที่เป็นเอกสารแบบข้อความและแบบรูปภาพ พบว่าสามารถใช้โปรแกรมรับข้อมูลผ่านแฟกซ์โมเด็มได้อย่างถูกต้อง พร้อมทั้งสามารถเพิ่มส่วนหัวของไฟล์และส่งไปให้เครื่องให้บริการปลายทางซึ่งทำการตรวจสอบและส่งข้อมูลไปยังเครื่องโทรสารปลายทางได้อย่างถูกต้อง

ปัญหาที่พบจากการทดลอง

1. ในการส่งข้อมูลจากเครื่องโทรสารไปยังเครื่องให้บริการ ผู้ส่งต้องระบุหมายเลขปลายทางที่ต้องการไว้ในเอกสารด้วย และเมื่อโปรแกรมบริการต้นทางรับข้อมูลมาแล้วจะต้องเปิดอ่านเพื่อเพิ่มส่วนหัวของข้อมูลเพื่อประโยชน์ในการวิเคราะห์หาปลายทางโดยโปรแกรมบริการปลายทาง
2. โปรแกรมที่สร้างขึ้นในส่วนการรับข้อมูลจากเครื่องโทรสาร เมื่อนำไปใช้กับอุปกรณ์แฟกซ์โมเด็มชุดอื่นอาจพบปัญหาไม่สามารถรับข้อมูลได้

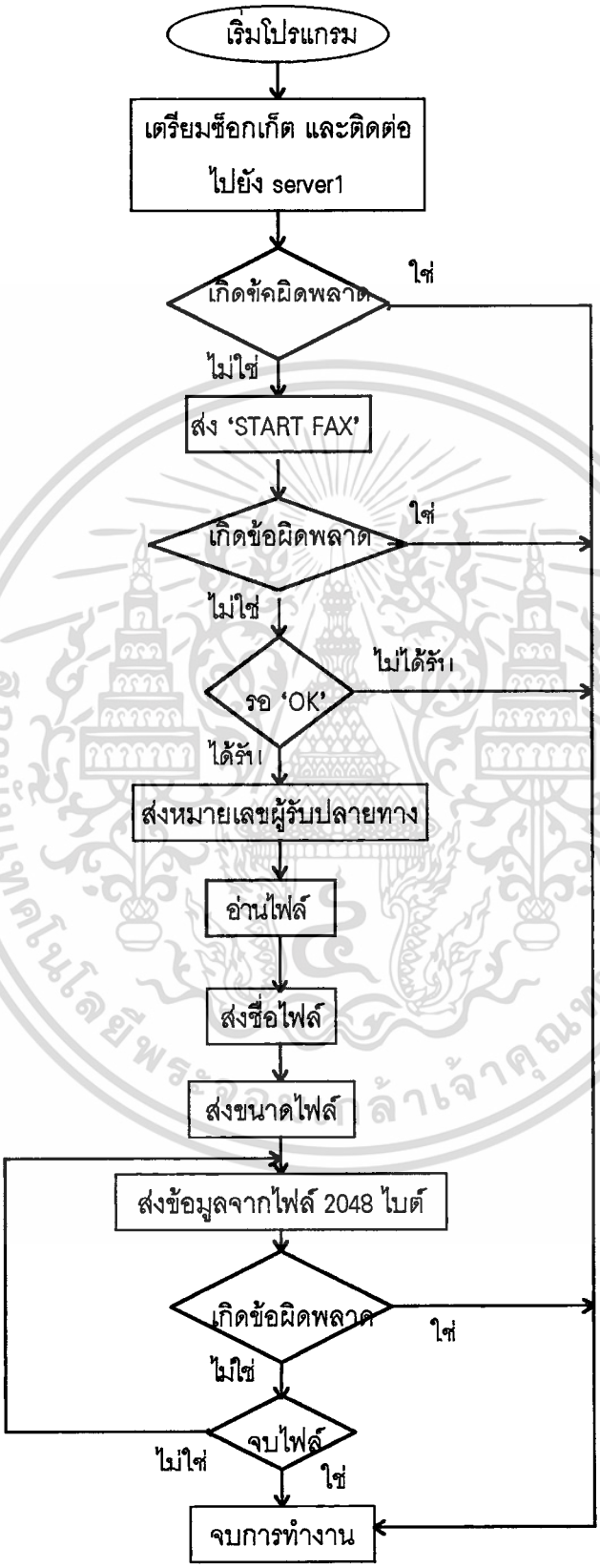
แนวทางในการพัฒนา

1. ปรับปรุงโปรแกรมเพื่อแก้ปัญหาความผิดพลาดที่เกิดขึ้น
2. พัฒนาให้โปรแกรมรับข้อมูลจากเครื่องโทรสารสามารถตรวจสอบหมายเลขปลายทางได้โดยตัวเอง ไม่ต้องอาศัยความช่วยเหลือจากผู้ควบคุมเครื่อง
3. พัฒนาให้เป็นโปรแกรมบริการที่สมบูรณ์
4. พัฒนาให้สามารถส่งเอกสารจากโปรแกรมไมโครซอฟท์เวิร์ดได้ (Microsoft Word)

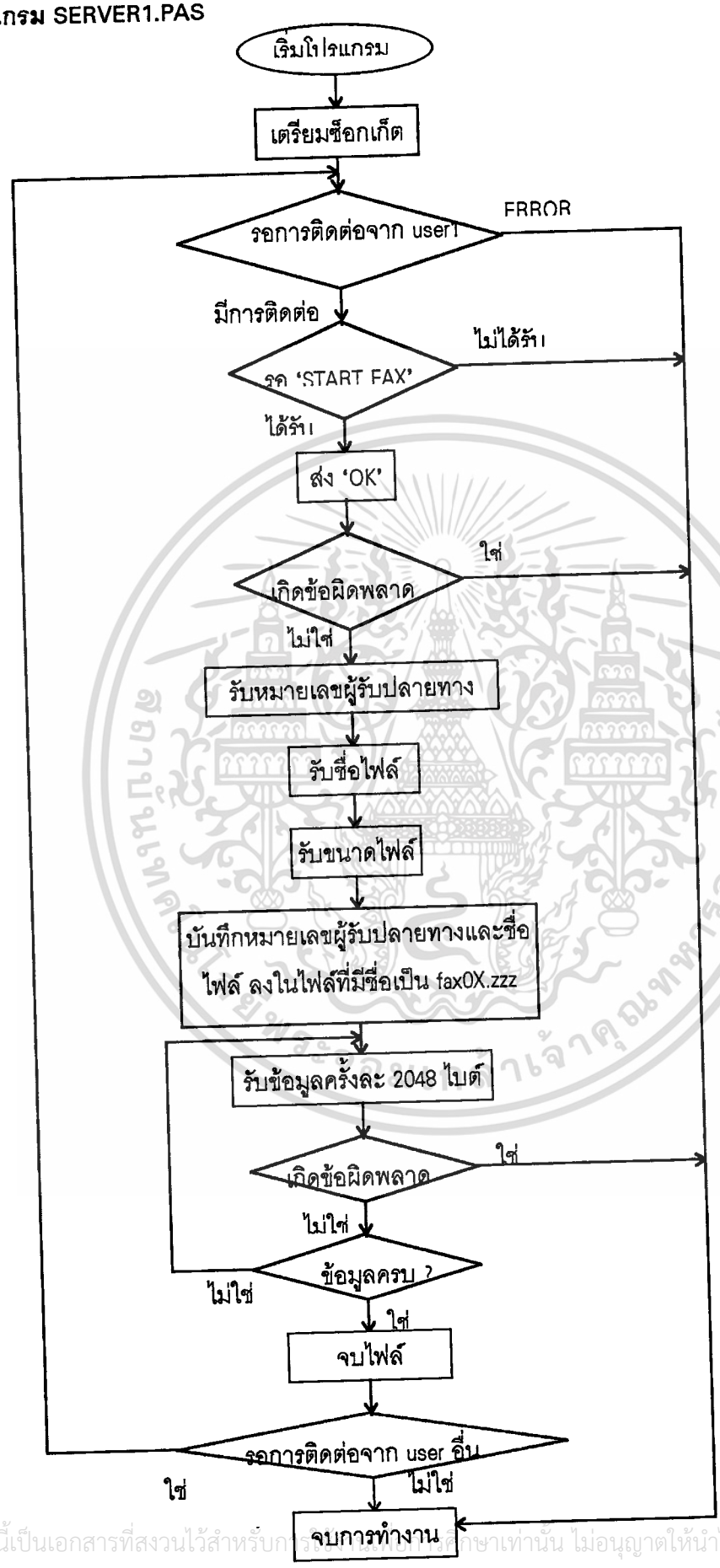
ภาคผนวก

ภาคผนวก ก

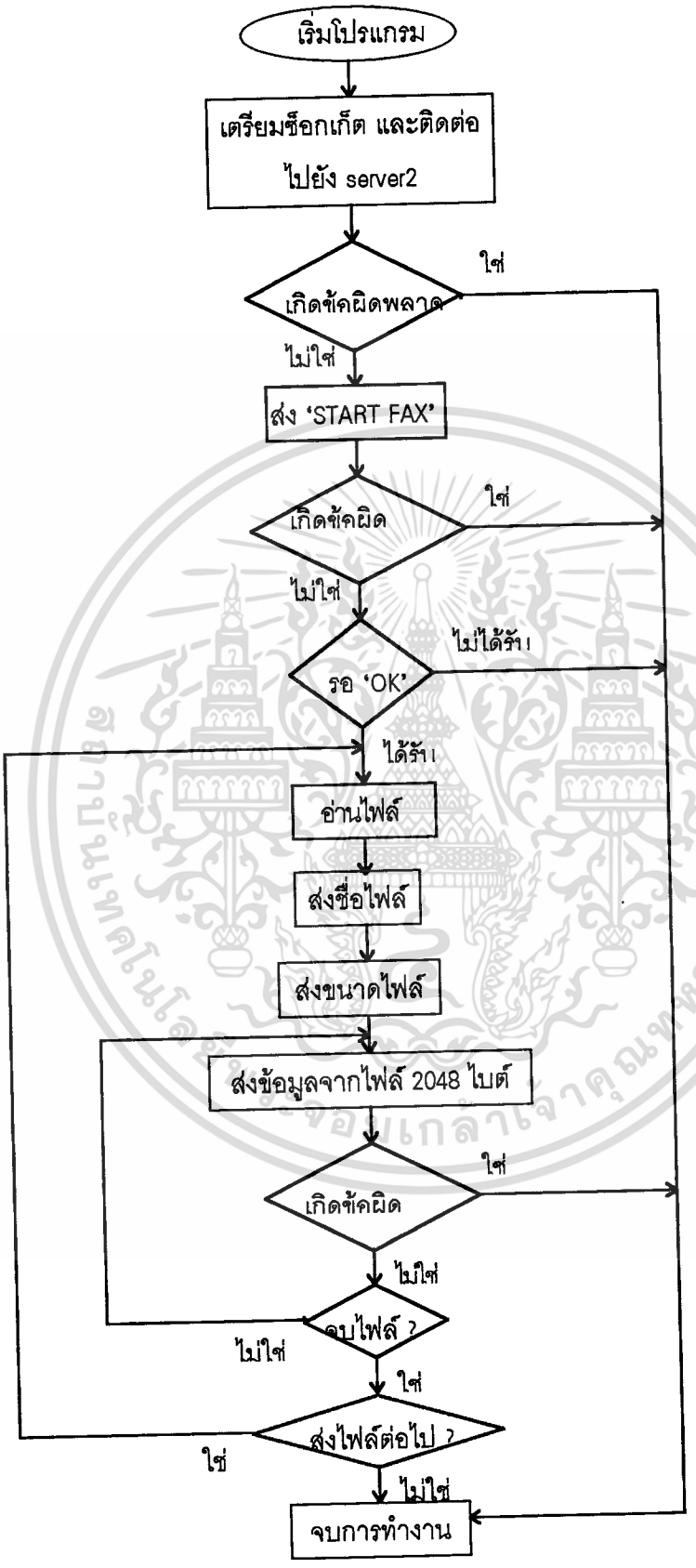
แผนผังโปรแกรม USER1.PAS



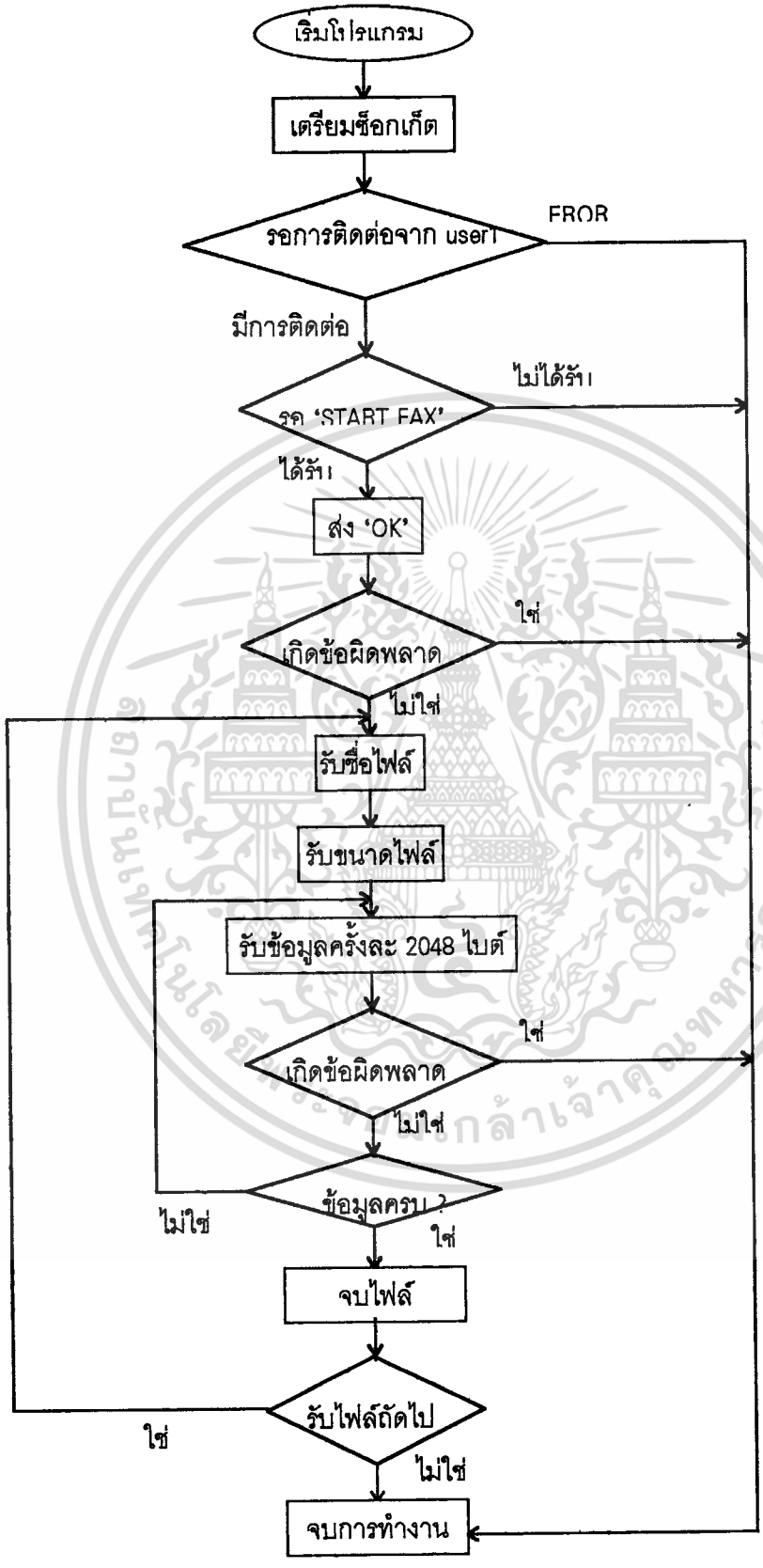
แผนผังโปรแกรม SERVER1.PAS



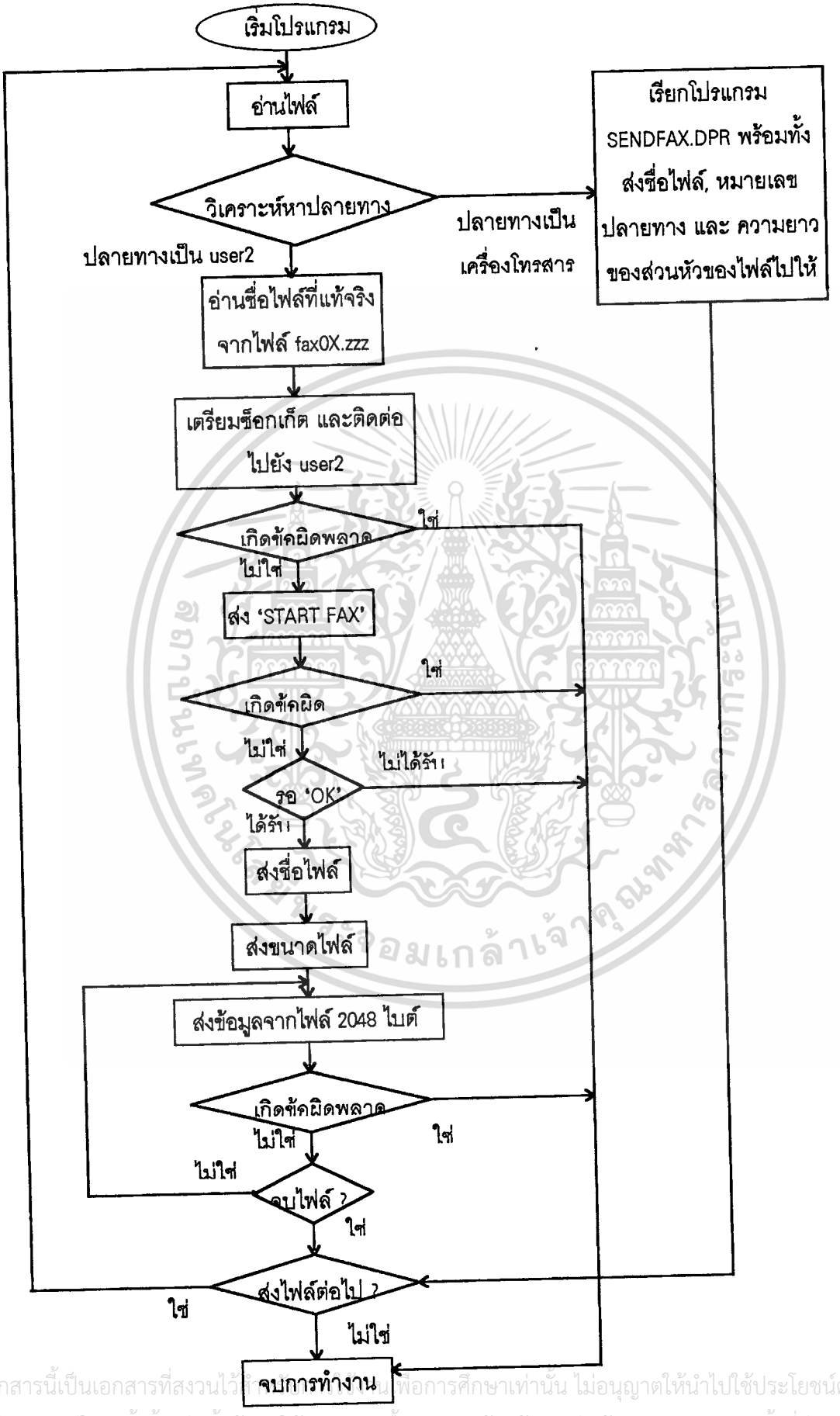
แผนผังโปรแกรม UCLIENT.PAS



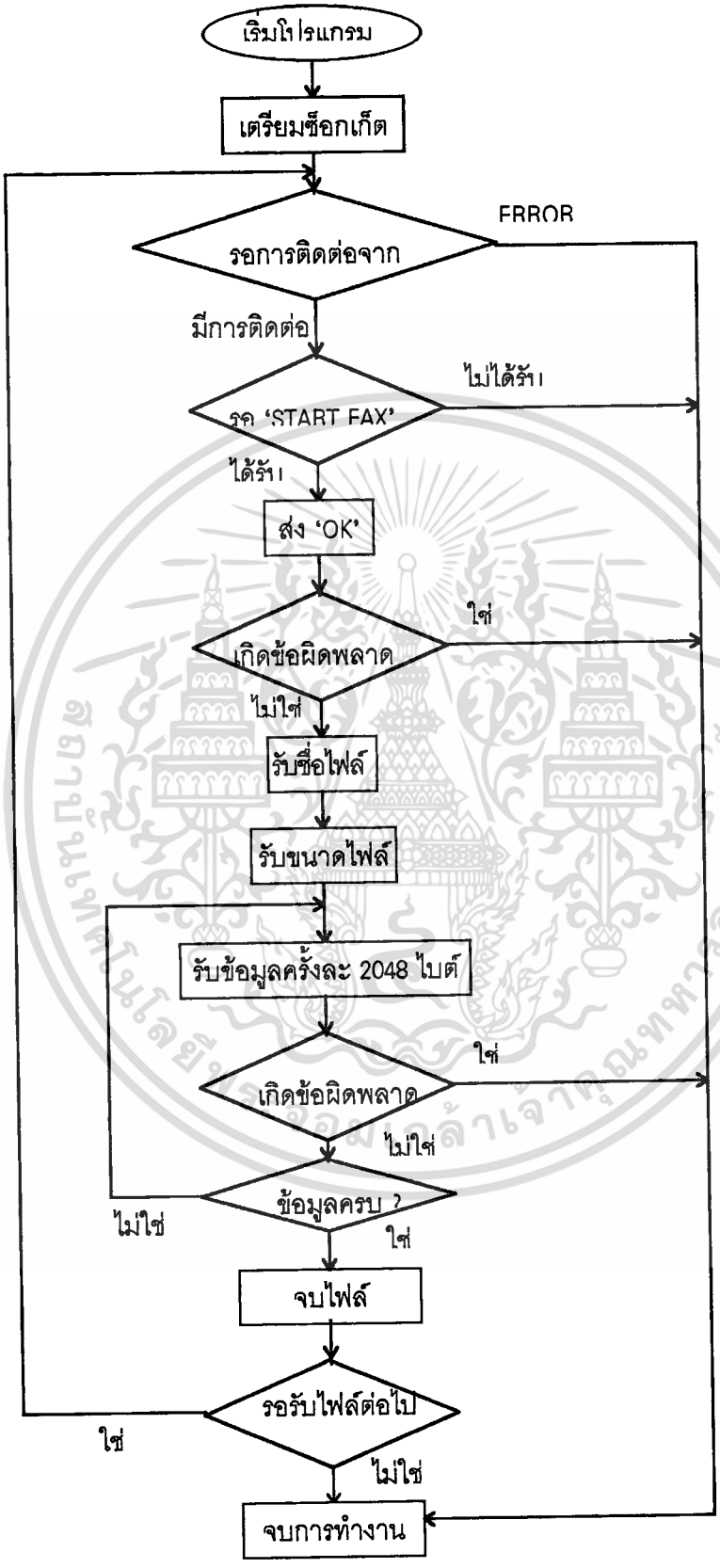
แผนผังโปรแกรม USERVER2.PAS



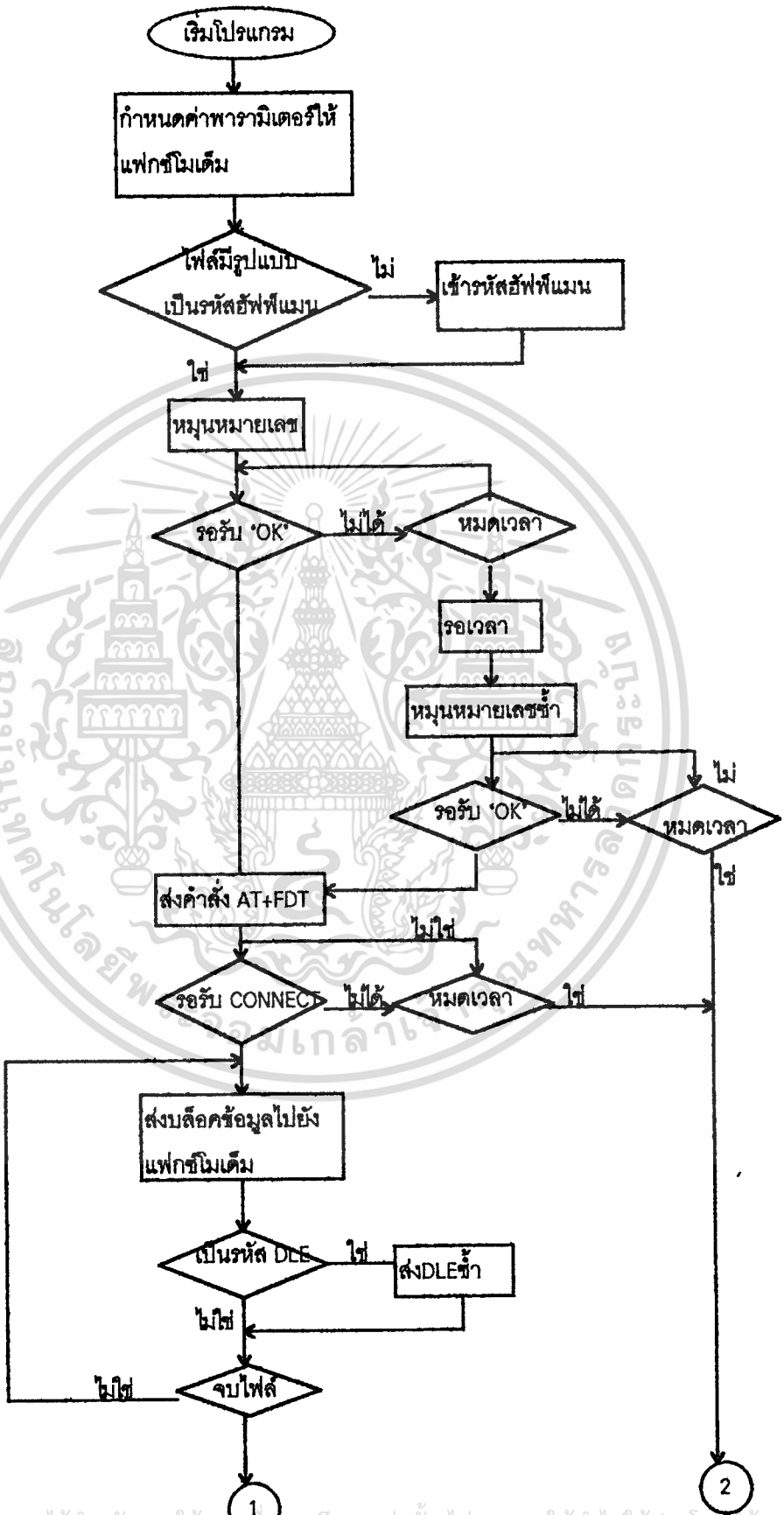
แผนผังโปรแกรม TOUSER2.PAS

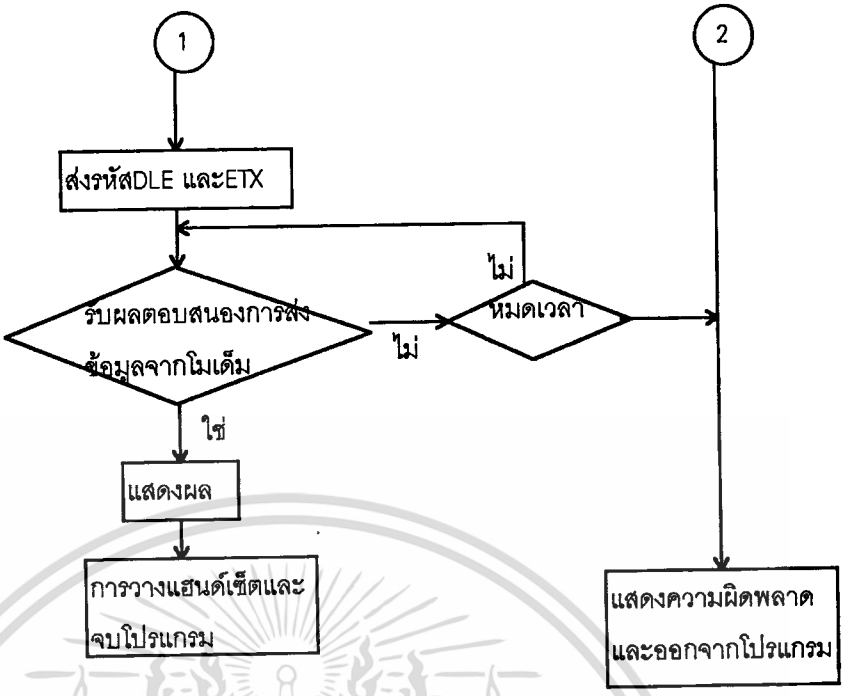


แผนผังโปรแกรม USER2.PAS

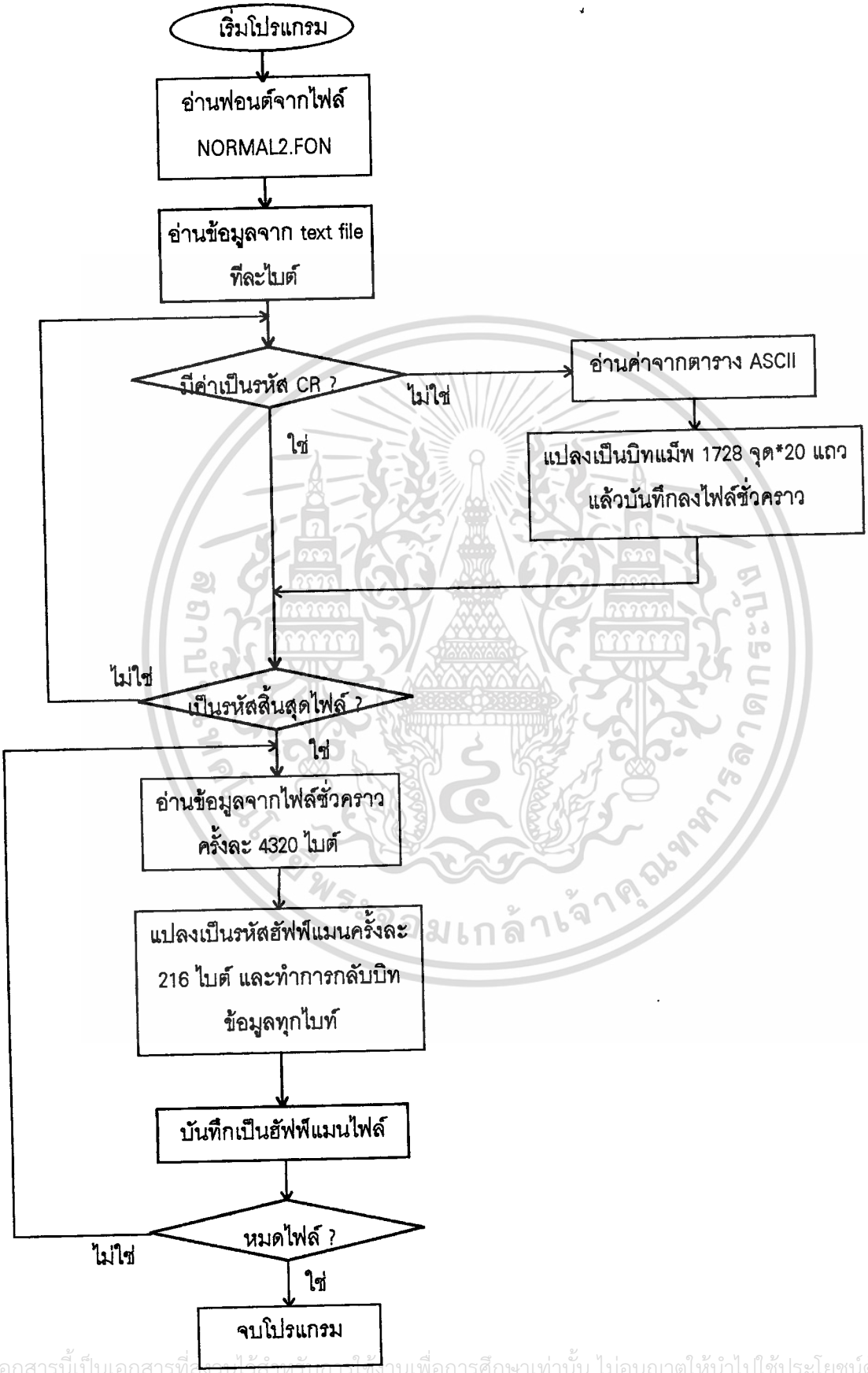


แผนผังโปรแกรม USENDFAX.PAS

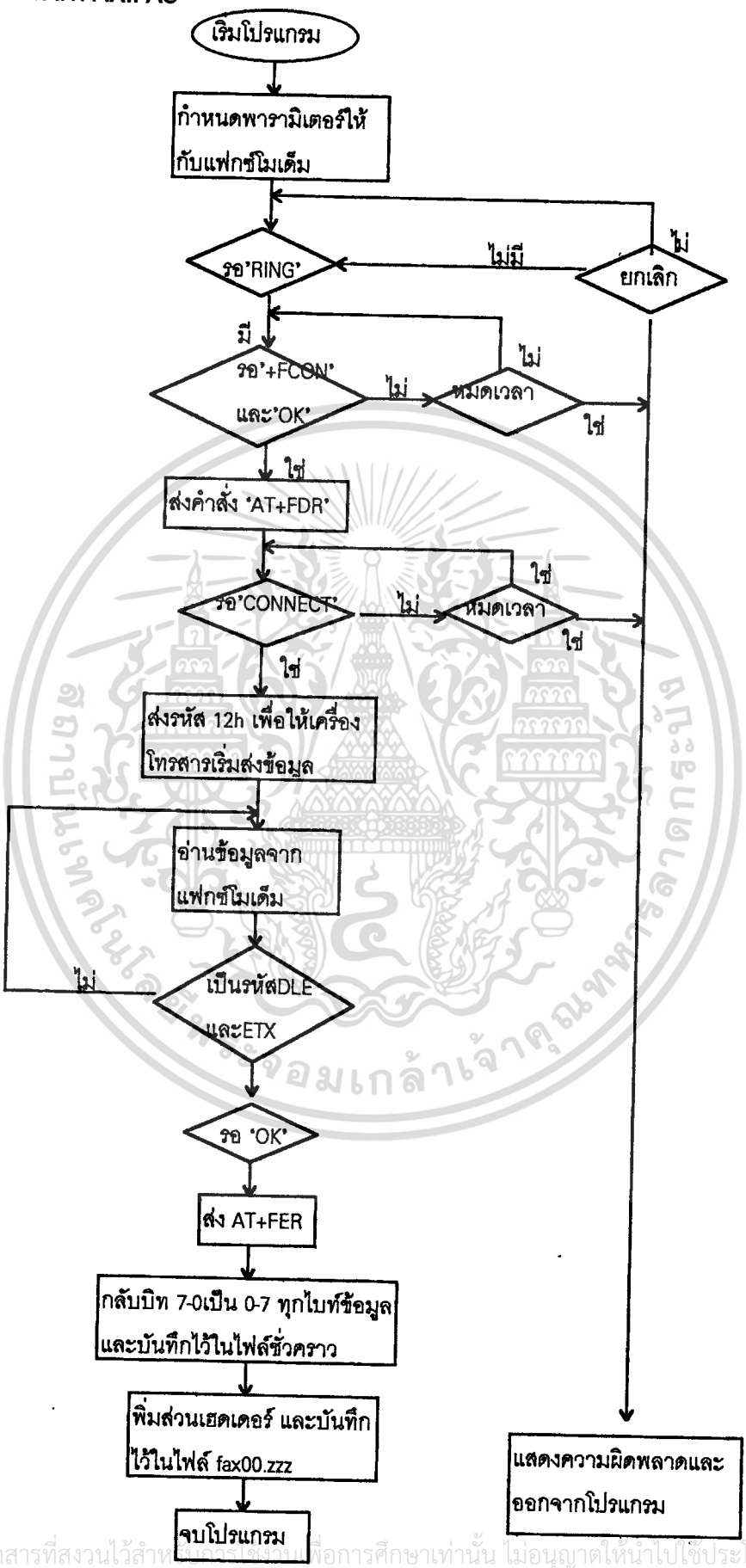




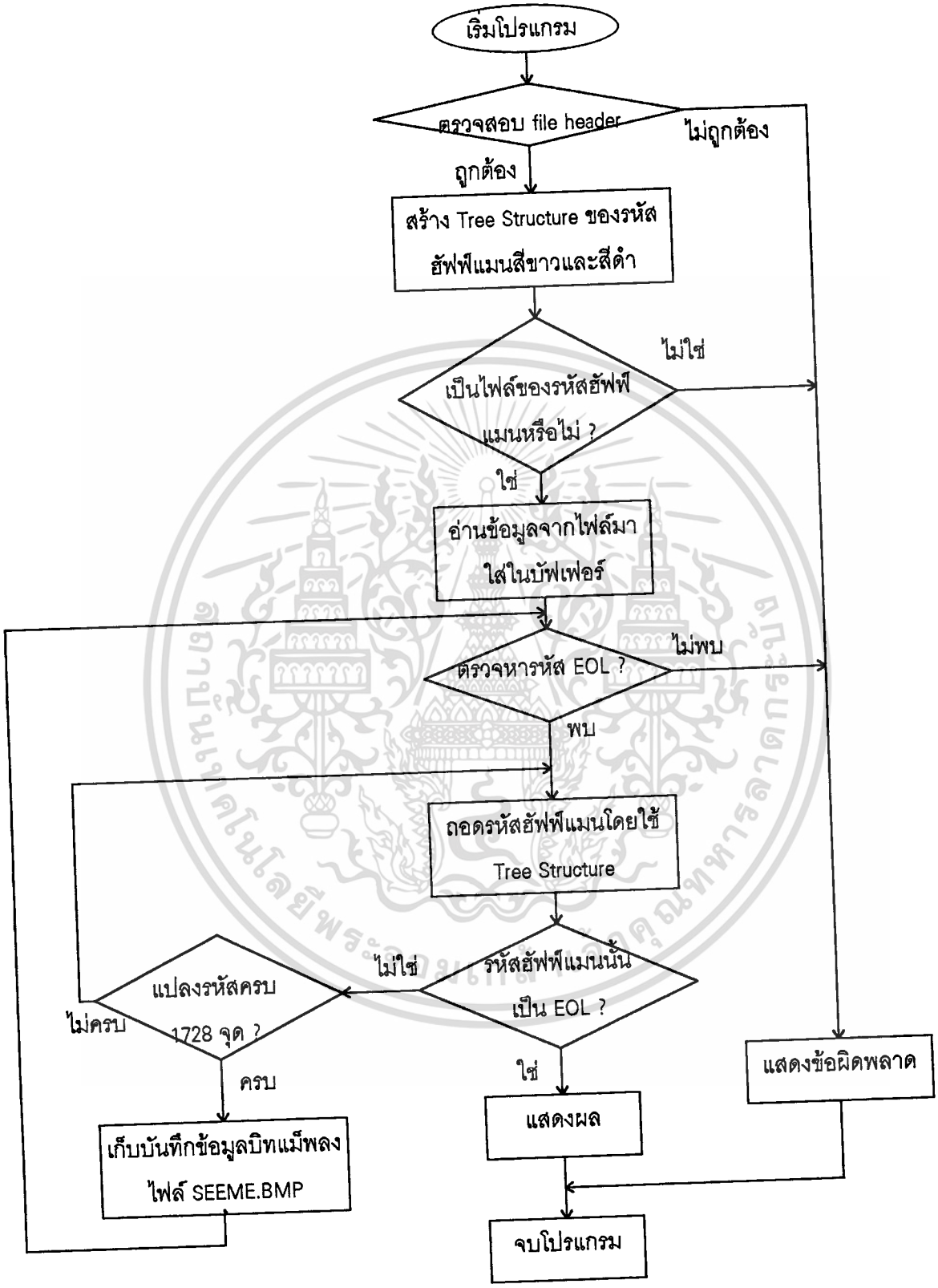
แผนผังโปรแกรม CODEFILE.PAS



แผนผังโปรแกรม UWAITFAX.PAS



แผนผังโปรแกรม VIEW.PAS



ภาคผนวก ก

รายละเอียดโปรแกรมต่างๆ

โปรแกรม FAXTASK.PAS

```
{project 'faxtool.dpr (faxtask.pas)' is the main program with many sub-
project in it. All
 sub-projects are
 - 'usersend.dpr(user1.pas)' is for user1 to send ASCII file to server1.
 - 'serve1.dpr(server1.pas)' is for server1 to receive file from user1.
 - ' ' is for server1 to receive data from fax machine.
 - 'CliFunc.dpr(uClient.pas)' is for server1 to send files to server2.
 - 'ServFunc.dpr(uListen.pas)' is for server2 to receive files from
server1.
 - ' ' is for server2 to send files to user2.
 - ' ' is for server2 to send datas to fax machine.
 - 'userecv.dpr(user2.pas)' is for user2 to receive file from server2.

 Furthermore,there are other programs contained in this project.
 - 'setting.dpr(uConfig.pas)' is used in function of fax machine.
 - about.pas
 - FuncTn1.pas
 - FuncTn2.pas
 - cookie.pas
 - cheese.pas
 - uTimer.pas
 }

unit Faxtask;

interface

uses
   SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
   Forms, Dialogs, About, Buttons, StdCtrls, User1, User2, FuncTn1,
   FuncTn2, Server1, uServer2, uClient, ToUser2;
type
  TMainForm = class(TForm)
    User1Button: TButton;
    Server1Button: TButton;
    Server2Button: TButton;
    User2Button: TButton;
    Label1: TLabel;
    AboutButton: TBitBtn;
    ExitButton: TBitBtn;
    procedure AboutButtonClick(Sender: TObject);
    procedure ExitButtonClick(Sender: TObject);
    procedure User1ButtonClick(Sender: TObject);
    procedure Server1ButtonClick(Sender: TObject);
    procedure User2ButtonClick(Sender: TObject);
    procedure Server2ButtonClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;
```

```
implementation

{$R *.DFM}

procedure TMainForm.AboutButtonClick(Sender: TObject);
begin
  AboutForm.ShowModal;
end;

procedure TMainForm.ExitButtonClick(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.User1ButtonClick(Sender: TObject);
begin
  User1Form.ShowModal;
end;

procedure TMainForm.Server1ButtonClick(Sender: TObject);
begin
  NestForm1.ShowModal;
  case NestForm1.WhatSelected of
       1 : WinExec('faxrecv.exe',sw_show);
       2 : Server1Form.ShowModal;
       3 : ClientForm.ShowModal;
         end;
end;

procedure TMainForm.User2ButtonClick(Sender: TObject);
begin
  User2Form.ShowModal;
end;

procedure TMainForm.Server2ButtonClick(Sender: TObject);
begin
  NestForm2.ShowModal;
  case NestForm2.Selected2 of
       1 : WinExec('sendfax.exe',sw_Show);
       2 : Server2Form.ShowModal;
       3 : SendToUser2Form.ShowModal;
         end;
end;

end.
```

โปรแกรม FUNCTN1.PAS

```
unit Functn1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Server1, Uclient;

type
  TNestForm1 = class(TForm)
    FaxRecv: TButton;
    UserRecv: TButton;
    Server2Send: TButton;
    procedure UserRecvClick(Sender: TObject);
    procedure FaxRecvClick(Sender: TObject);
    procedure Server2SendClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    WhatSelected : integer;
  end;

var
  NestForm1: TNestForm1;

implementation

{$R *.DFM}

procedure TNestForm1.UserRecvClick(Sender: TObject);
begin
  WhatSelected := 2;
  NestForm1.Close;
end;

procedure TNestForm1.FaxRecvClick(Sender: TObject);
begin
  WhatSelected := 1;
  NestForm1.Close;
end;

procedure TNestForm1.Server2SendClick(Sender: TObject);
begin
  WhatSelected := 3;
  NestForm1.Close;
end;

end.
```

โปรแกรม FUNCTN2.PAS

```
unit Functn2;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TNestForm2 = class(TForm)
    SendFax: TButton;
    WaitServer1: TButton;
    SendUser2: TButton;
    procedure SendFaxClick(Sender: TObject);
    procedure WaitServer1Click(Sender: TObject);
    procedure SendUser2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    Selected2 : integer;
  end;

var
  NestForm2: TNestForm2;

implementation

{$R *.DFM}

procedure TNestForm2.SendFaxClick(Sender: TObject);
begin
  Selected2 := 1;
  NestForm2.Close;
end;

procedure TNestForm2.WaitServer1Click(Sender: TObject);
begin
  Selected2 := 2;
  NestForm2.Close;
end;

procedure TNestForm2.SendUser2Click(Sender: TObject);
begin
  Selected2 := 3;
  NestForm2.Close;
end;

end.
```

โปรแกรม USER1.PAS

```
{This program is used by user1 to send file to Server1 }
unit User1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, Sockets;

const
  FaxAppPort = '2407';
type
  TUser1Form = class(TForm)
    OkClientBitBtn: TBitBtn;
    CancelClientBitBtn: TBitBtn;
    EditServerIP: TEdit;
    Label1: TLabel;
    EditFilename: TEdit;
    Label3: TLabel;
    ClientBrowseBtn: TButton;
    ClientMemo: TMemo;
    Label4: TLabel;
    OpenDialog1: TOpenDialog;
    ClientSocket: TSockets;
    GroupBox1: TGroupBox;
    RadioIP: TRadioButton;
    RadioPhone: TRadioButton;
    EditReceiver: TEdit;
    procedure OkClientBitBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure ClientBrowseBtnClick(Sender: TObject);
    procedure ClientSocketErrorOccurred(Sender: TObject; Error: Integer;
      Msg: String);
    procedure ClientSocketSessionClosed(Sender: TObject; Socket: Word);
    procedure CancelClientBitBtnClick(Sender: TObject);
  private
    { Private declarations }
    procedure initClient;
    procedure BeginSendFile;
    procedure CloseSocket;
    procedure DoClient;
    procedure SendFile;
    procedure ClientHandshake;
    function  SocketsError: Boolean;

  public
    { Public declarations }
    Timedout : Boolean;
    ErrorReturn : Integer;
  end;

var
  User1Form: TUser1Form;

implementation

{$R *.DFM}

procedure TUser1Form.OkClientBitBtnClick(Sender: TObject);
```

```
begin
  initClient;
  BeginSendFile;
end;

procedure TUser1Form.FormCreate(Sender: TObject);
begin
  EditServerIP.Text := '10.0.0.8';
  ClientMemo.Lines.Clear;
  OkClientBitBtn.Enabled := True;
  CancelClientBitBtn.Enabled := True;
  ClientBrowseBtn.Enabled := True;
end;

{ initial variables }
procedure TUser1Form.initClient;
begin
  Timedout := False;
  ErrorReturn := 0;
end;

{ set parameters of sockets, and request connection to server }
procedure TUser1Form.BeginSendFile;
begin
 ClientMemo.Lines.Clear;
 ClientSocket.IPAddr := EditServerIP.Text;
 ClientSocket.Port := FaxAppPort;
 ClientSocket.NonBlocking := False;
 ClientSocket.Timeout := 30;
 ClientMemo.Lines.Add('>> trying to connect <<');
 ClientSocket.SConnect;
 if SocketsError then
   begin
   ClientMemo.Lines.Add('..can not find server..');
   CloseSocket;
   Exit;
   end;
   DoClient;
end;

procedure TUser1Form.ClientBrowseBtnClick(Sender: TObject);
begin
  if OpenDialog1.Execute then
     EditFileName.Text := OpenDialog1.FileName
     else MessageDlg('no selected file'+#13+' Browse again!!',
          mtInformation,[mbOK],0);
end;

procedure TUser1Form.ClientSocketErrorOccurred(Sender: TObject; Error:
            Integer; Msg: String);
begin
 ErrorReturn := Error;
 if Error = WSAETIMEDOUT then
    begin
    Timedout := True;
    ErrorReturn := 0;
    end;
 MessageDlg('@'+IntToStr(Error)+':'+Msg, mtError,[mbOk],0);
 if ClientSocket.SocketNumber <> INVALID_SOCKET then
    begin
    ClientMemo.Lines.Add('<< close socket >>');
    CloseSocket;
```

```
    end;
end;

function TUser1Form.SocketsError: Boolean;
begin
 Result := False;
 if Timedout or (ErrorReturn <> 0) then
   begin
   Result := True;
   ErrorReturn := 0;
   MessageDlg( 'Time out or socket error', mtError,[mbOk],0);
   end;
end;

procedure TUser1Form.CloseSocket;
begin
  if ClientSocket.SocketNumber <> INVALID_SOCKET then
     ClientSocket.SClose;
  OkClientBitBtn.Enabled := True;
end;

procedure TUser1Form.ClientSocketSessionClosed(Sender: TObject; Socket:
                                                    Word);
begin
  ClientMemo.Lines.Add('>> socket closed <<');
  CloseSocket;
end;

procedure TUser1Form.DoClient;
begin
 ClientMemo.Lines.Add('>> connected <<');
 OkClientBitBtn.Enabled := False;
 CancelClientBitBtn.Enabled := True;
 ClientBrowseBtn.Enabled := False;

 ClientHandshake;
 if SocketsError then
   begin
   ClientMemo.Lines.Add('>> socket error <<');
   closeSocket;
   Exit;
   end;

 SendFile;
 ClientMemo.Lines.Add('..finished..');

 OkClientBitBtn.Enabled := True;
 CancelClientBitBtn.Enabled := True;
 ClientBrowseBtn.Enabled := True;
end;

{ after connecting and handshaking is complete, begin send data to
  receiver }
procedure TUser1Form.SendFile;
var
 f: File;
 numread: Integer;
 fsize, bytewrite: LongInt;
 FileToSave,FileToSend,FaxIP,Phone:String;
 str1: String;
 szBuf: array[0..2047] of Char;
begin
```

```
  { dependent on user's setting, where is the destination of data }
  if RadioIP.Checked then
     FaxIP := EditReceiver.Text
     else Phone := EditReceiver.Text;

  FileToSend := EditFileName.Text;  { set filename to send }
  AssignFile(f, FileToSend);        { assign f for use as FileToSend }
  {$I-}                             { ignore I/O error repoet }
  Reset(f,1);                       { open file to read }
  {$I+}                             { re-enable I/O error report }
  if IOResult <> 0 then
    { if I/O error occur then report to user and exit program }
    begin
    ClientMemo.Lines.Add('..can not open file..');
    CloseSocket;
    Exit;
    end;

  ClientMemo.Lines.Add('..sending..');
  fsize := FileSize(f);       { get file's size to send }
  bytewrite := 0;             { number of data (bytes) is sent }

  if RadioIP.Checked then
     ClientSocket.Text := '('+FaxIP+')'
     else ClientSocket.Text := '('+Phone+')';
  ClientSocket.Text := '('+FileToSend+')';      { send (file name) }
  ClientSocket.Text := '('+IntToStr(fsize)+')'; { send (text of filesize)}
  { while not End of file, repeat send block of data (2048 bytes) }
  while not Eof(f) and not SocketsError do
    begin
    BlockRead(f, szBuf, sizeof(szBuf), numread);
    ClientSocket.SSend(ClientSocket.SocketNumber,szBuf,numread);
    bytewrite := bytewrite+numread;
    end;
  if SocketsError then
     begin
     ClientMemo.Lines.Add('..error while sending..');
     CloseSocket;
     end;

  ClientMemo.Lines.Add('..end of file..');
  CloseFile(f);
end;

{ interaction between Client and Server to make sure that is the same
   application }
procedure TUser1Form.ClientHandshake;
var
 str: String;
begin
 ClientSocket.Text := 'START FAX';   { send text 'START FAX' }
 if SocketsError then
   begin
   ClientMemo.Lines.Add('..error..');
   Exit;
   end;

 str := ClientSocket.Text;            { receive String from client }
 if SocketsError then
   begin
   ClientMemo.Lines.Add('..error..');
```

```
    Exit;
    end;
  ClientMemo.Lines.Add('>'+str+'<');

  if Pos('OK', str) = 0 then        { find 'OK' in String }
    begin
    ClientMemo.Lines.Add('..invalid reply..');
    ErrorReturn := -1;
    Exit;
    end;
end;

procedure TUser1Form.CancelClientBitBtnClick(Sender: TObject);
begin
  { stop sending, cancel parameter setting and back to main form }
  if ClientSocket.SocketNumber <> INVALID_SOCKET then
     ClientSocket.SClose;
  ClientMemo.Lines.Add('..server stoped..');

  initClient;
  Close;
end;

end.
```

โปรแกรม SERVER1.PAS

```
{ This program is used by server1 to receive files from user1's }
unit Server1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, Sockets;

const
  FaxAppPort = '2407';

type
  TServer1Form = class(TForm)
    EditServerDir: TEdit;
    Label1: TLabel;
    ServerMemo: TMemo;
    Label2: TLabel;
    OkServBitBtn: TBitBtn;
    CancelServBitBtn: TBitBtn;
    ServerSocket: TSockets;
    procedure OkServBitBtnClick(Sender: TObject);
    procedure CancelServBitBtnClick(Sender: TObject);
    procedure ServerSocketErrorOccurred(Sender: TObject; Error: Integer;
      Msg: String);
    procedure ServerSocketSessionClosed(Sender: TObject; Socket: Word);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    procedure initServer;
    procedure BeginListening;
    procedure DoServer;
    procedure CloseSocket;
    procedure GetFile;
    procedure ServerHandShake;
    function GetFaxIP: String;
    function GetSavedFileName: String;
    function GetSavedFileSize: LongInt;
    function SocketsError: Boolean;
  public
    { Public declarations }
    Timedout : Boolean;
    ErrorReturn : Integer;
    IPlen : integer;
  end;

var
  Server1Form: TServer1Form;
  count : integer;
implementation

{$R *.DFM}

procedure TServer1Form.FormCreate(Sender: TObject);
begin
  EditServerDir.Text := 'c:\faxsend';
  ServerMemo.Lines.Clear;
  OkServBitBtn.Enabled := True;
  CancelServBitBtn.Enabled := True;
```

```
end;

procedure TServer1Form.OkServBitBtnClick(Sender: TObject);
begin
  initServer;
  BeginListening;
end;

procedure TServer1Form.initServer;
begin
  ErrorReturn := 0;
  Timedout := False;
end;

{ set parameters of server's socket before waiting request from client }
procedure TServer1Form.BeginListening;
begin
  ServerSocket.IPAddr := '';
  ServerSocket.Port := FaxAppPort;
  ServerSocket.NonBlocking := False;
  ServerSocket.Timeout := 0;

  ServerMemo.Lines.Add('..waiting..');
  ServerSocket.SListen;

 if SocketsError then
   begin
   ServerMemo.Lines.Add('..no connection..');
   ServerSocket.SCancelListen;
   CloseSocket;
   Exit;
   end;

 count := 0;
 while true do
 begin
 DoServer;   { receive 1 file each time }
 if MessageDlg('waiting next client',mtConfirmation,[mbYES,mbNO],0) =
    mrNo then   break;
 inc(count);
 ServerMemo.Lines.Add('..waiting next user..');
 end;      { end of while }
 ServerMemo.Lines.Add('..'+IntToStr(count+1)+' client connected..');
end;

procedure TServer1Form.ServerSocketSessionClosed(Sender: TObject; Socket:
          Word);
begin
  ServerMemo.Lines.Add('..socket closed..');
  CloseSocket;
end;

procedure TServer1Form.ServerSocketErrorOccurred(Sender: TObject; Error:
          Integer; Msg: String);
begin
 ErrorReturn := Error;
 if Error = WSAETIMEDOUT then
    begin
    Timedout := True;
    ErrorReturn := 0;
    end;
 MessageDlg('@'+IntToStr(Error)+':'+Msg, mtError,[mbOk],0);
```

```
 if ServerSocket.SocketNumber <> INVALID_SOCKET then
    begin
    ServerMemo.Lines.Add('<< close socket >>');
    CloseSocket;
    end;
end;

function TServer1Form.SocketsError: Boolean;
begin
 Result := False;
 if Timedout or (ErrorReturn <> 0) then
   begin
   Result := True;
   ErrorReturn := 0;
   MessageDlg( 'Time out or socket error', mtError,[mbOk],0);
   end;
end;

procedure TServer1Form.CloseSocket;
begin
  if ServerSocket.SocketNumber <> INVALID_SOCKET then
     begin
     ServerSocket.SClose;
     end;
  OkServBitBtn.Enabled := True;
end;

procedure TServer1Form.CancelServBitBtnClick(Sender: TObject);
begin
  { stop listening, cancel parameter setting and back to main form }
  if ServerSocket.SocketNumber <> INVALID_SOCKET then
     ServerSocket.SClose;
  ServerMemo.Lines.Add('..server1 stoped..');

  initServer;  { restore old value parameter }
  Close;
end;

procedure TServer1Form.DoServer;
begin
 ServerMemo.Lines.Add('');
 ServerSocket.SAccept;
 ServerMemo.Lines.Add('..connected..');

 ServerHandshake;
 if SocketsError then
   begin
   CloseSocket;
   Exit;
   end;

 GetFile;
 ServerMemo.Lines.Add('..finished..');
end;

procedure TServer1Form.GetFile;
var    f: File;
       numread: Integer;
       fsize, byteleft: LongInt;
       FileToSave,FileToSend,FaxToSend,FaxHeader:String;
       str1: String;
       szBuf: array[0..2047] of Char;
```

```
egin
 FaxToSend := GetFaxIP;
 if Pos('.',FaxToSend) = 0 then
    FaxHeader := 'Fax Phone No '+ '('+FaxToSend+ ')' +#13#10
    else  FaxHeader := 'Destination Fax IP '+'('+FaxToSend+')'+#13#10;
 FileToSave := GetSavedFileName;    { get file name from client }
 fsize:= GetSavedFileSize;          { get file size from client }

 ServerMemo.Lines.Add('>'+ FaxToSend +'<');
 ServerMemo.Lines.Add('>'+ FileToSave +'<'); { show received file name }
 FileToSave := 'File Name ('+ExTractFileName(FileToSave)+')'+#13#10;
 ServerMemo.Lines.Add('>'+ IntToStr(fsize) +'<'); { show received file
                                                    size }


 { assign f for file to save (open file to write }
 AssignFile(f, EditServerDir.Text+'\fax0'+IntToStr(count)+'.zzz');
 {$I-}  Rewrite(f,1); {$I+}
 if IOResult <> 0 then
    begin
    ServerMemo.Lines.Add('..error opening file..');
    Exit;
    end;

  BlockWrite(f,FaxHeader,length(FaxHeader)+1);
  BlockWrite(f,FileToSave,length(FileToSave)+1);
  byteleft := fsize;          { number of data to receive }
  while (byteleft > 0) and (not SocketsError) do
    begin
    numread := sizeof(szBuf); { each time to read is 2048 bytes }
    numread :=
         ServerSocket.SReceive(ServerSocket.SocketNumber,szBuf,numread);
    if not SocketsError then
       begin
       BlockWrite(f,szBuf,numread);
       byteleft := byteleft - numread;
       end
       else
        begin
        ServerMemo.Lines.Add('..error while receiving..');
        ServerSocket.SClose;
        Exit;
        end;
    end;    { end of while }
  CloseFile(f);
  CloseSocket;
 end;

 { get destination number, may be Ipaddress or Fax Phone number }
 function TServer1Form.GetFaxIP: String;
 var
   Buf: String;
   szBuf: array[0..255] of char absolute Buf;
   idx,len: integer;
 begin
   Buf := '';
   Buf := ServerSocket.Peek;
   if SocketsError then
      Exit;
   idx := pos(')',Buf);
   if idx <= 0 then
      Exit;
   len := idx;
```

```
   ServerSocket.SReceive(ServerSocket.SocketNumber,@szBuf[1],len);
   if SocketsError then
      Exit;
   szBuf[0] := chr(len);
   IPlen := len;
   Result := Copy(Buf,2,length(Buf)-2);

end;

{ get file's name is sent in parenthis from client }
function TServer1Form.GetSavedFileName: String;
var
 Buf: String;
 szBuf: array[0..255] of char absolute Buf;
 idx, len: integer;
begin
 Buf := '';
 Buf := ServerSocket.Peek;
 if SocketsError then
    Exit;
 idx := pos(')', Buf);
 if idx <= 0 then
   Exit;
 len := idx;
 ServerSocket.SReceive(ServerSocket.SocketNumber,@szBuf[1], len);
 if SocketsError then
   Exit;
 szBuf[0] := chr(len);
 Result := Copy(Buf, 2, length(Buf)-2);

end;

{ get size of file is sent in parenthis from client }
function TServer1Form.GetSavedFileSize: LongInt;
var
 Buf: String;
 szBuf: array[0..255] of char absolute Buf;
 ch: char;
 idx, len: integer;
begin
 Buf := '';
 Buf := ServerSocket.Peek;
 ch := #0;
 if SocketsError then
   Exit;
 idx := pos(')', Buf);
 if idx <= 0 then
   Exit;
 len := idx;
 ServerSocket.SReceive(ServerSocket.SocketNumber,@szBuf[1], len);

{ receive array of char and change into String }
 if SocketsError then
   Exit;
 szBuf[0] := chr(len);
 Result := StrToInt(Copy(Buf, 2, length(Buf)-2));

end;

{ interaction to procedure'ClientHandshake' of sender's program }
procedure TServer1Form.ServerHandshake;
var   str: String;
```

```
begin
  str := ServerSocket.Text;              { receive String from client }
  ServerMemo.Lines.Add('>'+str+'<');
  if Pos('START FAX', str) = 0 then      { find 'START FAX' in String }
     begin
     ServerMemo.Lines.Add('..invalid header..');
     ErrorReturn := -1;
     Exit;
     end;

  ServerSocket.Text := 'OK';             { send 'OK' to client }
end;

end.
```

โปรแกรม UCLIENT.PAS

```
{ This program is used by server1 to send files to server2 }
unit Uclient;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, Sockets, Server1;

const
  FaxAppPort = '2407';
type
  TClientForm = class(TForm)
    OkClientBitBtn: TBitBtn;
    CancelClientBitBtn: TBitBtn;
    EditServerIP: TEdit;
    Label1: TLabel;
    ClientMemo: TMemo;
    Label4: TLabel;
    ClientSocket: TSockets;
    EditDir: TEdit;
    Label2: TLabel;
    procedure OkClientBitBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure ClientSocketErrorOccurred(Sender: TObject; Error: Integer;
      Msg: String);
    procedure ClientSocketSessionClosed(Sender: TObject; Socket: Word);
    procedure CancelClientBitBtnClick(Sender: TObject);
  private
    { Private declarations }
    procedure initClient;
    procedure BeginSendFile;
    procedure CloseSocket;
    procedure DoClient;
    procedure SendFile;
    procedure SendFaxFile;
    procedure ClientHandshake;
    function  SocketsError: Boolean;

  public
    { Public declarations }
    Timedout : Boolean;
    ErrorReturn : Integer;

  end;

var
  ClientForm: TClientForm;

implementation

{$R *.DFM}

procedure TClientForm.OkClientBitBtnClick(Sender: TObject);
begin
  initClient;
  BeginSendFile;
end;
```

```
procedure TClientForm.FormCreate(Sender: TObject);
begin
  EditServerIP.Text := '10.0.0.8';
  ClientMemo.Lines.Clear;
  OkClientBitBtn.Enabled := True;
  CancelClientBitBtn.Enabled := True;
  EditDir.Text := Server1Form.EditServerDir.Text;
end;

procedure TClientForm.initClient;
begin
  Timedout := False;
  ErrorReturn := 0;
end;

procedure TClientForm.BeginSendFile;
begin
 ClientMemo.Lines.Clear;
 ClientSocket.IPAddr := EditServerIP.Text;
 ClientSocket.Port := FaxAppPort;
 ClientSocket.NonBlocking := False;
 ClientSocket.Timeout := 30;
 ClientMemo.Lines.Add('>> trying to connect <<');
 ClientSocket.SConnect;
 if SocketsError then
   begin
   ClientMemo.Lines.Add('..can not find server..');
   CloseSocket;
   Exit;
   end;
 DoClient;
end;

procedure TClientForm.ClientSocketErrorOccurred(Sender: TObject; Error:
            Integer; Msg: String);
begin
 ErrorReturn := Error;
 if Error = WSAETIMEDOUT then
    begin
    Timedout := True;
    ErrorReturn := 0;
    end;
 MessageDlg('@'+IntToStr(Error)+':'+Msg, mtError,[mbOk],0);
 if ClientSocket.SocketNumber <> INVALID_SOCKET then
    begin
    ClientMemo.Lines.Add('<< close socket >>');
    CloseSocket;
    end;
end;

function TClientForm.SocketsError: Boolean;
begin
 Result := False;
 if Timedout or (ErrorReturn <> 0) then
   begin
   Result := True;
   ErrorReturn := 0;
   MessageDlg( 'Time out or socket error', mtError,[mbOk],0);
   end;
end;

procedure TClientForm.CloseSocket;
```

```
begin
  if ClientSocket.SocketNumber <> INVALID_SOCKET then
     begin
     ClientSocket.SClose;
     end;
  OkClientBitBtn.Enabled := True;
end;


procedure TClientForm.ClientSocketSessionClosed(Sender: Tobject;
            Socket: Word);
begin
  ClientMemo.Lines.Add('>> socket closed <<');
  CloseSocket;
end;

procedure TClientForm.DoClient;
begin
 ClientMemo.Lines.Add('>> connected <<');
 OkClientBitBtn.Enabled := False;
 CancelClientBitBtn.Enabled := True;

 ClientHandshake;
 if SocketsError then
   begin
   ClientMemo.Lines.Add('>> socket error <<');
   closeSocket;
   Exit;
   end;

 SendFile;
 ClientMemo.Lines.Add('..finished..zzz');

 SendFaxFile;
 ClientMemo.Lines.Add('..finished..aaa');

 OkClientBitBtn.Enabled := True;
 CancelClientBitBtn.Enabled := True;
end;

{ send file 'fax0X.zzz' until user cancel next file }
procedure TClientForm.SendFile;
var
 f: File;
 numread: Integer;
 fsize, bytewrite: LongInt;
 FileToSave,FileToSend : String;
 str1: String;
 szBuf: array[0..2047] of Char;
 count :integer;
 Dir : String;
begin
count := 0;
Dir := EditDir.Text;
while true do
 begin
 FileToSend := Dir+'\fax0'+IntToStr(count)+'.zzz';
 AssignFile(f, FileToSend);
 {$I-} Reset(f,1); {$I+}
 if IOResult <> 0 then
   ClientMemo.Lines.Add('can not open file '+FileToSend)
   else
```

```
    begin
    ClientMemo.Lines.Add('sending '+FileToSend);
    fsize := FileSize(f);
    bytewrite := 0;

    ClientSocket.Text := '('+FileToSend+')';
    ClientSocket.Text := '('+IntToStr(fsize)+')';
    while not Eof(f) and not SocketsError do
      begin
      BlockRead(f, szBuf, sizeof(szBuf), numread);
      ClientSocket.SSend(ClientSocket.SocketNumber,szBuf,numread);
      bytewrite := bytewrite+numread;
      end;
    if SocketsError then
       begin
       ClientMemo.Lines.Add('..error while sending..');
       CloseSocket;
       end;

    ClientMemo.Lines.Add('..end of file..');
    CloseFile(f);
    DeleteFile(FileToSend);
    end;
 if MessageDlg('send next',mtConfirmation,[mbYES,mbNO],0) = mrNo then
    break;
 inc(count);
 end;
 ClientMemo.Lines.Add('..'+IntToStr(count+1)+' files are sent');

end;
{ send file 'fax0X.aaa' until user cancel next file }
procedure TClientForm.SendFaxFile;
var
 f: File;
 numread: Integer;
 fsize, bytewrite: LongInt;
 FileToSave,FileToSend : String;
 str1: String;
 szBuf: array[0..2047] of Char;
 count :integer;
 Dir : String;
begin
count := 0;
Dir := EditDir.Text;

{begin read file '.aaa'}
 count := 0;
 while true do
 begin
 FileToSend := Dir+'\fax0'+IntToStr(count)+'.aaa';
 AssignFile(f, FileToSend);
 {$I-}  Reset(f,1);   {$I+}
 if IOResult <> 0 then
   ClientMemo.Lines.Add('can not open file '+FileToSend)
   else
    begin
    ClientMemo.Lines.Add('sending '+FileToSend);
    fsize := FileSize(f);
    bytewrite := 0;

    ClientSocket.Text := '('+FileToSend+')';
    ClientSocket.Text := '('+IntToStr(fsize)+')';
    while not Eof(f) and not SocketsError do
```

```
      begin
      BlockRead(f, szBuf, sizeof(szBuf), numread);
      ClientSocket.SSend(ClientSocket.SocketNumber,szBuf,numread);
      bytewrite := bytewrite+numread;
      end;
    if SocketsError then
       begin
       ClientMemo.Lines.Add('..error while sending..');
       CloseSocket;
       end;

    ClientMemo.Lines.Add('..end of file..');
    CloseFile(f);
    DeleteFile(FileToSend);
    end;
 if MessageDlg('send next',mtConfirmation,[mbYES,mbNO],0) = mrNo then
    break;
 inc(count);
 end;
 ClientMemo.Lines.Add('..'+IntToStr(count+1)+' files are sent');

end;

procedure TClientForm.ClientHandshake;
var
 str: String;
begin
 ClientSocket.Text := 'START FAX';
 if SocketsError then
   begin
   ClientMemo.Lines.Add('..error..');
   Exit;
   end;

 str := ClientSocket.Text;
 if SocketsError then
   begin
   ClientMemo.Lines.Add('..error..');
   Exit;
   end;
 ClientMemo.Lines.Add('>'+str+'<');

 if Pos('OK', str) = 0 then
   begin
   ClientMemo.Lines.Add('..invalid reply..');
   ErrorReturn := -1;
   Exit;
   end;
end;

procedure TClientForm.CancelClientBitBtnClick(Sender: TObject);
begin
  if ClientSocket.SocketNumber <> INVALID_SOCKET then
     ClientSocket.SClose;
  ClientMemo.Lines.Add('..server stoped..');

  initClient;
  Close;
end;

end.
```

โปรแกรม USERVER2.PAS

```
unit Userver2;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, Sockets;

const
  FaxAppPort = '2407';

type
  TServer2Form = class(TForm)
    Server2Dir: TEdit;
    Label1: TLabel;
    ServerMemo: TMemo;
    Label2: TLabel;
    OkServBitBtn: TBitBtn;
    CancelServBitBtn: TBitBtn;
    ServerSocket: TSockets;
    procedure OkServBitBtnClick(Sender: TObject);
    procedure CancelServBitBtnClick(Sender: TObject);
    procedure ServerSocketErrorOccurred(Sender: TObject; Error: Integer;
      Msg: String);
    procedure ServerSocketSessionClosed(Sender: TObject; Socket: Word);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    procedure initServer;
    procedure BeginListening;
    procedure DoServer;
    procedure CloseSocket;
    procedure GetFile;
    procedure ServerHandShake;
    function GetSavedFileName: String;
    function GetSavedFileSize: LongInt;
    function SocketsError: Boolean;

  public
    { Public declarations }
    Timedout : Boolean;
    ErrorReturn : Integer;
    IPlen : integer;
  end;

var
  Server2Form: TServer2Form;

implementation

{$R *.DFM}

procedure TServer2Form.FormCreate(Sender: TObject);
begin
  Server2Dir.Text := 'c:\recvfax';
  ServerMemo.Lines.Clear;
  OkServBitBtn.Enabled := True;
  CancelServBitBtn.Enabled := True;
end;
```

```
procedure TServer2Form.OkServBitBtnClick(Sender: TObject);
begin
  initServer;
  BeginListening;
end;

procedure TServer2Form.initServer;
begin
  ErrorReturn := 0;
  Timedout := False;
end;

procedure TServer2Form.BeginListening;
begin
  ServerSocket.IPAddr := '';
  ServerSocket.Port := FaxAppPort;
  ServerSocket.NonBlocking := False;
  ServerSocket.Timeout := 0;

  ServerMemo.Lines.Add('>> waiting server1 <<');
  ServerSocket.SListen;

 if SocketsError then
   begin
   ServerMemo.Lines.Add('..no server1..');
   ServerSocket.SCancelListen;
   CloseSocket;
   Exit;
   end;

 DoServer;
end;

procedure TServer2Form.ServerSocketSessionClosed(Sender: TObject; Socket:
            Word);
begin
  ServerMemo.Lines.Add('..socket closed..');
  CloseSocket;
end;

procedure TServer2Form.ServerSocketErrorOccurred(Sender: TObject; Error:
            Integer; Msg: String);
begin
 ErrorReturn := Error;
 if Error = WSAETIMEDOUT then
    begin
    Timedout := True;
    ErrorReturn := 0;
    end;
 MessageDlg('@'+IntToStr(Error)+':'+Msg, mtError,[mbOk],0);
 if ServerSocket.SocketNumber <> INVALID_SOCKET then
    begin
    ServerMemo.Lines.Add('<< close socket >>');
    CloseSocket;
    end;
end;

function TServer2Form.SocketsError: Boolean;
begin
 Result := False;
 if Timedout or (ErrorReturn <> 0) then
```

```
    begin
    Result := True;
    ErrorReturn := 0;
    MessageDlg( 'Time out or socket error', mtError,[mbOk],0);
    end;
end;

procedure TServer2Form.CloseSocket;
begin
  if ServerSocket.SocketNumber <> INVALID_SOCKET then
     begin
     ServerSocket.SClose;
     end;
  OkServBitBtn.Enabled := True;
end;

procedure TServer2Form.CancelServBitBtnClick(Sender: TObject);
begin
  if ServerSocket.SocketNumber <> INVALID_SOCKET then
     ServerSocket.SClose;
  ServerMemo.Lines.Add('..server2 stoped..');

  initServer;
  Close;
end;

procedure TServer2Form.DoServer;
begin
 ServerSocket.SAccept;
 ServerMemo.Lines.Add('..connected..');

 ServerHandshake;
 if SocketsError then
   begin
   CloseSocket;
   Exit;
   end;

 GetFile;
 ServerMemo.Lines.Add('..finished..');
end;

procedure TServer2Form.GetFile;
var   f: File;
      numread: Integer;
      fsize, byteleft: LongInt;
      FileToSave,FileToSend : String;
      szBuf: array[0..2047] of Char;
      filecount : integer;
begin
 filecount := 0;
 while true do
  begin
  FileToSave := GetSavedFileName;
  fsize:= GetSavedFileSize;

  ServerMemo.Lines.Add('>'+ FileToSave +'<');
  ServerMemo.Lines.Add('>'+ IntToStr(fsize) +'<');

  AssignFile(f, Server2Dir.Text+'\'+ExtractFileName(FileToSave));
  {$I-}  Rewrite(f,1);  {$I+}
  if IOResult <> 0 then
```

```
      begin
      ServerMemo.Lines.Add('..error opening file..');
      Exit;
      end;

  byteleft := fsize;
  while (byteleft > 0) and (not SocketsError) do
    begin
    numread := sizeof(szBuf);
    numread :=
          ServerSocket.SReceive(ServerSocket.SocketNumber,szBuf,numread);
    if not SocketsError then
       begin
       BlockWrite(f,szBuf,numread);
       byteleft := byteleft - numread;
       end
       else begin
            ServerMemo.Lines.Add('..error while receiving..');
            ServerSocket.SClose;
            Exit;
            end;
    end;
  CloseFile(f);
  if MessageDlg('receive next file',mtConfirmation,[mbYES,mbNO],0) = mrNo
     then  break;
  inc(fileCount);
  end;
 ServerMemo.Lines.Add('..'+IntToStr(fileCount+1)+' files are received');

end;

{ get file's name is sent in parenthis from client }
function TServer2Form.GetSavedFileName: String;
var
 Buf: String;
 szBuf: array[0..255] of char absolute Buf;
 idx, len: integer;
begin
 Buf := '';
 Buf := ServerSocket.Peek;
 if SocketsError then
    Exit;
 idx := pos(')', Buf);
 if idx <= 0 then
   Exit;
 len := idx;
 ServerSocket.SReceive(ServerSocket.SocketNumber,@szBuf[1], len);
 if SocketsError then
   Exit;
 szBuf[0] := chr(len);
 Result := Copy(Buf, 2, length(Buf)-2);

end;

{ get size of file is sent in parenthis from client }
function TServer2Form.GetSavedFileSize: LongInt;
var
 Buf: String;
 szBuf: array[0..255] of char absolute Buf;
 ch: char;
 idx, len: integer;
begin
```

```
 Buf := '';
 Buf := ServerSocket.Peek;
 ch := #0;
 if SocketsError then
   Exit;
 idx := pos(')', Buf);
 if idx <= 0 then
   Exit;
 len := idx;
 ServerSocket.SReceive(ServerSocket.SocketNumber,@szBuf[1], len);
 if SocketsError then
   Exit;
 szBuf[0] := chr(len);
 Result := StrToInt(Copy(Buf, 2, length(Buf)-2));

end;

procedure TServer2Form.ServerHandshake;
var   str: String;
begin
  str := ServerSocket.Text;
  ServerMemo.Lines.Add('>'+str+'<');
  if Pos('START FAX', str) = 0 then
     begin
     ServerMemo.Lines.Add('..invalid header..');
     ErrorReturn := -1;
     Exit;
     end;

  ServerSocket.Text := 'OK';
end;

end.
```

โปรแกรม TOUSER2.PAS

```
{This program is used by server2 to send files to user2s}
unit ToUser2;

interface

uses
 SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
 Forms, Dialogs, StdCtrls, Buttons, Sockets, UServer2, FuncTn2, USendFax,
 UPhone;

const
  FaxAppPort = '2407';
type
  TSendToUser2Form = class(TForm)
    OkClientBitBtn: TBitBtn;
    CancelClientBitBtn: TBitBtn;
    ClientMemo: TMemo;
    Label4: TLabel;
    ClientSocket: TSockets;
    ReadDir: TEdit;
    Label1: TLabel;
    procedure OkClientBitBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure ClientSocketErrorOccurred(Sender: TObject; Error: Integer;
      Msg: String);
    procedure ClientSocketSessionClosed(Sender: TObject; Socket: Word);
    procedure CancelClientBitBtnClick(Sender: TObject);
  private
    { Private declarations }
    procedure initClient;
    procedure BeginSendFile;
    procedure SendToFaxMach;
    procedure CloseSocket;
    procedure DoClient;
    procedure SendFile;
    procedure ClientHandshake;
    procedure Sleep(stm : LongInt);
    function  SocketsError: Boolean;

  public
    { Public declarations }
    Timedout : Boolean;
    ErrorReturn : Integer;
    FileToSend, FaxIP : String;
    f : File;
    fName : String;
    bgin1,bgin2 : integer;
    Phone : String;
  end;

var
  SendToUser2Form: TSendToUser2Form;

implementation

{$R *.DFM}

procedure TSendToUser2Form.OkClientBitBtnClick(Sender: TObject);
begin
```

```
  initClient;
  BeginSendFile;
  SendToFaxMach;
end;

procedure TSendToUser2Form.FormCreate(Sender: TObject);
begin
  ClientMemo.Lines.Clear;
  OkClientBitBtn.Enabled := True;
  CancelClientBitBtn.Enabled := True;
  ReadDir.Text := 'c:\recvfax';
end;

procedure TSendToUser2Form.initClient;
begin
  Timedout := False;
  ErrorReturn := 0;
  ClientMemo.Lines.Add('initial client success');
end;

procedure TSendToUser2Form.BeginSendFile;
var   str1, Head1, Head2, Head3 : String;
      idx,idx1,idx2,idy,idy2,start1,start2 : integer;
      Fcount : integer;
      tx : TextFile;
      Dir : String;
begin
  Dir := ReadDir.Text;
  Fcount := 0;
  while true do
  begin
  FaxIP := '';
  Phone := '';
  fName := '';
  SendFaxForm.passFile := '';
  FileToSend := Dir+'\fax0'+IntToStr(Fcount)+'.zzz';
  AssignFile(f, FileToSend);  { assign FileToSend as untype file }
  AssignFile(tx,FileToSend);  { assign FileToSend as text file to read
                                header }
  {$I-}
  Reset(f,1);
  Reset(tx);
  {$I+}
  if IOResult <> 0 then
     begin
     ClientMemo.Lines.Add('can not open file '+
         ExtractFileName(FileToSend));
     end
     else
      begin
      str1 := '';
      Head1 := 'Destination Fax IP (';
      Head2 := 'Fax Phone No (';

      Readln(tx,str1);
      idx1 := Pos(Head1,str1);
      idx2 := Pos(Head2,str1);
      if idx1 > 0 then
         start1 := length(Head1)+1
         else if idx2>0 then
                 start1 := length(Head2)+1
                 else
```

```
                begin
                ClientMemo.Lines.Add('cannot read header');
                end;
    idy := Pos(')',str1);
    bgin1 := idy+3;
    if idy<=0 then
       else
       if idx1>0 then
          FaxIP := Copy(str1,start1+1,(idy-start1-1))
          else Phone := Copy(str1,start1+1,(idy-start1-1));

    Head3 := 'File Name (';
    str1 := '';

    ReadLn(tx,str1);
    idx := Pos(Head3,str1);
    idy2 := Pos(')',str1);
    start2 := length(Head3)+1;
    bgin2 := bgin1+idy2+3;
    if (idx<=0) or (idy2<=0) then
       else
       fName := Copy(str1,idx+start2-1,idy2-start2-1);

    CloseFile(tx);
    if faxIP<>'' then
       begin
       ClientMemo.Lines.Add('FaxIP '+FaxIP);
       ClientSocket.IPAddr := FaxIP;
       ClientSocket.Port := FaxAppPort;
       ClientSocket.NonBlocking := False;
       ClientSocket.Timeout := 0;
       ClientMemo.Lines.Add('>> trying to connect <<');
       ClientSocket.SConnect;
       if SocketsError then
          begin
          ClientMemo.Lines.Add('..can not find user2..');
          CloseSocket;
          Exit;
          end;
       DoClient;
       end
       else
        begin
        ClientMemo.Lines.Add('Phone '+Phone);

        {pass parameter (passfile) for sending to fax}
        SendFaxForm.Passfile := FileToSend;
        PhoneNumber := Phone;
        SendFaxForm.BeginAt := bgin2-2;
        SendFaxForm.SendFaxBrowseBtn.Enabled := False;
        SendFaxForm.DialBtn.Enabled := False;
        SendFaxForm.ShowModal;
        end;
    end;
 if MessageDlg('sever2 send to next user',mtConfirmation,
         [mbYES,mbNO],0) = mrNo then  break;
 inc(Fcount);
 end;         { end of while }

end;
```

```
{ read 'faxoX.aaa' that is send from fax sender, and send it to fax
  reciever }
procedure TSendToUser2Form.SendToFaxMach;
var   str1, Head2 : String;
      idx2,idy,start1 : integer;
      Fcount : integer;
      tx : TextFile;
      Dir : String;
begin
  ClientMemo.Lines.Add('..send to fax machine..');
  Dir := ReadDir.Text;
  Fcount := 0;
  while true do
  begin
  Phone := '';
  SendFaxForm.passFile := '';
  FileToSend := Dir+'\fax0'+IntToStr(Fcount)+'.aaa';
  AssignFile(f, FileToSend);
  AssignFile(tx,FileToSend);
  {$I-}
  Reset(f,1);
  Reset(tx);
  {$I+}
  if IOResult <> 0 then
     begin
     ClientMemo.Lines.Add('..can not open file..'+
            ExtractFileName(FileToSend));
     end
     else
      begin
      str1 := '';
      Head2 := 'Fax Phone No (';

      Readln(tx,str1);
      idx2 := Pos(Head2,str1);
      if idx2>0 then
         start1 := length(Head2)+1;
      idy := Pos(')',str1);
      bgin1 := idy+3;
      if idy<=0 then
         else
         Phone := Copy(str1,start1,(idy-start1));

      {pass parameter (passfile) for sending to fax}
      SendFaxForm.Passfile := FileToSend;
      PhoneNumber := Phone;
      SendFaxForm.BeginAt := 30;
      SendFaxForm.SendFaxBrowseBtn.Enabled := False;
      SendFaxForm.DialBtn.Enabled := False;
      SendFaxForm.ShowModal;
      end;

  if MessageDlg('sever2 send to next fax',mtConfirmation,[mbYES,mbNO],0) =
       mrNo then  break;
  inc(Fcount);
  end; { end of while }

end;

procedure TSendToUser2Form.ClientSocketErrorOccurred(Sender: TObject;
            Error: Integer; Msg: String);
begin
```

```
 ErrorReturn := Error;
 if Error = WSAETIMEDOUT then
    begin
    Timedout := True;
    ErrorReturn := 0;
    end;
 MessageDlg('@'+IntToStr(Error)+':'+Msg, mtError,[mbOk],0);
 if ClientSocket.SocketNumber <> INVALID_SOCKET then
    begin
    ClientMemo.Lines.Add('<< close socket >>');
    CloseSocket;
    end;
end;

function TSendToUser2Form.SocketsError: Boolean;
begin
 Result := False;
 if Timedout or (ErrorReturn <> 0) then
   begin
   Result := True;
   ErrorReturn := 0;
   MessageDlg( 'Time out or socket error', mtError,[mbOk],0);
   end;
end;

procedure TSendToUser2Form.CloseSocket;
begin
  if ClientSocket.SocketNumber <> INVALID_SOCKET then
     begin
     ClientSocket.SClose;
     end;
  OkClientBitBtn.Enabled := True;
end;

procedure TSendToUser2Form.ClientSocketSessionClosed(Sender: Tobject;
            Socket: Word);
begin
  ClientMemo.Lines.Add('>> socket closed <<');
  CloseSocket;
end;

procedure TSendToUser2Form.DoClient;
begin
 ClientMemo.Lines.Add('>> connected <<');
 OkClientBitBtn.Enabled := False;
 CancelClientBitBtn.Enabled := True;

 ClientHandshake;
 if SocketsError then
   begin
   ClientMemo.Lines.Add('>> socket error <<');
   closeSocket;
   Exit;
   end;

 SendFile;
 ClientMemo.Lines.Add('..finished..');

 OkClientBitBtn.Enabled := True;
 CancelClientBitBtn.Enabled := True;
end;
```

```
procedure TSendToUser2Form.SendFile;
var
 numread: Integer;
 fsize, bytewrite: LongInt;
 szBuf: array[0..2047] of Char;
begin
 ClientMemo.Lines.Add('..sending..');
 fsize := FileSize(f)-bgin2-1;    { get file's size to send }
 seek(f,bgin2-2);
 bytewrite := 0;                  { number of data (bytes) is sent }

 ClientSocket.Text := '('+fName+')';              { send (file name) }
 ClientSocket.Text := '('+IntToStr(fsize)+')';    { send (file size) }
 while not Eof(f) and not SocketsError do
   begin
   BlockRead(f, szBuf, sizeof(szBuf), numread);
   ClientSocket.SSend(ClientSocket.SocketNumber,szBuf,numread);
   bytewrite := bytewrite+numread;
   end;
 if SocketsError then
    begin
    ClientMemo.Lines.Add('..error while sending..');
    CloseSocket;
    end;

 ClientMemo.Lines.Add('..end of file..');
 sleep(1000);
 CloseFile(f);
 DeleteFile(FileToSend);
end;

procedure TSendToUser2Form.ClientHandshake;
var
 str: String;
begin
 ClientSocket.Text := 'START FAX';
 if SocketsError then
   begin
   ClientMemo.Lines.Add('..error..');
   Exit;
   end;

 str := ClientSocket.Text;
 if SocketsError then
   begin
   ClientMemo.Lines.Add('..error..');
   Exit;
   end;
 ClientMemo.Lines.Add('>'+str+'<');

 if Pos('OK', str) = 0 then
   begin
   ClientMemo.Lines.Add('..invalid reply..');
   ErrorReturn := -1;
   Exit;
   end;
end;

procedure TSendToUser2Form.Sleep(stm : LongInt);
var   t : LongInt;
begin
  t := GetTickCount;          { milliseconds }
```

```
  while (t+stm) >= GetTickCount do
        ;
end;

procedure TSendToUser2Form.CancelClientBitBtnClick(Sender: TObject);
begin
  if ClientSocket.SocketNumber <> INVALID_SOCKET then
     ClientSocket.SClose;
  ClientMemo.Lines.Add('..sending stoped..');

  initClient;
  Close;
end;

end.
```

โปรแกรม USER2.PAS

```
{This program is used by user2 to receive file from server2}
unit User2;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, Sockets;

const
  FaxAppPort = '2407';

type
  TUser2Form = class(TForm)
    EditServerDir: TEdit;
    Label1: TLabel;
    ServerMemo: TMemo;
    Label2: TLabel;
    OkServBitBtn: TBitBtn;
    CancelServBitBtn: TBitBtn;
    ServerSocket: TSockets;
    procedure OkServBitBtnClick(Sender: TObject);
    procedure CancelServBitBtnClick(Sender: TObject);
    procedure ServerSocketErrorOccurred(Sender: TObject; Error: Integer;
      Msg: String);
    procedure ServerSocketSessionClosed(Sender: TObject; Socket: Word);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    procedure initServer;
    procedure BeginListening;
    procedure DoServer;
    procedure CloseSocket;
    procedure GetFile;
    procedure ServerHandShake;
    function GetSavedFileName: String;
    function GetSavedFileSize: LongInt;
    function SocketsError: Boolean;

  public
    { Public declarations }
    Timedout : Boolean;
    ErrorReturn : Integer;
    IPlen : integer;
  end;

var
  User2Form: TUser2Form;

implementation

{$R *.DFM}

procedure TUser2Form.OkServBitBtnClick(Sender: TObject);
begin
  initServer;
  BeginListening;
end;
```

```
procedure TUser2Form.initServer;
begin
  ErrorReturn := 0;
  Timedout := False;
end;

procedure TUser2Form.BeginListening;
var   count : integer;
begin
   ServerSocket.IPAddr := '';
   ServerSocket.Port := FaxAppPort;
   ServerSocket.NonBlocking := False;
   ServerSocket.Timeout := 0;

   ServerMemo.Lines.Add('>> waiting server2 <<');
   ServerSocket.SListen;

  if SocketsError then
    begin
    ServerMemo.Lines.Add('..can not find server2..');
    ServerSocket.SCancelListen;
    CloseSocket;
    Exit;
    end;

  count := 0;
  while true do
    begin
    DoServer;
    if MessageDlg('receive next file',mtConfirmation,[mbYES,mbNO],0) =
         mrNo then break;
    inc(count);
    ServerMemo.Lines.Add('..waiting next file..');
    end;
 end;

 procedure TUser2Form.ServerSocketSessionClosed(Sender: TObject; Socket:
 Word);
 begin
   ServerMemo.Lines.Add('..socket closed..');
   CloseSocket;
 end;

 procedure TUser2Form.ServerSocketErrorOccurred(Sender: TObject; Error:
              Integer; Msg: String);
 begin
  ErrorReturn := Error;
  if Error = WSAETIMEDOUT then
     begin
     Timedout := True;
     ErrorReturn := 0;
     end;
  MessageDlg('@'+IntToStr(Error)+':'+Msg, mtError,[mbOk],0);
  if ServerSocket.SocketNumber <> INVALID_SOCKET then
     begin
     ServerMemo.Lines.Add('<< close socket >>');
     CloseSocket;
     end;
 end;

  function TUser2Form.SocketsError: Boolean;
  begin
```

```
 Result := False;
 if Timedout or (ErrorReturn <> 0) then
   begin
   Result := True;
   ErrorReturn := 0;
   MessageDlg( 'Time out or socket error', mtError,[mbOk],0);
   end;
end;

procedure TUser2Form.CloseSocket;
begin
  if ServerSocket.SocketNumber <> INVALID_SOCKET then
     begin
     ServerSocket.SClose;
     end;
  OkServBitBtn.Enabled := True;
end;

procedure TUser2Form.CancelServBitBtnClick(Sender: TObject);
begin
  if ServerSocket.SocketNumber <> INVALID_SOCKET then
     ServerSocket.SClose;
  ServerMemo.Lines.Add('..user2 stoped..');

  initServer;
  Close;
end;

procedure TUser2Form.DoServer;
begin
 ServerSocket.SAccept;
 ServerMemo.Lines.Add('..connected..');

 ServerHandshake;
 if SocketsError then
   begin
   CloseSocket;
   Exit;
   end;

 GetFile;
 ServerMemo.Lines.Add('..finished..');
end;

procedure TUser2Form.GetFile;
var   f: File;
      numread: Integer;
      fsize, byteleft: LongInt;
      FileToSave,FileToSend,FaxToSend,FaxHeader:String;
      str1: String;
      szBuf: array[0..2047] of Char;
begin
   FileToSave := GetSavedFileName;
   ServerMemo.Lines.Add('>'+ FileToSave +'<');

   fsize:= GetSavedFileSize;
   ServerMemo.Lines.Add('>'+ IntToStr(fsize) +'<');

   AssignFile(f, EditServerDir.Text+'\'+ExtractFileName(FileToSave));
   {$I-}  Rewrite(f,1); {$I+}
   if IOResult <> 0 then
     begin
```

```
    ServerMemo.Lines.Add('..error opening file..');
    Exit;
    end;

 byteleft := fsize;
 while (byteleft > 0) and (not SocketsError) do
   begin
   numread := sizeof(szBuf);
   numread :=
        ServerSocket.SReceive(ServerSocket.SocketNumber,szBuf,numread);
   if not SocketsError then
      begin
      BlockWrite(f,szBuf,numread);
      byteleft := byteleft - numread;
      end
      else begin
           ServerMemo.Lines.Add('..error while receiving..');
           ServerSocket.SClose;
           Exit;
           end;
   end;
 CloseFile(f);
 CloseSocket;
end;

{ get file's name is sent in parenthis from client }
function TUser2Form.GetSavedFileName: String;
var
 Buf: String;
 szBuf: array[0..255] of char absolute Buf;
 idx, idy, len: integer;
begin
  Buf := '';
  Buf := ServerSocket.Peek;
  if SocketsError then
     Exit;
  idx := pos(')',Buf);
  if idx <= 0 then
     Exit;
  len := idx;
  ServerSocket.SReceive(ServerSocket.SocketNumber,@szBuf[1],len);
  if SocketsError then
     Exit;
  szBuf[0] := chr(len);
  Result := Copy(Buf,2,length(Buf)-2);

 end;

 { get size of file is sent in parenthis from client }
 function TUser2Form.GetSavedFileSize: LongInt;
 var
  Buf: String;
  szBuf: array[0..255] of char absolute Buf;
  ch: char;
  idx, len: integer;
 begin
  Buf := '';
  Buf := ServerSocket.Peek;
  ch := #0;
  if SocketsError then
    Exit;
  idx := pos(')', Buf);
```

```
if idx <= 0 then
  Exit;
len := idx;
ServerSocket.SReceive(ServerSocket.SocketNumber,@szBuf[1], len);
if SocketsError then
  Exit;
szBuf[0] := chr(len);
Result := StrToInt(Copy(Buf, 2, length(Buf)-2));

end;

procedure TUser2Form.ServerHandshake;
var   str: String;
begin
  str := ServerSocket.Text;
  ServerMemo.Lines.Add('>'+str+'<');
  if Pos('START FAX', str) = 0 then
     begin
     ServerMemo.Lines.Add('..invalid header..');
     ErrorReturn := -1;
     Exit;
     end;

   ServerSocket.Text := 'OK';
end;

procedure TUser2Form.FormCreate(Sender: TObject);
begin
   EditServerDir.Text := 'c:\recvfax\data';
   ServerMemo.Lines.Clear;
   OkServBitBtn.Enabled := True;
   CancelServBitBtn.Enabled := True;

end;

end.
```

ไฟล์ DECLARE.PAS

```
{ this file is strore declaring variable from 'view.pas' and
  'codefile.pas' which are the same }

type
   HuffCode    = record
      bit      : byte;
      code     : word;
               end;

Const
   EOLcode     : HuffCode  = (bit:12; code:$0010);

   inverse     : array[0..255] of byte   =
      ( $00,$80,$40,$C0,$20,$A0,$60,$E0,$10,$90,$50,$D0,$30,$B0,$70,$F0,
        $08,$88,$48,$C8,$28,$A8,$68,$E8,$18,$98,$58,$D8,$38,$B8,$78,$F8,
        $04,$84,$44,$C4,$24,$A4,$64,$E4,$14,$94,$54,$D4,$34,$B4,$74,$F4,
        $0C,$8C,$4C,$CC,$2C,$AC,$6C,$EC,$1C,$9C,$5C,$DC,$3C,$BC,$7C,$FC,
        $02,$82,$42,$C2,$22,$A2,$62,$E2,$12,$92,$52,$D2,$32,$B2,$72,$F2,
        $0A,$8A,$4A,$CA,$2A,$AA,$6A,$EA,$1A,$9A,$5A,$DA,$3A,$BA,$7A,$FA,
        $06,$86,$46,$C6,$26,$A6,$66,$E6,$16,$96,$56,$D6,$36,$B6,$76,$F6,
        $0E,$8E,$4E,$CE,$2E,$AE,$6E,$EE,$1E,$9E,$5E,$DE,$3E,$BE,$7E,$FE,
        $01,$81,$41,$C1,$21,$A1,$61,$E1,$11,$91,$51,$D1,$31,$B1,$71,$F1,
        $09,$89,$49,$C9,$29,$A9,$69,$E9,$19,$99,$59,$D9,$39,$B9,$79,$F9,
        $05,$85,$45,$C5,$25,$A5,$65,$E5,$15,$95,$55,$D5,$35,$B5,$75,$F5,
        $0D,$8D,$4D,$CD,$2D,$AD,$6D,$ED,$1D,$9D,$5D,$DD,$3D,$BD,$7D,$FD,
        $03,$83,$43,$C3,$23,$A3,$63,$E3,$13,$93,$53,$D3,$33,$B3,$73,$F3,
        $0B,$8B,$4B,$CB,$2B,$AB,$6B,$EB,$1B,$9B,$5B,$DB,$3B,$BB,$7B,$FB,
        $07,$87,$47,$C7,$27,$A7,$67,$E7,$17,$97,$57,$D7,$37,$B7,$77,$F7,
        $0F,$8F,$4F,$CF,$2F,$AF,$6F,$EF,$1F,$9F,$5F,$DF,$3F,$BF,$7F,$FF);
```

โปรแกรม USENDFAX.PAS

```
unit Usendfax;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, MSComm, UConfig, UPhone;

type
  HuffmanCode    = record
        cnt      : byte;
        code     : word;
                 end;

type
  TSendFaxForm = class(TForm)
    EditHuffName: TEdit;
    Label1: TLabel;
    SendFaxBrowseBtn: TButton;
    ShowConfig: TEdit;
    Label2: TLabel;
    OkSendFaxBtn: TBitBtn;
    CloseSendBtn: TBitBtn;
    SendFaxConfigBtn: TButton;
    DialBtn: TButton;
    SendFaxMemo: TMemo;
    Label4: TLabel;
    MSCommSend: TMSComm;
    HuffOpenDialog: TOpenDialog;
    EditIDstring: TEdit;
    Label3: TLabel;
    CheckBox1: TCheckBox;
    procedure SendFaxConfigBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure DialBtnClick(Sender: TObject);
    procedure SendFaxBrowseBtnClick(Sender: TObject);
    procedure OkSendFaxBtnClick(Sender: TObject);
    procedure CloseSendBtnClick(Sender: TObject);
  private
    { Private declarations }
    procedure DoCommTransmitLow(Sender: TObject; Count: Word);
    procedure DoCommError(Sender: TObject; CommErr: Word; ErrStr: String)
      ;
    procedure DoCommCommEvent(Sender: TObject; CommEvent: TCommEvents);
    procedure DoCommReceive(Sender: TObject; Count: Word);
    procedure WaitForResp(rsp: String; tm: LongInt; param: integer);
    procedure Sleep(stm: LongInt);
    procedure ATcommand(cmmd: String);

    procedure initComm;
    procedure initCommVariable;
    procedure CommOpen;
    procedure CommClose;
    procedure ModemError;
    procedure QuitSerialComm;
    procedure initSendFaxModem;
    procedure DialFax;
    procedure PrepareToSend(sendFname : String);
    procedure HangUp;
```

```
    procedure SendPageToFax;
    function  Stop : Boolean;

    function  RemoveHeader(name :String; i :integer) : String;
    function  LoadHuffman(fiName:String) : String;
    function  IsHuffman(name :String) : Boolean;
    procedure CodeToHuff(Tname, Hname : String);
    procedure AsciiToPel;
    procedure OneDimention;
    procedure ToHuffman(var AnyCode :HuffmanCode);
    function  ErrorRead(var fi :FILE) :Boolean;
    function  ErrorWrite(var fi :FILE) :Boolean;

  public
    { Public declarations }
    PassFile : String;
    BeginAt  : integer;
  end;

var
  SendFaxForm: TSendFaxForm;
  CommTransmitLow : Boolean;
implementation

{$R *.DFM}
{$I declare.pas}
{$I cookie.pas}
{$I codefile.pas}

procedure TSendFaxForm.SendFaxConfigBtnClick(Sender: TObject);
var  ConfigFile : TextFile;
begin
  ConfigForm.ShowModal;     { see UConfig.pas }
  initComm;                 { see cookie.pas }
end;

procedure TSendFaxForm.FormCreate(Sender: TObject);
begin
  EditHuffName.Text := '';
  EditIDstring.Text := '';
  SendFaxMemo.Lines.Clear;
  initComm;
  CommTransmitLow := False;
end;

procedure TSendFaxForm.DialBtnClick(Sender: TObject);
begin
  DialForm.ShowModal;      { see UPhone.pas }
end;

procedure TSendFaxForm.SendFaxBrowseBtnClick(Sender: TObject);
begin
  if HuffOpenDialog.Execute then
     EditHuffName.Text := HuffOpenDialog.FileName
     else MessageDlg('no selected file'+#13+'  Browse again!!',
          mtInformation,[mbOK],0);
end;


{ main : when OK button is clicked.
```

```
    'passFile' is filename sent from 'ToUser2.pas',if passFile not
  equal null so program must send passFile to fax machine and phone number
  is also passed from 'ToUser2.pass', else program send file selected by
  user , phone number is also selected by user.
      passFile may have extension of '.zzz' or '.aaa'. If file '.zzz' so
  this file is sent from user1 and is text file, program must code into
  huffman before send to fax machine. If file '.aaa' so this file is sent
  from source fax machine and file is huffman file, program can send to
  fax receiver immediately.
      If passFile is null string, file to send and phone number are set
  by user, program will check file extension before. If file extension is
  '.HUF' then send it, else code to huffman before sending }
procedure TSendFaxForm.OkSendFaxBtnClick(Sender: TObject);
var   fName, hufName, newName, hName : String;
begin
  if passFile = '' then
     begin
     fName := EditHuffName.Text;
     hufName := LoadHuffman(fName);          { see cookie.pas }
     end
     else fName := PassFile;

  CloseSendBtn.Enabled := False;
  DialBtn.Enabled := False;
  SendFaxConfigBtn.Enabled := False;
  SendFaxBrowseBtn.Enabled := False;
  SendFaxMemo.Lines.Clear;
  initCommVariable;                          { see cookie.pas }
  CommOpen;                                  { see cookie.pas }
  if ToError then
     begin
     CommClose;                              { see cookie.pas }
     MessageDlg('Cannot open comm port',mtError,[mbOk],0);
     SendFaxConfigBtn.Enabled := True;
     CloseSendBtn.Enabled := True;
     exit;
     end
     else
       begin
       initSendFaxModem;                      { see cookie.pas }
       if Stop or ToError then
          begin
          SendFaxMemo.Lines.Add('cannot initial modem or Stop');
          SendFaxMemo.Lines.Add('try another port or close!');
          SendFaxConfigBtn.Enabled := True;
          CloseSendBtn.Enabled := True;
          QuitSerialComm;                     { see cookie.pas }
          exit;
          end
          else
            begin
            { if passfile <> 0, file is send from ToUser2.pas,
              not allow to view file }
            if passFile <> '' then
               begin
               newName := RemoveHeader(passFile,BeginAt);
               hufName := ChangeFileExt(newName,'.huf');

                { file '.zzz' is text file, must be coded before send or
                  it is '.aaa' file must be sent without code to huffman }
               if ExtractFileExt(passFile) = '.zzz' then
```

```
            begin
            CodeToHuff(newName,hufName);     { see codefile.pas }
            DialFax;                         { see cookie.pas }
            if Stop or ToError then
               begin
               SendFaxMemo.Lines.Add('');
               SendFaxMemo.Lines.Add('dialing error or stop');
               SendFaxMemo.Lines.Add('Try again or close');
               SendFaxConfigBtn.Enabled := True;
               DialBtn.Enabled := True;
               CloseSendBtn.Enabled := True;
               QuitSerialComm;
               exit;
               end;
            PrepareToSend(hufName);          { see cookie.pas }
            end
            else
              begin
              DialFax;
              if Stop or ToError then
                 begin
                 SendFaxMemo.Lines.Add('');
                 SendFaxMemo.Lines.Add('dialing error or stop');
                 SendFaxMemo.Lines.Add('Try again or close');
                 SendFaxConfigBtn.Enabled := True;
                 DialBtn.Enabled := True;
                 CloseSendBtn.Enabled := True;
                 QuitSerialComm;
                 exit;
                 end;
              PrepareToSend(newName);
              end;
         end
         else    {passfile = null string}
           begin
           DialFax;
           if Stop or ToError then
              begin
              SendFaxMemo.Lines.Add('');
              SendFaxMemo.Lines.Add('dialing error or stop');
              SendFaxMemo.Lines.Add('Try again or close');
              SendFaxConfigBtn.Enabled := True;
              DialBtn.Enabled := True;
              CloseSendBtn.Enabled := True;
              QuitSerialComm;
              exit;
              end;
           PrepareToSend(hufName);   {see cookie.pas}
           end;
      if ToError then
         begin
         SendFaxMemo.Lines.Add('');
         SendFaxMemo.Lines.Add('send page error!!');
         CloseSendBtn.Enabled := True;
         exit;
         end;
      HangUp;                              { see cookie.pas }
      QuitSerialComm;
    end;
end;
CloseSendBtn.Enabled := True;
```

```
  DialBtn.Enabled := True;
  SendFaxConfigBtn.Enabled := True;
  SendFaxBrowseBtn.Enabled := True;
  passFile := '';
  SendFaxMemo.Lines.Add('');
end;

{ this function is remove file's header and change file's extension into
  '.new' if 'passFile' is not equal to null string ,then return that
  name.
      'name' and 'i' are sent from 'ToUser2.pas'. 'name' is file's name,
  'i' is number of byte to remove }
function TSendFaxForm.RemoveHeader(name: String; i: integer) : String;
var   f1, f2 : file;
      buf : array[0..1023] of char;
      remain : integer;
      N : String[20];
begin
  AssignFile(f1,name);
  reset(f1,1);
  N := ChangeFileExt(name,'.new');
  AssignFile(f2,N);
  rewrite(f2,1);
  Seek(f1,i);
  while not EOF(f1) do
    begin
    BlockRead(f1,buf,sizeof(buf),remain);
    if remain <> 0 then
       BlockWrite(f2,buf,remain);
    end;

  CloseFile(f1);
  CloseFile(f2);

  Result := N;
end;

{can be clicked after finish sending, to close application}
procedure TSendFaxForm.CloseSendBtnClick(Sender: TObject);
begin
  Close;
end;

end.
```

ไฟล์ COOKIE.PAS

```
{ This file is full of functions that are used for send fax file to fax
  machine }

const
  CommBufSize = 1024;

  CED = #13#10+'CED';
  OK = #13#10+'OK'+#13#10;
  CONNECT = #13#10+'CONNECT'+#13#10;
  RING = #13#10+'RING'+#13#10;
  NoCarrier = #13#10+'NO CARRIER'+#13#10;
  NoDialTone = #13#10+'NO DIALTONE'+#13#10;
  BUSY = #13#10+'BUSY'+#13#10;

  DLE = #16;
  EOP = #46;  {reverse bit of 'end of page' code}
  ETX = #3;

var
  CommBuf      : array[0..CommBufSize-1] of char;  {store data read from
                                                    modem}
  Response     : String;
  CommDataPos  : Word;     {number of data income in port's buffer}
  CurrPos      : Word;     {point to data that is read from port's buffer}
  TransRate    : integer;
  IDstring     : String;
  ToError      : Boolean;
  HuffFileSize, Rest : LongInt;
  Huff         : File;
  RetryDial    : LongInt;

{ read information from 'fax.ini', show and set MSComm's parameter(such
  as Port, BaudRate, Number of Data Bit, Parity bit, Flow control, Stop
  bit) }
procedure TSendFaxForm.initComm;
var   fi  : Textfile;
      Str : String;
      br  : String[6];
      len : byte;
begin
  ToError := False;
  AssignFile(fi,'fax.ini');
  {$I-}  Reset(fi);  {$I+}
  if IOresult <> 0 then
     begin
     ToPort := 1;
     MSCommSend.Port := 1;

     ToRate := 1;
     MSCommSend.BaudRate := br2400;
     TransRate := 0;
     br := '2400';

     ToResol := 1;
     ToRetry := 1;
     end
     else
       begin
       Readln(fi,Str);
       len := Pos(Str,'PORT = ');
```

```
      ToPort := StrToInt(copy(Str,len+8,1));

      case ToPort of
       1 : MSCommSend.Port := 1;
       2 : MSCommSend.Port := 2;
       3 : MSCommSend.Port := 3;
       4 : MSCommSend.Port := 4;
       end;


      Readln(fi,Str);
      len := Pos(Str,'Transfer Rate = ');
      ToRate := StrToInt(copy(Str,len+17,1));

      case ToRate of
       1 : begin
           MSCommSend.BaudRate := br2400;
           TransRate := 0;
           br := '2400';
           end;
       2 : begin
           MSCommSend.BaudRate := br4800;
           TransRate := 1;
           br := '4800';
           end;
       3 : begin
           MSCommSend.BaudRate := br9600;
           TransRate := 3;
           br := '9600';
           end;
       4 : begin
           MSCommSend.BaudRate := br14400;
           TransRate := 5;
           br := '14400';
           end;
       end;

      Readln(fi,Str);
      len := Pos(Str,'Resolution = ');
      ToResol := StrToInt(copy(Str,len+14,1));

      Readln(fi,Str);
      len := Pos(Str,'Retry = ');
      ToRetry := StrToInt(copy(Str,len+9,1));
      CloseFile(fi);
      end; { end of else }

  { no parity, 8 data , 1 stop }
  MSCommSend.ParityBits := pbNone;
  MSCommSend.DataBits := dbEight;
  MSCommSend.StopBits := sbOne;
  MSCommSend.FlowControl := fcRTSCTS;

  SendFaxMemo.Lines.Add('MSComm Ready');
  ShowConfig.Text := 'com'+IntToStr(MSCommSend.port)+','+br+',n,8,1';
end;

{ when event 'OnReceive' occure(when data come to MSComm), the message is
  sent to program and call this procedure to read data and restore in
  buffer }
procedure TSendFaxForm.DoCommReceive(Sender: TObject; Count: Word);
var i : Word;
    ch : char;
```

```
      StrRes : String;
begin
  for i:=1 to Count do
      begin
      MSCommSend.Read(@ch, SizeOf(ch));
      CommBuf[CommDataPos] := ch;
      inc(CommDataPos);
      if CommDataPos = CommBufSize then
         CommDataPos := 0;
      StrRes := StrRes+ch;
      end;
end;

{ when event 'OnTransmitLow' occur, message is sent to program then call
  this procedure to sent DLE code and ETX code }
procedure TSendFaxForm.DoCommTransmitLow(Sender: TObject; Count: Word);
begin
  SendFaxMemo.Lines.Add('>CommTransmitLow');
  ATcommand(DLE);
  ATcommand(ETX);
  CommTransmitLow := True;
end;

{ when event 'OnError' occur, message is sent to program then call this
  procedure, but the event never occur then I'm not sure what to do }
procedure TSendFaxForm.DoCommError(Sender: TObject; CommErr: Word;
            ErrStr: String);
begin
  MessageDlg('DoCommError',mtinformation,[mbOk],0);
end;

{ when event 'OnCommEvent' occur, message is sent to program then call
  this procedure,but the event never occur then I'm not sure what to do }
procedure TSendFaxForm.DoCommCommEvent(Sender: TObject; CommEvent:
            TCommEvents);
begin
  MessageDlg('DoCommCommEvent',mtinformation,[mbOk],0);
end;

procedure TSendFaxForm.initCommVariable;
begin
  ToError := False;
  IDstring := EditIDstring.Text;
  { read ToRetry from 'fax.ini' and set sleep time (millisecs) }
  case ToRetry of
       1 : RetryDial := 10000;
       2 : RetryDial := 20000;
       3 : RetryDial := 30000;
       4 : RetryDial := 40000;
       5 : RetryDial := 50000;
       end;
end;

{ open MSComm to connect to MODEM, and set procedures for each event of
  MSComm }
procedure TSendFaxForm.CommOpen;
var   temp : array[0..254] of char;
begin
  if not MSCommSend.Open then
     begin
     Application.MessageBox(StrPCopy(temp,MSCommSend.GetError),
                            'Cannot open comm port',mb_iconstop);
```

```
    ToError := True;
    end
    else
      begin
      ToError := False;
      SendFaxMemo.Lines.Add('success in opening port');
      with MSCommSend do
           begin
           OnReceive := DoCommReceive;
           OnTransmitLow := DoCommTransmitLow;
           OnError := DoCommError;
           OnCommEvent := DoCommCommEvent;
           end;
      end;
end;

{ close port that has been opened }
procedure TSendFaxForm.CommClose;
begin
  MSCommSend.Close;
end;

{ sent command to modem, command correspond to variable cmmd }
procedure TSendFaxForm.ATcommand(cmmd: String);
begin
  MSCommSend.Write(@cmmd[1], Length(cmmd));
end;

{ wait for response from modem, if no correct response(rsp) in time(tm)
   then report error }
procedure TSendFaxForm.WaitForResp(rsp: String; tm: LongInt; param:
                                   integer);

var   start : LongInt;
      i     : integer;
begin
   Response := '';
   i :=0;
   start := GetTickCount;    { milleseconds }
   while true do
     begin
     if CurrPos <> CommDataPos then
        begin
        Response := Response + CommBuf[CurrPos];
        inc(i);
        inc(CurrPos);
        if CurrPos = CommBufSize then
           CurrPos := 0;
        if Pos(rsp,Response)<>0 then
           begin
           ToError := False;
           SendFaxMemo.Lines.Add(Response);
           exit;
           end;
        start := GetTickCount;  {re-start timer}
        end;
     if (GetTickCount-start) > tm then
        begin
        if param = 0 then
           begin
           QuitSerialComm;
           MessageDlg('No response from modem.',mtError,[mbOk],0);
```

```
          Exit;
          end
        else
          begin     {param = 1}
          ToError := True;      {fail}
          Exit;
          end;
        end;
    Application.ProcessMessages;
    if Stop then
       exit;
    end;  { end of while }
end;

{ if user check the CheckBox1 then return 'true' }
function TSendFaxForm.Stop:Boolean;
begin
  Result := CheckBox1.Checked;
end;

{ clear buffer, and close MSComm }
procedure TSendFaxForm.QuitSerialComm;
var   i : word;
begin
  for i:=0 to 59 do
      Response[i] := #0;
  for i:=0 to 59 do
      CommBuf[i] := #0;
  MSCommSend.Close;
  CloseSendBtn.Enabled := True;
end;

{ delay time in stm milliseconds }
procedure TSendFaxForm.Sleep(stm: LongInt);
var   t : LongInt;
begin
  t := GetTickCount;   { milliseconds }
  while (t+stm) >= GetTickCount do
        Application.ProcessMessages;
end;

{ send ATcommand set to initial modem before sending fax }
procedure TSendFaxForm.initSendFaxModem;
var   cmmd : String;
begin
   cmmd := 'ATZ'+#13;
   ATcommand(cmmd);
   WaitForResp(OK,5000,1);
   if ToError then
      begin
      ModemError;
      exit;
      end;
   sleep(800);
   cmmd := 'ATE1V1'+#13;
   ATcommand(cmmd);
   WaitForResp(OK,5000,1);
   if ToError then
      begin
      ModemError;
      exit;
```

```
    end;
 cmmd := 'ATM1X4&D2S7=120S8=2'+#13;
 ATcommand(cmmd);
 WaitForResp(OK,5000,1);
 if ToError then
    begin
    ModemError;
    exit;
    end;
 cmmd := 'AT+FLID="' +IDstring+ '"'+#13;
 ATcommand(cmmd);
 WaitForResp(OK,5000,1);
 if ToError then
    begin
    ModemError;
    exit;
    end;
 cmmd := 'AT+FCR=1;+FCLASS=2'#13;
 ATcommand(cmmd);
 WaitForResp(OK,5000,1);
 if ToError then
    begin
    ModemError;
    exit;
    end;
 cmmd := 'AT+FDIS='+IntToStr(ToResol-1)+','+IntToStr(TransRate)+
         ',0,2,0,0,0,5'#13;
 ATcommand(cmmd);
 WaitForResp(OK,5000,1);
 if ToError then
    begin
    ModemError;
    exit;
    end;
  SendFaxMemo.Lines.Add('initial modem success');
end;

{ if no request response from modem, call procedure 'QuitSerialComm' then
  report error }
procedure TSendFaxForm.ModemError;
begin
  QuitSerialComm;
  MessageDlg('No Request Response',mtError,[mbOk],0);
  MessageDlg('Close Application !!!',mtWarning,[mbOk],0);
end;

{ send set of commands to dial fax machine and check connection, if not
  success then wait for a while and redial }
procedure TSendFaxForm.DialFax;
var   tmp : String;
begin
  SendFaxMemo.Lines.Add('>> dialing '+PhoneNumber+' <<');
  tmp := 'ATDT '+PhoneNumber+#13;
  ATcommand(tmp);
  WaitForResp(OK,30000,1);

  if ToError then
     begin
     SendFaxMemo.Lines.Add('cannot connect to '+PhoneNumber);
     SendFaxMemo.Lines.Add('..sleeping..');
     sleep(RetryDial);
```

```
    SendFaxMemo.Lines.Add('');
    SendFaxMemo.Lines.Add('>> redial '+PhoneNumber+' <<' );

    ATcommand(tmp);
    WaitForResp(OK,30000,1);
    if ToError then
       begin
       SendFaxMemo.Lines.Add('>> cannot dial <<');
       SendFaxMemo.Lines.Add('>> please try again later <<');
       HangUp;
       exit;
       end;
    end;
  sleep(200);
  tmp := 'AT+FDT'+#13;
  ATcommand(tmp);
  WaitForResp(CONNECT,30000,1);
  if ToError then
     begin
     HangUp;
     exit;
     end;
   SendFaxMemo.Lines.Add('Dial success');
end;

{after connect to fax machine, send document to fax machine}
procedure TSendFaxForm.PrepareToSend(sendFname : String);
begin
   if not IsHuffman(sendFname) then
      exit;
   AssignFile(Huff,sendFname);
   {$I-} Reset(Huff,1); {$I+}

  {begin send fax}
   seek(Huff,26);
   HuffFileSize := FileSize(Huff);
   Rest := HuffFileSize;

   SendPageToFax;
   CloseFile(Huff);
 end;

 { check file header, if file isn't huffman file, return 'false' }
 function TSendFaxForm.IsHuffman(name : String) : Boolean;
 var     BufStr : String;
         f : File;
 begin
    AssignFile(f,name);
    ReSet(f,1);
    BlockRead(f,BufStr[1],20);
    BufStr[0] := Chr(20);
    if (pos('HUFFMAN_FILE',BufStr) = 0 then
       begin
       MessageDlg('Not a Huffman File',mtError,[mbCancel],0);
       Result := false;
       CloseFile(f);
       exit;
       end;
    CloseFile(f);
    Result := true;
  end;
```

```
{ write data to MSComm for sending to fax machine until end of file, send
  set of codes after data, wait for report from fax receiver }
procedure TSendFaxForm.SendPageToFax;
var   Done : integer;
      SendBuf  : array[0..CommBufSize-1] of char;
      tmp,fptStr,fhngStr  : String;
      i,fptInt,fhngInt : integer;
      Response1, Response2: String;
begin
  While not EOF(Huff) do
    begin
    BlockRead(Huff, SendBuf, SizeOf(SendBuf), Done);
    for i:=0 to (Done-1) do
        begin
        tmp := SendBuf[i];
        if tmp = DLE then
           ATcommand(tmp);      {if data is 10h then send 10h again}
        ATcommand(tmp);
        end;
    end;      {if end of file then leave while loop}

  SendFaxMemo.Lines.Add('end of file');

 { after end of data, wait for OnTransmitLow message from MSComm, then
   send DLE and ETX code to fax modem }
  While not CommTransmitLow do
    Application.ProcessMessages;
  WaitForResp(OK,8000,1);
  if ToError then
     begin
     SendFaxMemo.Lines.Add('(e)no fax response');
     end
     else SendFaxMemo.Lines.Add('send fax message ok');
  sleep(2000);

  tmp := 'AT+FET=2'+#13;
  ATcommand(tmp);
  WaitForResp('+FPTS:',10000,1);
  if ToError then
     begin
     SendFaxMemo.Lines.Add('(e)+FPTS');
     fptStr := '-1'
     end
     else
      begin
      WaitForResp(#13,10000,1);
      fptStr := copy(Response,1,2);
      end;
   fptInt := StrToInt(fptStr);
   SendFaxMemo.Lines.Add('check value of +FPTS response:'+fptStr);
   case fptInt of
        -1 : SendFaxMemo.Lines.Add('+FPTS response not found');
         1 : SendFaxMemo.Lines.Add('page good');
         2 : SendFaxMemo.Lines.Add('page bad,retrain requested');
         3 : SendFaxMemo.Lines.Add('page good,retrain requested');
         4 : SendFaxMemo.Lines.Add('page bad,interrupt requested');
         5 : SendFaxMemo.Lines.Add('page good,interrupt requested');
        end;
   WaitForResp('+FHNG:',10000,1);
   if ToError then
      begin
      SendFaxMemo.Lines.Add('(e)+FHNG:');
```

```
    fhngStr := '-1'
    end
    else
     begin
     WaitForResp(#13,10000,1);
     fhngStr := copy(Response,1,2);
     end;
  fhngInt := StrToInt(fhngStr);
  SendFaxMemo.Lines.Add('check value of +FHNG response:'+fhngStr);
  case fhngInt of
       -1 : SendFaxMemo.Lines.Add('+FHNG response not found');
        0 : SendFaxMemo.Lines.Add('normal call termination');
        else SendFaxMemo.Lines.Add('error call termination');
        end;

  WaitForResp(OK,8000,1);
  if ToError then
     begin
     SendFaxMemo.Lines.Add('(e)no OK');
     end;

  SendFaxMemo.Lines.Add('send page complete');
  MessageDlg('Close Now!!!',mtConfirmation,[mbOK],0);

end;

{ send command set to hang up the phone and cancel connection }
procedure TSendFaxForm.HangUp;
var   tmp : String;
begin
   SendFaxMemo.Lines.Add('Hang Up');
   tmp := '+++';
   ATcommand(tmp);
   WaitForResp(OK,3000,1);
   if ToError then
      exit;
   tmp := 'ATH0'+#13;
   ATcommand(tmp);
   WaitforResp(OK,3000,1);
   if ToError then
      exit;
   tmp := 'ATZ'+#13;
   ATcommand(tmp);
   WaitforResp(OK,3000,1);
   if ToError then
      exit;
 end;

 { this procedure is called when you click 'OK' button, if file extension
   in 'EditHuffName.Text' is not '.HUF', call CodeToHuff() }
 function TSendFaxForm.LoadHuffman(fiName:String) : String;
 var   hName,Ext : String;
 begin
   Ext := ExtractFileExt(finame);
   if Ext <> '.HUF' then
      begin
      hName := ChangeFileExt(finame,'.HUF');
      CodeToHuff(finame,hName);
      Result := hName;
      end
      else Result := fiName;
```

โปรแกรม UCONFIG.PAS

```
unit Uconfig;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, MSComm;

type
  TConfigForm = class(TForm)
    PortGroup: TGroupBox;
    RateGroup: TGroupBox;
    ResolGroup: TGroupBox;
    RedialGroup: TGroupBox;
    RadioCom1: TRadioButton;
    RadioCom2: TRadioButton;
    RadioCom3: TRadioButton;
    RadioCom4: TRadioButton;
    Rate2400: TRadioButton;
    Rate4800: TRadioButton;
    Rate9600: TRadioButton;
    RadioNormal: TRadioButton;
    RadioFine: TRadioButton;
    Retry10: TRadioButton;
    Retry20: TRadioButton;
    Retry30: TRadioButton;
    Retry40: TRadioButton;
    Retry50: TRadioButton;
    Rate14400: TRadioButton;
    ChangeConfig: TBitBtn;
    procedure FormCreate(Sender: TObject);
    procedure ChangeConfigClick(Sender: TObject);
  private
    { Private declarations }
    function ReadError  :Boolean;
    function WriteError :Boolean;
    procedure SaveFile;
  public
    { Public declarations }
  end;

var
  ConfigForm: TConfigForm;
  fi : TextFile;
  ToPort,ToRate,ToResol,ToRetry : integer;

implementation

{$R *.DFM}

{ if no existing 'fax.ini' then set initial parameters (port number,
  transfer rate, retry dial time, and fax resolution), else read
  information from file and set parameters }
procedure TConfigForm.FormCreate(Sender: TObject);
var  len : integer;
     Str : string;
begin
   AssignFile(fi,'fax.ini');
   if ReadError then   { no existing 'fax.ini' }
      begin
```

```
      ToPort := 1;        { Port 1-4 ->> com1,com2,com3,com4 }
      ToRate := 1;        { Transfer Rate 1-5 ->> 2400,4800,9600,14400 }
      ToResol:= 1;        { Resolution 1,2 ->> normal,fine }
      ToRetry:= 1;        { retry dialling 1-5 ->> 10,20,30,40,50 sec }

      { initial form }
      RadioCom1.Checked := true;
      Rate2400.Checked := true;
      RadioNormal.Checked := true;
      Retry10.Checked := true;
      end
      else        { fax.ini is existing }
        begin     { read fax.ini and set visible form }
        Readln(fi,Str);
        len := Pos(Str,'PORT = ');
        ToPort := StrToInt(copy(Str,len+8,1));
        case ToPort of
             1 : RadioCom1.Checked := True;
             2 : RadioCom2.Checked := True;
             3 : RadioCom3.Checked := True;
             4 : RadioCom4.Checked := True;
             end;
        Readln(fi,Str);
        len := Pos(Str,'Transfer Rate = ');
        ToRate := StrToInt(copy(Str,len+17,1));
        case ToRate of
             1 : Rate2400.Checked := True;
             2 : Rate4800.Checked := True;
             3 : Rate9600.Checked := True;
             4 : Rate14400.Checked := True;
             end;
        Readln(fi,Str);
        len := Pos(Str,'Resolution = ');
        ToResol := StrToInt(copy(Str,len+14,1));
        case ToResol of
             1 : RadioNormal.Checked := True;
             2 : RadioFine.Checked := True;
             end;
        Readln(fi,Str);
        len := Pos(Str,'Retry = ');
        ToRetry := StrToInt(copy(Str,len+9,1));
        case ToRetry of
             1 : Retry10.Checked := True;
             2 : Retry20.Checked := True;
             3 : Retry30.Checked := True;
             4 : Retry40.Checked := True;
             5 : Retry50.Checked := True;
             end;
        end;
end;

function TConfigForm.ReadError :Boolean;
begin
   {$I-}
   Reset(fi);
   {$I+}
   Result := IOresult<>0;
end;

function TConfigForm.WriteError :Boolean;
begin
   {$I-}
```

```
   ReWrite(fi);
   {$I+}
   Result := IOresult<>0;
end;

procedure TConfigForm.SaveFile;
begin
   if WriteError then
      begin
      MessageDlg('Cannot ReWrite fax.ini',mtERROR,[mbOK],0);
      CloseFile(fi);
      close;
      end
      else
        begin
        Writeln(fi,'Port = '+IntToStr(ToPort));
        Writeln(fi,'Transfer Rate = '+IntToStr(ToRate));
        Writeln(fi,'Resolution = '+IntToStr(ToResol));
        Writeln(fi,'Retry = '+IntToStr(ToRetry));
        CloseFile(fi);
        end;
end;

{ save information of all parameters into file 'fax.ini',depend on user's
  setting on form }
procedure TConfigForm.ChangeConfigClick(Sender: TObject);
begin
  if RadioCom1.Checked = True then
     ToPort := 1
     else if RadioCom2.Checked = True then
          ToPort := 2
          else if RadioCom3.Checked = True then
               ToPort := 3
               else if RadioCom4.Checked = True then
                    ToPort := 4;

  if Rate2400.Checked = True then
     ToRate := 1
     else if Rate4800.Checked = True then
          ToRate := 2
          else if Rate9600.Checked = True then
               ToRate := 3
               else if Rate14400.Checked = True then
                    ToRate := 4;

  if ReTry10.Checked = True then
     ToRetry := 1
     else if Retry20.Checked = True then
          ToRetry := 2
          else if Retry30.Checked = True then
               ToRetry := 3
               else if Retry40.Checked = True then
                    ToRetry := 4
                    else if Retry50.Checked = True then
                    ToRetry := 5;

  if RadioNormal.Checked = True then
     ToResol := 1
     else ToResol := 2;

  { write to fax.ini }
  SaveFile;
```

```
  close;
end;

end.
```

โปรแกรม UPHONE.PAS

```
unit Uphone;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons;

type
  TDialForm = class(TForm)
    EditPhoneNo: TEdit;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
    fi : TextFile;
    procedure WriteDialFile;
    function  ReadError(var fi:TextFile) : Boolean;
    function  WriteError(var fi:TextFile) : Boolean;

  public
    { Public declarations }
    Number : String;
  end;

var
  DialForm: TDialForm;
  PhoneNumber : String;
implementation

{$R *.DFM}

procedure TDialForm.FormCreate(Sender: TObject);
var   tmp : String;
      len : byte;
begin
  AssignFile(fi,'dial.no');
  if not ReadError(fi) then
     begin              { existing 'dial.no'}
     EditPhoneNo.Text := '';
     Readln(fi, tmp);
     len := pos('Phone Number : ',tmp);
     PhoneNumber := copy(tmp,len+15,length(tmp)-15);
     EditPhoneNo.Text := PhoneNumber;
     CloseFile(fi);
     end;
end;

function TDialForm.ReadError(var fi:TextFile) : Boolean;
begin
  Result := False;
  {$I-} Reset(fi); {$I+}
  if IOresult <> 0 then
     Result := True;
end;

function TDialForm.WriteError(var fi:TextFile): Boolean;
```

```
begin
  Result := False;
  {$I-} ReWrite(fi);  {$I+}
  if IOresult <> 0 then
     Result := True;
end;

procedure TDialForm.WriteDialFile;
var   tmp : String;
begin
  tmp := 'Phone Number : '+PhoneNumber;
  Writeln(fi, tmp);
  CloseFile(fi);
end;

procedure TDialForm.BitBtn1Click(Sender: TObject);
begin
  PhoneNumber := EditPhoneNo.Text;
  AssignFile(fi,'dial.no');
  if WriteError(fi) then
     begin
     CloseFile(fi);
     MessageDlg('Error Writing File',mtError,[mbOK],0);
     Close;
     end
     else
     WriteDialFile;
  Close;
end;

end.
```

ไฟล์ CODEFILE.PAS

```
{ program CodeFile.Pas, used for change text file into huffman file, and
  convert bit MSB-LSB , called by UsendFax.pas }

const  CR          = #13;
       LF          = #10;
       TAB         = #9;
       TabSize : integer    = 8;
       PEL         = 216;
       Blk         = 1;
       Wht         = 0;
       BfSize      = 4096;
       HuffmanHeader: String = 'HUFFMAN_FILE';

       Terminate  : array[0..63,0..1] of HuffmanCode = (
       ((cnt: 8;code:$3500), (cnt:10;code:$0D0C)),
       ((cnt: 6;code:$1C00), (cnt: 3;code:$4000)),
       ((cnt: 4;code:$7000), (cnt: 2;code:$C000)),
       ((cnt: 4;code:$8000), (cnt: 2;code:$8000)),
       ((cnt: 4;code:$B000), (cnt: 3;code:$6000)),
       ((cnt: 4;code:$C000), (cnt: 4;code:$3000)),
       ((cnt: 4;code:$E000), (cnt: 4;code:$2000)),
       ((cnt: 4;code:$F000), (cnt: 5;code:$1800)),
       ((cnt: 5;code:$9800), (cnt: 6;code:$1400)),
       ((cnt: 5;code:$A000), (cnt: 6;code:$1000)),
       ((cnt: 5;code:$3800), (cnt: 7;code:$0800)),
       ((cnt: 5;code:$4000), (cnt: 7;code:$0A00)),
       ((cnt: 6;code:$2000), (cnt: 7;code:$0E00)),
       ((cnt: 6;code:$0C00), (cnt: 8;code:$0400)),
       ((cnt: 6;code:$D000), (cnt: 8;code:$0700)),
       ((cnt: 6;code:$D400), (cnt: 9;code:$0C00)),
       ((cnt: 6;code:$A800), (cnt:10;code:$05C0)),
       ((cnt: 6;code:$AC00), (cnt:10;code:$0600)),
       ((cnt: 7;code:$4E00), (cnt:10;code:$0200)),
       ((cnt: 7;code:$1800), (cnt:11;code:$0CE0)),
       ((cnt: 7;code:$1000), (cnt:11;code:$0D00)),
       ((cnt: 7;code:$2E00), (cnt:11;code:$0D80)),
       ((cnt: 7;code:$0600), (cnt:11;code:$06E0)),
       ((cnt: 7;code:$0800), (cnt:11;code:$0500)),
       ((cnt: 7;code:$5000), (cnt:11;code:$02E0)),
       ((cnt: 7;code:$5600), (cnt:11;code:$0300)),
       ((cnt: 7;code:$2600), (cnt:12;code:$0CA0)),
       ((cnt: 7;code:$4800), (cnt:12;code:$0CB0)),
       ((cnt: 7;code:$3000), (cnt:12;code:$0CC0)),
       ((cnt: 8;code:$0200), (cnt:12;code:$0CD0)),
       ((cnt: 8;code:$0300), (cnt:12;code:$0680)),
       ((cnt: 8;code:$1A00), (cnt:12;code:$0690)),
       ((cnt: 8;code:$1B00), (cnt:12;code:$06A0)),
       ((cnt: 8;code:$1200), (cnt:12;code:$06B0)),
       ((cnt: 8;code:$1300), (cnt:12;code:$0D20)),
       ((cnt: 8;code:$1400), (cnt:12;code:$0D30)),
       ((cnt: 8;code:$1500), (cnt:12;code:$0D40)),
       ((cnt: 8;code:$1600), (cnt:12;code:$0D50)),
       ((cnt: 8;code:$1700), (cnt:12;code:$0D60)),
       ((cnt: 8;code:$2800), (cnt:12;code:$0D70)),
       ((cnt: 8;code:$2900), (cnt:12;code:$06C0)),
       ((cnt: 8;code:$2A00), (cnt:12;code:$06D0)),
       ((cnt: 8;code:$2B00), (cnt:12;code:$0DA0)),
       ((cnt: 8;code:$2C00), (cnt:12;code:$0DB0)),
       ((cnt: 8;code:$2D00), (cnt:12;code:$0540)),
       ((cnt: 8;code:$0400), (cnt:12;code:$0550)),
       ((cnt: 8;code:$0500), (cnt:12;code:$0560)),
```

```
        ((cnt: 8;code:$0A00), (cnt:12;code:$0570)),
        ((cnt: 8;code:$0B00), (cnt:12;code:$0640)),
        ((cnt: 8;code:$5200), (cnt:12;code:$0650)),
        ((cnt: 8;code:$5300), (cnt:12;code:$0520)),
        ((cnt: 8;code:$5400), (cnt:12;code:$0530)),
        ((cnt: 8;code:$5500), (cnt:12;code:$0240)),
        ((cnt: 8;code:$2400), (cnt:12;code:$0370)),
        ((cnt: 8;code:$2500), (cnt:12;code:$0380)),
        ((cnt: 8;code:$5800), (cnt:12;code:$0270)),
        ((cnt: 8;code:$5900), (cnt:12;code:$0280)),
        ((cnt: 8;code:$5A00), (cnt:12;code:$0580)),
        ((cnt: 8;code:$5B00), (cnt:12;code:$0590)),
        ((cnt: 8;code:$4A00), (cnt:12;code:$02B0)),
        ((cnt: 8;code:$4B00), (cnt:12;code:$02C0)),
        ((cnt: 8;code:$3200), (cnt:12;code:$05A0)),
        ((cnt: 8;code:$3300), (cnt:12;code:$0660)),
        ((cnt: 8;code:$3400), (cnt:12;code:$0670)));

        MakeUp      : array[0..26,0..1] of HuffmanCode = (
        ((cnt: 5;code:$D800), (cnt:10;code:$03C0)),
        ((cnt: 5;code:$9000), (cnt:12;code:$0C80)),
        ((cnt: 6;code:$5C00), (cnt:12;code:$0C90)),
        ((cnt: 7;code:$6E00), (cnt:12;code:$05B0)),
        ((cnt: 8;code:$3600), (cnt:12;code:$0330)),
        ((cnt: 8;code:$3700), (cnt:12;code:$0340)),
        ((cnt: 8;code:$6400), (cnt:12;code:$0350)),
        ((cnt: 8;code:$6500), (cnt:13;code:$0360)),
        ((cnt: 8;code:$6800), (cnt:13;code:$0368)),
        ((cnt: 8;code:$6700), (cnt:13;code:$0250)),
        ((cnt: 9;code:$6600), (cnt:13;code:$0258)),
        ((cnt: 9;code:$6680), (cnt:13;code:$0260)),
        ((cnt: 9;code:$6900), (cnt:13;code:$0268)),
        ((cnt: 9;code:$6980), (cnt:13;code:$0390)),
        ((cnt: 9;code:$6A00), (cnt:13;code:$0398)),
        ((cnt: 9;code:$6A80), (cnt:13;code:$03A0)),
        ((cnt: 9;code:$6B00), (cnt:13;code:$03A8)),
        ((cnt: 9;code:$6B80), (cnt:13;code:$03B0)),
        ((cnt: 9;code:$6C00), (cnt:13;code:$03B8)),
        ((cnt: 9;code:$6C80), (cnt:13;code:$0290)),
        ((cnt: 9;code:$6D00), (cnt:13;code:$0298)),
        ((cnt: 9;code:$6D80), (cnt:13;code:$02A0)),
        ((cnt: 9;code:$4C00), (cnt:13;code:$02A8)),
        ((cnt: 9;code:$4C80), (cnt:13;code:$02D0)),
        ((cnt: 9;code:$4D00), (cnt:13;code:$02D8)),
        ((cnt: 6;code:$6000), (cnt:13;code:$0320)),
        ((cnt: 9;code:$4D80), (cnt:13;code:$0328)));

        EOL         : HuffmanCode         = (cnt:12; code:$0010);
        ZeroCode    : HuffmanCode         = (cnt:1; code:0);
        ToCompare   : array[0..15] of word  = 8000,$4000,$2000,$1000,$800,
                      $400,$200,$100,$80,$40,$20,$10,8,4,2,1);
        bits        : array[0..7]  of byte  = ($80,$40,$20,$10,8,4,2,1);

var     readfont        : array[0..255,0..19,0..1] of char;
        WriteBufPos,BufPos,WriteBufBit: integer;
        BufBit,fontRes,remain,Wremain : integer;
        ReadBf,WriteBf                : array[0..BfSize-1] of char;
        temp                          : array[0..199] of char;
        buf                           : array[0..19,0..215] of char;
        i,j,k,row,x,y,CodeBit         : integer;
        ChCount,Bfcount               : integer;
        ToCount, ToFlag               : Word;
        FiSize,size,ToPosition        : LongInt;
        EndCheck                      : Boolean;
```

```
        InputFile                       : File;
        OutputFile,FontFile             : File;

{ main: Tname is text-file name, Hname is huffman-file name, Both are
 passed from 'USendFax.Pas'.
   Program read block of data (4096 bytes) until byte-data is 'end of
 file code' or number of read data is equal to file's size.
   Check and save byte into temporary variable until read byte is 'new
 line code', then code 1 ASCII line into black and white runlength and
 code to huffman code
   After coding, reverse bit MSB->LSB and save in file's name of Hname }

procedure TSendFaxForm.CodeToHuff(Tname, Hname : String);
begin
  AssignFile(FontFile,'normal2.fon');
  if ErrorRead(FontFile) then
     begin
     SendFaxMemo.Lines.Add('Error in opening fontfile');
     exit;
     end;
  BlockRead(FontFile,readfont,SizeOf(readfont),fontRes);
  if fontRes = 0 then
     begin
     SendFaxMemo.Lines.Add('Error in normal2.fon');
     CloseFile(FontFile);
     exit;
     end;
  CloseFile(FontFile);

  AssignFile(InputFile,Tname);
  AssignFile(OutputFile,Hname);

  if (ErrorRead(InputFile) or ErrorWrite(OutputFile)) then
     begin
      SendFaxMemo.Lines.Add('Error in opening file');
      CloseFile(InputFile);
      CloseFile(OutputFile);
      exit;
     end;
  WriteBufPos := 0;
  WriteBufBit := WriteBufPos;
  ChCount := 0;
  FiSize := FileSize(InputFile);
  for ChCount := 0 to SizeOf(WriteBf)-1 do
      WriteBf[ChCount] := #0;
  ChCount := 0;

  BlockWrite(OutputFile,WriteBf,1024);
  size := 1;
  while not EOF(InputFile) do
    begin
    BlockRead(InputFile,ReadBf,BfSize,remain);
    i := 0;
    while i < remain do
       begin
       if Ord(ReadBf[i]) >= 32 then
          begin
          if ReadBf[i] <> #141 then
             begin
             temp[ChCount] := ReadBf[i];
             ChCount := ChCount + 1;
             end
```

```
    else
     if ReadBf[i+1] = LF then
         begin
         temp[ChCount] := CR;
         ChCount := ChCount+1;
         inc(i);
         inc(size);
         end;
    end
  else begin
       case ReadBf[i] of
            TAB : begin
                  for x:=0 to TabSize-1 do
                    begin
                    temp[ChCount] := #32;
                    inc(ChCount);
                    end;
                  end;
            LF  : begin
                  temp[ChCount] := CR;
                  inc(ChCount);
                  inc(size);
                  end;
            CR  : begin
                  temp[ChCount] := CR;
                  inc(ChCount);
                  inc(i);
                  inc(size);
                  end;
            #26 : begin
                  temp[ChCount] := CR;
                  inc(ChCount);
                  if (ChCount>=2) and (temp[ChCount-2] = chr($9E))
                    then temp[ChCount-2] := #32;
                  EndCheck := true;
                  i := remain;
                  end;
       end; { case }
       end; { else }
  if size = FiSize then
     begin
     temp[Chcount] := CR;
     inc(ChCount);
     EndCheck := true;
     end;
  if (ChCount > 0) and (temp[ChCount-1] = CR) then
     begin
     ChCount :=0;
     AsciiToPel;
     OneDimention;
     if temp[ChCount-1] <> CR then
        begin
        AsciiToPel;
        OneDimention;
        end;
     ChCount := 0;
     end;
  inc(size);
  inc(i);
  end;
if EndCheck then break;
```

```
    inc(size);
    end;

  for i:=1 to 6 do
      ToHuffman(EOL);
  if WriteBufPos < BfSize then
     BlockWrite(OutputFile,WriteBf,WriteBufPos,Wremain);

  seek(OutputFile,0);
  BlockWrite(OutputFile,HuffmanHeader[1],Length(HuffmanHeader));

  CloseFile(InputFile);
  CloseFile(OutputFile);

end;

{ return 'false' if cannot open file to read }
function TSendFaxForm.ErrorRead(var fi :FILE) :Boolean;
begin
   {$I-}
   Reset(fi,1);
   {$I+}
   ErrorRead := (IOresult <> 0);
end;

{ return 'false' if cannot open file to write }
function TSendFaxForm.ErrorWrite(var fi :FILE) :Boolean;
begin
   {$I-}
   ReWrite(fi,1);
   {$I+}
   ErrorWrite := (IOresult <> 0);
end;


{ convert 1 ASCII line(88 characters) into 20 scan lines(1 scan line =
  1728 pixels), 216 bytes is contain 1728 pixels (216*8=1728) }
procedure TSendFaxForm.AsciiToPel;
var       k : integer;
begin
  for row := 0 to 19 do
     begin
     k := ChCount;
     for j := 0 to 19 do
         buf[row,j] := #0;
     x := 0;
     j := 20;
     while x <> 88 do
       begin
       case temp[k] of
        #209,#212,#213,#214,#215,#216,#217,#218,#231,#232,#233,#234,
        #235,#236,#237 :
          begin
          buf[row][j-2] :=
          Chr(Ord(buf[row][j-2]) or Ord(readfont[Ord(temp[k])][row][0]));
          buf[row][j-1] :=
          Chr(Ord(buf[row][j-1]) or Ord(readfont[Ord(temp[k])][row][1]));
          end;
        CR : while x<88 do
             begin
             for y := 0 to 1 do
                begin
```

```
                    buf[row,j] := #0;
                    inc(j);
                    end;
                y := 2;
                inc(x);
                end;
          else   { default case }
            begin
            for y := 0 to 1 do
                begin
                buf[row,j] := readfont[Ord(temp[k]), row, y];
                inc(j);
                end;
            inc(x);
            y := 2;
            end;    {end of else}
          end;      {end of case statement}
        inc(k);
        end;        {end of while x<>88}
      for j:=196 to 215 do
      buf[row,j] := #0;
      j:=216
      end;          {end of for row}
  ChCount := k;
  row := 20;
end;

{ read bitmap 1 scan line, call ToHuffman() to change black or white
  runlength into huffman one dimention, until 20 scan lines}
procedure TSendFaxForm.OneDimention;
var       pxl : array[0..1728] of integer;
          ix,change   : integer;
          DiRes,MoRes : integer;
          cond        : byte;
begin
   for row := 0 to 19 do
       begin
       Bfcount := 0;
       change  := 0;
       ToHuffman(EOL);
       ToFlag  := Wht;
       ToCount   := 0;
       for BufPos := 0 to PEL-1 do
           begin
           if buf[row,BufPos] = #0 then
             begin
              if ToFlag = Wht then
                 ToCount := ToCount + 8
              else
                 begin
                 pxl[change] := ToCount;
                 inc(change);
                 ToFlag := Wht;
                 ToCount := 8;
                 end;
              end
           else
              for BufBit := 0 to 7 do
                  begin
                  cond := Ord(buf[row,BufPos]) and bits[BufBit];
                  if cond <> 0 then
```

```
                    begin
                    if ToFlag = Wht then
                       begin
                       pxl[change] := ToCount;
                       inc(change);
                       ToFlag := Blk;
                       ToCount := 0;
                       end;
                    inc(ToCount);
                    end
                 else
                    begin
                    if ToFlag = Blk then
                       begin
                       pxl[change] := ToCount;
                       inc(change);
                       ToFlag := Wht;
                       ToCount := 0;
                       end;
                    inc(ToCount);
                    end;
                 end;
           end;
        BufPos := PEL;
        pxl[change] := ToCount;
        inc(change);
        ToFlag := Wht;
        for ix := 0 to change-1 do
           begin
           if pxl[ix] <= 63 then
              ToHuffman(Terminate[pxl[ix],ToFlag])
           else
              begin
              ToCount := pxl[ix];
              j := 0;
              repeat
                DiRes := ToCount div ((j+1)* 64);
                MoRes := ToCount mod ((j+1)* 64);
                inc(j);
              until not ( (DiRes<>1) or (MoRes>63) );
              ToHuffman(MakeUp[j-1,ToFlag]);
              ToCount := MoRes;
              ToHuffman(Terminate[ToCount,ToFlag]);
              end; {else}
           if ToFlag = Blk then
              ToFlag := Wht
              else  ToFlag := Blk;
           end;

        if BfCount < 28 then
           begin
           while BfCount <> 28 do
             ToHuffman(ZeroCode);
           end;
        end;
     row := 20;
end;

{ convert line of pixel into huffman code, depend on black/white
  runlength }
procedure TSendFaxForm.ToHuffman(var AnyCode :HuffmanCode);
```

```
var       cond  : word;
          ToChr : byte;
begin
   for CodeBit:= 0 to AnyCode.cnt-1 do
       begin
       cond := AnyCode.code and ToCompare[CodeBit];
       if cond <> 0 then
          begin
          ToChr := Ord(WriteBf[WriteBufPos]) or bits[WriteBufBit];
          WriteBf[WriteBufPos] := Chr(ToChr);
          end;
       inc(WriteBufBit);
       if WriteBufBit = 8 then
         begin
         WriteBf[WriteBufPos] := Chr(inverse[ord(WriteBf[WriteBufPos])]);
         WriteBufBit := 0;
         inc(WriteBufPos);
         inc(BfCount);
         if WriteBufPos = BfSize then
            begin
            BlockWrite(OutputFile,WriteBf,BfSize,Wremain);
            for WriteBufPos := 0 to Bfsize-1 do
                WriteBf[WriteBufPos] := #0;
            WriteBufPos := 0;
            end;
         end;
       end;
   inc(CodeBit);
end;
```

```
end;
```

โปรแกรม UWAITFAX.PAS

```
unit Uwaitfax;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, Buttons, MSComm, UConfig,
  UBitMap;

type
   TreePtr     = ^pTree;
   pTree       = record
      bit      : byte;
      cnt      : integer;
      Left     : TreePtr;
      Right    : TreePtr;
               end;
      NODE     = pTree;
type
  TWaitFaxForm = class(TForm)
    WaitFaxMemo: TMemo;
    OKWaitBtn: TBitBtn;
    CloseWaitBtn: TBitBtn;
    WaitFaxConfigBtn: TButton;
    EditShow: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    MSCommWait: TMSComm;
    ViewFaxBtn: TButton;
    FaxOpenDialog: TOpenDialog;
    DirName: TEdit;
    Label4: TLabel;
    CheckBox1: TCheckBox;
    EditAddNo: TEdit;
    AddBtn: TButton;
    procedure FormCreate(Sender: TObject);
    procedure WaitFaxConfigBtnClick(Sender: TObject);
    procedure OKWaitBtnClick(Sender: TObject);
    procedure CloseWaitBtnClick(Sender: TObject);
    procedure ViewFaxBtnClick(Sender: TObject);
    procedure AddBtnClick(Sender: TObject);
  private
    { Private declarations }
    procedure DoCommTransmitLow(Sender: TObject; Count: Word);
    procedure DoCommError(Sender: TObject; CommErr: Word; ErrStr:
              String);
    procedure DoCommCommEvent(Sender: TObject; CommEvent:
              TCommEvents);
    procedure DoCommReceive(Sender: TObject; Count: Word);
    procedure WaitForResp(rsp: String; tm: LongInt; param: integer);
    procedure Sleep(stm: LongInt);
    procedure ATcommand(cmmd: String);

    procedure initComm;
    procedure initCommVariable;
```

```
    procedure CommOpen;
    procedure CommClose;
    procedure ModemError;
    procedure QuitSerialComm;
    procedure initWaitFaxModem;
    procedure ToReceive;
    procedure WaitForRing;
    procedure Negotiate;
    procedure ReceiveFax;
    procedure SaveToHuffFile(Num: integer);
    procedure EjectSome;
    function  Stop : Boolean;
    function  GetModemChar(var ModemChar:Char):Boolean;

  public
    { Public declarations }
  end;

var
  WaitFaxForm: TWaitFaxForm;
  FaxFile : String;

implementation

{$R *.DFM}
{$I declare.pas}
{$I cheese.pas}
{$I view.pas}

procedure TWaitFaxForm.FormCreate(Sender: TObject);
begin
  WaitFaxMemo.Lines.Clear;
  initComm;
  AddBtn.Enabled := False;
  DirName.Text := 'c:\faxsend';
end;

procedure TWaitFaxForm.WaitFaxConfigBtnClick(Sender: TObject);
begin
  ConfigForm.ShowModal; {see UConfig.pas}
  initComm;             {see cheese.pas}
end;

procedure TWaitFaxForm.OKWaitBtnClick(Sender: TObject);
begin
  CloseWaitBtn.Enabled := False;
  WaitFaxMemo.Lines.Clear;
  WaitFaxMemo.Lines.Add('with cheese.pas');
  initCommVariable;       { see cheese.pas }
  CommOpen;               { see cheese.pas }
  if ToError then
     begin
     CommClose;
     MessageDlg('Cannot open commport',mtError,[mbOk],0);
     end
     else
       begin
       initWaitFaxModem;        { see cheese.pas }
       WaitFaxMemo.Lines.Add('initial modem success');
```

```
        ToReceive;              { see cheese.pas }
        end;

  CloseWaitBtn.Enabled := True;

end;

procedure TWaitFaxForm.CloseWaitBtnClick(Sender: TObject);
begin
  Close;
end;

procedure TWaitFaxForm.ViewFaxBtnClick(Sender: TObject);
begin
  if FaxOpenDialog.Execute then
     begin
     FaxFile := FaxOpenDialog.FileName;
     SeeMe(FaxFile);
     end
     else exit;
  AddBtn.Enabled := True;
end;

procedure TWaitFaxForm.AddBtnClick(Sender: TObject);
const  Sz = 1024;
var    InFile,OutFile : File;
       buff : array [0..Sz-1] of char;
       remain : integer;
       i : integer;
       PhoneHeader : String;
begin
  AssignFile(InFile,faxFile);
  AssignFile(OutFile,ChangeFileExt(faxFile,'.aaa'));
  {$I-}
  reset(InFile,1);
  rewrite(OutFile,1);
  {$I+}
  if IOresult <> 0 then exit;

  for i := 1 to Sz-1 do
      buff[i] := #0 ;
  BlockWrite(OutFile,buff,30);

  while not EOF(InFile) do
    begin
    BlockRead(InFile,buff,sizeof(buff),remain);
    if remain >= 0 then
       BlockWrite(OutFile,buff,remain);
    end;
  closefile(InFile);
  DeleteFile(FaxFile);
  PhoneHeader := 'Fax Phone No ('+EditAddNo.Text+')';
  seek(OutFile,0);
  BlockWrite(OutFile,PhoneHeader[1],Length(PhoneHeader));
  CloseFile(OutFile);
  exit;
end;

end.
```

ไฟล์ CHEESE.PAS

```
{This file is full of functions that are used for waiting and
receiving fax
 file from fax machine}

const
  CommBufSize = 2048;

  CED = #13#10+'CED';
  OK = #13#10+'OK'+#13#10;
  CONNECT = #13#10+'CONNECT'+#13#10;
  RING = #13#10+'RING'+#13#10;
  NoCarrier = #13#10+'NO CARRIER'+#13#10;
  NoDialTone = #13#10+'NO DIALTONE'+#13#10;
  BUSY = #13#10+'BUSY'+#13#10;

  DLE = #16;
  ETX = #3;

  EOLdat : array[0..9] of byte =
($0,$80,$0,$08,$80,$0,$08,$80,$0,$80);

var
  CommBuf  : array[0..CommBufSize-1] of char;  {store data read from
                                                modem}
  Response : String;
  CommDataPos : Word;      {number of data income in port's buffer}
  CurrPos     : Word;      {point to data that is read from port's
                            buffer CurrPos is always <= CommDataPos}
  CommMsgCount : Word;
  TransRate : integer;
  ToError : boolean;
  ReceiveFile,HuffFile : File;
  ToSize : LongInt;

{read information from 'fax.ini', then set MSComm's parameter}
procedure TWaitFaxForm.initComm;
var   fi  : TextFile;
      Str : String;
      br  : String[6];
      len : byte;
begin
  ToError := False;
  AssignFile(fi,'fax.ini');
  Reset(fi);
  Readln(fi,Str);
  len := Pos(Str,'PORT = ');
  ToPort := StrToInt(copy(Str,len+8,1));

  case ToPort of
       1 : MSCommWait.Port := 1;
       2 : MSCommWait.Port := 2;
       3 : MSCommWait.Port := 3;
       4 : MSCommWait.Port := 4;
       end;

  Readln(fi,Str);
  len := Pos(Str,'Transfer Rate = ');
  ToRate := StrToInt(copy(Str,len+17,1));
```

```
    case ToRate of
         1 : begin
             MSCommWait.BaudRate := br2400;
             TransRate := 0;
             br := '2400';
             end;
         2 : begin
             MSCommWait.BaudRate := br4800;
             TransRate := 1;
             br := '4800';
             end;
         3 : begin
             MSCommWait.BaudRate := br9600;
             TransRate := 3;
             br := '9600';
             end;
         4 : begin
             MSCommWait.BaudRate := br14400;
             TransRate := 5;
             br := '14400';
             end;
        end;

    Readln(fi,Str);
    len := Pos(Str,'Resolution = ');
    ToResol := StrToInt(copy(Str,len+14,1));

    Readln(fi,Str);
    len := Pos(Str,'Retry = ');
    ToRetry := StrToInt(copy(Str,len+9,1));

    CloseFile(fi);

    { no parity, 8 data , 1 stop }
    MSCommWait.ParityBits := pbNone;
    MSCommWait.DataBits := dbEight;
    MSCommWait.StopBits := sbOne;
    MSCommWait.FlowControl := fcNone;

    {for send fax function}
    WaitFaxMemo.Lines.Add('>>MSComm Ready<<');
    EditShow.Text := 'com'+IntToStr(MSCommWait.port)+','+br+
                     ',n,8,1';
end;

{if data come to MSComm (number of count), Read the data}
procedure TWaitFaxForm.DoCommReceive(Sender: TObject; Count: Word);
var   i  : Word;
      ch : char;
      StrRes : String;
begin
  for i:=1 to Count do
      begin
      MSCommWait.Read(@ch, SizeOf(ch));
      CommBuf[CommDataPos] := ch;
      inc(CommDataPos);
      if CommDataPos = CommBufSize then
         CommDataPos := 0;
      StrRes := StrRes+ch;
```

```
      end;
end;

procedure TWaitFaxForm.DoCommTransmitLow(Sender: TObject; Count:
Word);
begin
  WaitFaxMemo.Lines.Add('>>OnTransmitLow<<');
end;

procedure TWaitFaxForm.DoCommError(Sender: TObject; CommErr: Word;
ErrStr: String);
begin
  WaitFaxMemo.Lines.Add('>>OnCommError<<');
end;

procedure TWaitFaxForm.DoCommCommEvent(Sender: TObject; CommEvent:
TCommEvents);
begin
  WaitFaxMemo.Lines.Add('>>OnCommEvent<<');
end;

procedure TWaitFaxForm.initCommVariable;
begin
  CommMsgCount := 0;
  ToError := False;
end;

{open MSComm, and set procedures for each event of MSComm}
procedure TWaitFaxForm.CommOpen;
var   temp : array[0..254] of char;
begin
  if not MSCommWait.Open then
     begin
     Application.MessageBox(StrPCopy(temp,MSCommWait.GetError),
                            'Cannot open comm port',mb_iconstop);
     ToError := True;
     end
     else begin
          ToError := False;
          WaitFaxMemo.Lines.Add('>>success in opening port');
          with MSCommWait do
               begin
               OnReceive := DoCommReceive;
               OnTransmitLow := DoCommTransmitLow;
               OnError := DoCommError;
               OnCommEvent := DoCommCommEvent;
               end;
          end;
end;

procedure TWaitFaxForm.CommClose;
begin
  MSCommWait.Close;
end;

{sent command to modem in time of milliseconds}
procedure TWaitFaxForm.ATcommand(cmmd: String);
begin
  MSCommWait.Write(@cmmd[1], Length(cmmd));
```

```
end;

{wait for response from modem, if no response in time then report
error}
procedure TWaitFaxForm.WaitForResp(rsp: String; tm: LongInt; param:
integer);
var   start : LongInt;
      i     : integer;
begin
  Response := '';
  i := 0;
  start := GetTickCount;
  while true do
    begin
    if CurrPos <> CommDataPos then
       begin
       Response := Response + CommBuf[CurrPos];
       inc(i);
       inc(CurrPos);
       if CurrPos = CommBufSize then
          CurrPos := 0;
       if Pos(rsp,Response)<>0 then
          begin
          ToError := False;
          WaitFaxMemo.Lines.Add('(w) '+Response);
          exit;
          end;
       start := GetTickCount;
       end;
    if (GetTickCount-start) > tm then
       begin
       if param=0 then
          begin
          QuitSerialComm;
          MessageDlg('No response from modem.',mtError,[mbOk],0);
          Exit;
          end
       else begin    {param = 1}
            ToError := True;  {fail}
            Exit;
            end;
       end;
    Application.ProcessMessages;
    if Stop then
       exit;
    end;  { end of while }
end;

{hang up the phone, close MSComm}
procedure TWaitFaxForm.QuitSerialComm;
var   txt : String;
      i   : word;
begin
  for i:=0 to 59 do
      Response[i] := #0;
  for i:=0 to 59 do
      CommBuf[i] := #0;
  ATcommand('+++');
```

```
   WaitForResp(OK,5000,1);
   txt := 'ATH0'+#13;

   ATcommand(txt);
   WaitForResp(OK,5000,1);

   if ToError then
      begin
      exit;
      end;
   txt := 'ATZ'+#13;
   ATcommand(txt);
   WaitforResp(OK,5000,1);
   if ToError then
      begin
      exit;
      end;
   WaitFaxMemo.Lines.Add('YaHoo !!!');
   MSCommWait.Close;
end;

{delay time in milliseconds}
procedure TWaitFaxForm.Sleep(stm: LongInt);
var   t : LongInt;
begin
   t := GetTickCount;
   while (t+stm) >= GetTickCount do
         Application.ProcessMessages;
end;

{send ATcommand set to initial modem before sending fax}
procedure TWaitFaxForm.initWaitFaxModem;
var   cmmd : String;
begin
   WaitFaxMemo.Lines.Add('>init fax modem to wait');
   cmmd := 'ATZ'+#13;
   ATcommand(cmmd);
   WaitForResp(OK,5000,1);
   if ToError then
      begin
      ModemError;
      Exit;
      end;

   sleep(800);
   cmmd := 'ATH&D2E0X0'+#13;
   ATcommand(cmmd);
   WaitForResp(OK,5000,1);
   if ToError then
      begin
      ModemError;
      Exit;
      end;

   cmmd := 'AT+FCLASS=2'+#13;
   ATcommand(cmmd);
   WaitForResp(OK,5000,1);
   if ToError then
      begin
```

```
      ModemError;
      Exit;
      end;

   cmmd := 'AT+FLID=""'+#13;
   ATcommand(cmmd);
   WaitForResp(OK,5000,1);
   if ToError then
      begin
      ModemError;
      Exit;
      end;

   cmmd := 'AT+FDCC=1,5,0,2,0,0,0,0'+#13;
   ATcommand(cmmd);
   WaitForResp(OK,5000,1);
   if ToError then
      begin
      ModemError;
      close;
      end;

   cmmd := 'AT+FAA=0;+FCR=1'#13;
   ATcommand(cmmd);
   WaitForResp(OK,5000,1);
   if ToError then
      begin
      ModemError;
      Exit;
      end;
end;

{if no request response from modem then report error}
procedure TWaitFaxForm.ModemError;
begin
  QuitSerialComm;
  MessageDlg('No Request Response',mtError,[mbOk],0);
  MessageDlg('Close Application !!!',mtWarning,[mbOk],0);
end;

{after connect to fax machine, send document to fax machine}
procedure TWaitFaxForm.ToReceive;
var   count : integer;
begin
  WaitFaxMemo.Lines.Add('>prepare to receive fax');
  AssignFile(ReceiveFile,'temp.fax');
  {$I-} ReWrite(ReceiveFile,1); {$I+}
  if IOresult<>0 then
    begin
    MessageDlg('Cannot open temp files',mtError,[mbOK],0);
    CloseFile(ReceiveFile);
    Exit;
    end;

  {begin send fax}
  count := 0;
  WaitFaxMemo.Lines.Add('>waiting ring');
  WaitForRing;
  if ToError or Stop then
```

```
    begin
    WaitFaxMemo.Lines.Add('(e)error or stop');
    CloseFile(ReceiveFile);
    Exit;
    end;
  WaitFaxMemo.Lines.Add('>ring is coming');
  Negotiate;
  if ToError then
    begin
    WaitFaxMemo.Lines.Add('(e)error');
    CloseFile(ReceiveFile);
    Exit;
    end;
  if Stop then
    begin
    WaitFaxMemo.Lines.Add('(e)user break');
    CloseFile(ReceiveFile);
    Exit;
    end;
  ReceiveFax;
  CloseFile(ReceiveFile);

  {reverse received data, save as 'fax00.fax','fax01.fax' and so on}
  SaveToHuffFile(count);
  DeleteFile('temp.fax');
end;

{ wait until there is ringing ,then back off the procedure }
procedure TWaitFaxForm.WaitForRing;
begin
  Response := '';
  while true do
    begin
    if CurrPos <> CommDataPos then
       begin
       Response := Response + CommBuf[CurrPos];
       inc(CurrPos);
       if CurrPos = CommBufSize then
          CurrPos := 0;
       if Pos(RING,Response)<>0 then
          begin
          ToError := False;
          WaitFaxMemo.Lines.Add('(r) '+Response);
          exit;
          end;
       end;
    if Stop then
      begin
      WaitFaxMemo.Lines.Add('(e)stop');
      Exit;
      end;
    Application.ProcessMessages;
    end;
end;

procedure TWaitFaxForm.Negotiate;
var cmd : String;
begin
  WaitFaxMemo.Lines.Add('');
```

```
    WaitFaxMemo.Lines.Add('>Fax Negotiation');

    ATcommand('ATA'+#13);
    WaitForResp(OK,30000,0);
    if ToError then
       begin
       ModemError;
       end;
    if Stop then
       exit;

    cmd := 'AT+FDR'+#13;
    ATcommand(cmd);
    WaitForResp(CONNECT,30000,0);
    if ToError then
       begin
       ModemError;
       exit;
       end;
    WaitFaxMemo.Lines.Add('>Fax connection');
  end;

  procedure TWaitFaxForm.EjectSome;
  var k: char;
  begin
    while True do {wait for FFh}
      begin
      Application.ProcessMessages;
      if GetModemChar(k) and (k = Chr($FF)) then
        break;
      if Stop then
        begin
        WaitFaxMemo.Lines.Add('0xFF not found');
        Exit;
        end;
      end;
    while True do {wait for 00h}
      begin
      Application.ProcessMessages;
      if GetModemChar(k) and (k = Chr(0)) then
        break;
      if Stop then
        begin
        WaitFaxMemo.Lines.Add('0x00 not found');
        Exit;
        end;
      end;
  end;

  procedure TWaitFaxForm.ReceiveFax;
  var   i,ix,iy : integer;
        txt : String;
        ForWrite : array[0..CommBufSize-1] of char;
        start: LongInt;
        k: Char;

  begin
    txt := #18; { code DC2 to tell fax to begin send data }
```

```
   ATcommand(txt);

   ix := 0;
   ToSize := 0;

   EjectSome;   {neglect some data of FFh}

   {begin receive real data from fax machine}
   {receive data until DLE(10h) ETX(03h) it will be end of data from
fax}
   i := 0;
   iy := 0;
   while True do
     begin
     if Stop then
       begin
       WaitFaxMemo.Lines.Add('(e)user break');
       Exit;
       end;

     if GetModemChar(k) then
        begin
        if iy = 1 then
           begin
           if k=ETX then break;
           iy := 0;
           if k<>DLE then continue;
           end
        else
           if k=DLE then
             begin iy := 1; continue; end;
        ForWrite[i] := k; i := i+1;
        ToSize := ToSize + 1;
        if i = CommBufSize then
           begin
           WaitFaxMemo.Lines.Add('>write '+IntToStr(CommBufSize)+'
bytes.');
           BlockWrite(ReceiveFile, ForWrite, CommBufSize);
           i := 0;
           end;
        end;
     Application.ProcessMessages;
     end;

     if i < CommBufSize then
       begin
       WaitFaxMemo.Lines.Add('>write '+IntToStr(i)+' byte(s)');
       BlockWrite(ReceiveFile,ForWrite,i);
       end;

     {after receive DLE and ETX}
     WaitForResp(OK,20000,0);
     if ToError then
        begin
        ModemError;
        Exit;
        end;
     sleep(1500);
```

```
    if Stop then Exit;

    ATcommand('AT+FDR'+#13);
    WaitForResp(OK,20000,0);
    if ToError then
       begin
       WaitFaxMemo.Lines.Add('(e)wait AT+FDR'); {Nobita}
       ModemError;
       end;
    QuitSerialComm;
end;

{first byte fill with ZERO value byte and then swap bits following
byte util found that Buf_Pos from receive file less than 100 bytes.
After that find EOL codes for 2 times appearance then it will be end
of data file and fill with 4 EOL codes and End of convert receive
data file to Huffman files.}
procedure TWaitFaxForm.SaveToHuffFile(Num : integer);
var   tempName : String;
      i, ix, iy, remain: integer;
      Fi : File;
      ch : array[0..CommBufSize-1] of char;
      ToRead : array[0..CommBufSize-1] of byte;
      FiPos : LongInt;
      Txt : String;
      len : integer;
begin
  FiPos := 0;
  tempName := DirName.Text+'\fax0'+IntToStr(Num)+'.huf';
  AssignFile(HuffFile,tempName);
  AssignFile(Fi,'temp.fax');
  {$I-}  ReWrite(HuffFile,1);
         ReSet(Fi,1);
  {$I+}
  if IOresult <> 0 then
     begin
     MessageDlg('Cannot save Huffman files',mtError,[mbOK],0);
     CloseFile(HuffFile);
     exit;
     end
     else
    { add byte #00 30 bytes before write huffman code into HuffFile,
      after save HuffFile, and user view file, then replace these #00
      with 'Fax Phone No (...)' , so file has
         1.'Fax Phone No (...)  (30 byte)
         2.'HUFFMAN_FILE'       (26 byte)
         3.inverse byte of huffman codes
      in program 'SendFax', remove header 30 bytes }

       begin
       seek(Fi, 0);
       for ix:=0 to CommBufSize-1 do
           ch[ix] := #0;
      BlockWrite(HuffFile, ch, 30);

       i := 0;
       ix := 0;
       while not Eof(Fi) do
          begin
```

```
          blockread(Fi,ToRead,CommBufSize,remain);
          for iy := 0 to (remain-1) do
              begin
              case ToRead[i] of
                   $80,$40,$20,$10,$08,$04,$02,$01 : inc(ix);
                   $00 : ix := 1;
                   else ix :=0;
                   end;
              inc(FiPos);
              if ix = 2 then
                 begin
                 if (ToSize-FiPos) < 100 then
                    break;
                 ix := 0 ;
                 end;
              ch[i] := Chr(inverse[ToRead[iy]]);   {see inverse[] in
                                                    declare.pas}
              inc(i);
              if i = CommBufSize then
                 begin
                 i := 0;
                 BlockWrite(HuffFile,ch,CommBufSize);
                 end;
             end; end of for}
          if ix = 2 then
             break;
          end;
        if i > 0 then
           BlockWrite(HuffFile,ch,i);
        BlockWrite(HuffFile,EOLdat,10);
        CloseFile(Fi);
        Txt := 'HUFFMAN_FILE';
        len := length(Txt);
        seek(HuffFile,0);
        BlockWrite(HuffFile,Txt[1],len);
        CloseFile(HuffFile);
        WaitFaxMemo.Lines.Add('>convert '+tempName);

        end;
end;

function TWaitFaxForm.Stop:Boolean;
  begin
  Result := CheckBox1.Checked;
  end;

function TWaitFaxForm.GetModemChar(var ModemChar:Char):Boolean;
  begin
  if CommDataPos = CurrPos then
    begin
    Result := False;
    end
  else
    begin
    ModemChar := CommBuf[CurrPos];
    inc(CurrPos);
    if CurrPos = CommBufSize then
      CurrPos := 0;
    Result := True;
```

```
    end;
end;
```

ไฟล์ VIEW.PAS

```
{ this program is used to display fax data(huffman code that is reversed
  every byte from MSB->LSB }

const   PEL          = 216;
        Blk          = 1;
        Wht          = 0;
        BufferSize   = 4320;
        NO           : Boolean = False;
        YES          : Boolean = True;

   Terminate : array[0..64,0..1] of HuffCode =
            (((bit: 8;code:$3500), (bit:10;code:$0D0C)),
            ((bit: 6;code:$1C00), (bit: 3;code:$4000)),
            ((bit: 4;code:$7000), (bit: 2;code:$C000)),
            ((bit: 4;code:$8000), (bit: 2;code:$8000)),
            ((bit: 4;code:$B000), (bit: 3;code:$6000)),
            ((bit: 4;code:$C000), (bit: 4;code:$3000)),
            ((bit: 4;code:$E000), (bit: 4;code:$2000)),
            ((bit: 4;code:$F000), (bit: 5;code:$1800)),
            ((bit: 5;code:$9800), (bit: 6;code:$1400)),
            ((bit: 5;code:$A000), (bit: 6;code:$1000)),
            ((bit: 5;code:$3800), (bit: 7;code:$0800)),
            ((bit: 5;code:$4000), (bit: 7;code:$0A00)),
            ((bit: 6;code:$2000), (bit: 7;code:$0E00)),
            ((bit: 6;code:$0C00), (bit: 8;code:$0400)),
            ((bit: 6;code:$D000), (bit: 8;code:$0700)),
            ((bit: 6;code:$D400), (bit: 9;code:$0C00)),
            ((bit: 6;code:$A800), (bit:10;code:$05C0)),
            ((bit: 6;code:$AC00), (bit:10;code:$0600)),
            ((bit: 7;code:$4E00), (bit:10;code:$0200)),
            ((bit: 7;code:$1800), (bit:11;code:$0CE0)),
            ((bit: 7;code:$1000), (bit:11;code:$0D00)),
            ((bit: 7;code:$2E00), (bit:11;code:$0D80)),
            ((bit: 7;code:$0600), (bit:11;code:$06E0)),
            ((bit: 7;code:$0800), (bit:11;code:$0500)),
            ((bit: 7;code:$5000), (bit:11;code:$02E0)),
            ((bit: 7;code:$5600), (bit:11;code:$0300)),
            ((bit: 7;code:$2600), (bit:12;code:$0CA0)),
            ((bit: 7;code:$4800), (bit:12;code:$0CB0)),
            ((bit: 7;code:$3000), (bit:12;code:$0CC0)),
            ((bit: 8;code:$0200), (bit:12;code:$0CD0)),
            ((bit: 8;code:$0300), (bit:12;code:$0680)),
            ((bit: 8;code:$1A00), (bit:12;code:$0690)),
            ((bit: 8;code:$1B00), (bit:12;code:$06A0)),
            ((bit: 8;code:$1200), (bit:12;code:$06B0)),
            ((bit: 8;code:$1300), (bit:12;code:$0D20)),
            ((bit: 8;code:$1400), (bit:12;code:$0D30)),
            ((bit: 8;code:$1500), (bit:12;code:$0D40)),
            ((bit: 8;code:$1600), (bit:12;code:$0D50)),
            ((bit: 8;code:$1700), (bit:12;code:$0D60)),
            ((bit: 8;code:$2800), (bit:12;code:$0D70)),
            ((bit: 8;code:$2900), (bit:12;code:$06C0)),
            ((bit: 8;code:$2A00), (bit:12;code:$06D0)),
            ((bit: 8;code:$2B00), (bit:12;code:$0DA0)),
            ((bit: 8;code:$2C00), (bit:12;code:$0DB0)),
            ((bit: 8;code:$2D00), (bit:12;code:$0540)),
            ((bit: 8;code:$0400), (bit:12;code:$0550)),
            ((bit: 8;code:$0500), (bit:12;code:$0560)),
            ((bit: 8;code:$0A00), (bit:12;code:$0570)),
            ((bit: 8;code:$0B00), (bit:12;code:$0640)),
            ((bit: 8;code:$5200), (bit:12;code:$0650)),
            ((bit: 8;code:$5300), (bit:12;code:$0520)),
```

```
            ((bit: 8;code:$5400), (bit:12;code:$0530)),
            ((bit: 8;code:$5500), (bit:12;code:$0240)),
            ((bit: 8;code:$2400), (bit:12;code:$0370)),
            ((bit: 8;code:$2500), (bit:12;code:$0380)),
            ((bit: 8;code:$5800), (bit:12;code:$0270)),
            ((bit: 8;code:$5900), (bit:12;code:$0280)),
            ((bit: 8;code:$5A00), (bit:12;code:$0580)),
            ((bit: 8;code:$5B00), (bit:12;code:$0590)),
            ((bit: 8;code:$4A00), (bit:12;code:$02B0)),
            ((bit: 8;code:$4B00), (bit:12;code:$02C0)),
            ((bit: 8;code:$3200), (bit:12;code:$05A0)),
            ((bit: 8;code:$3300), (bit:12;code:$0660)),
            ((bit: 8;code:$3400), (bit:12;code:$0670)),
              ((bit:12;code:$0010), (bit:12;code:$0010)));

   MakeUp : array[0..26,0..1] of HuffCode =
              (((bit: 5; code:$D800), (bit:10; code:$03C0)),
              ((bit: 5; code:$9000), (bit:12; code:$0C80)),
              ((bit: 6; code:$5C00), (bit:12; code:$0C90)),
              ((bit: 7; code:$6E00), (bit:12; code:$05B0)),
              ((bit: 8; code:$3600), (bit:12; code:$0330)),
              ((bit: 8; code:$3700), (bit:12; code:$0340)),
              ((bit: 8; code:$6400), (bit:12; code:$0350)),
              ((bit: 8; code:$6500), (bit:13; code:$0360)),
              ((bit: 8; code:$6800), (bit:13; code:$0368)),
              ((bit: 8; code:$6700), (bit:13; code:$0250)),
              ((bit: 9; code:$6600), (bit:13; code:$0258)),
              ((bit: 9; code:$6680), (bit:13; code:$0260)),
              ((bit: 9; code:$6900), (bit:13; code:$0268)),
              ((bit: 9; code:$6980), (bit:13; code:$0390)),
              ((bit: 9; code:$6A00), (bit:13; code:$0398)),
              ((bit: 9; code:$6A80), (bit:13; code:$03A0)),
              ((bit: 9; code:$6B00), (bit:13; code:$03A8)),
              ((bit: 9; code:$6B80), (bit:13; code:$03B0)),
              ((bit: 9; code:$6C00), (bit:13; code:$03B8)),
              ((bit: 9; code:$6C80), (bit:13; code:$0290)),
              ((bit: 9; code:$6D00), (bit:13; code:$0298)),
              ((bit: 9; code:$6D80), (bit:13; code:$02A0)),
              ((bit: 9; code:$4C00), (bit:13; code:$02A8)),
              ((bit: 9; code:$4C80), (bit:13; code:$02D0)),
              ((bit: 9; code:$4D00), (bit:13; code:$02D8)),
              ((bit: 6; code:$6000), (bit:13; code:$0320)),
              ((bit: 9; code:$4D80), (bit:13; code:0328)));

   ToCompare  : array[0..15] of word  = ($8000,$4000,$2000,$1000,$800,
                        $400,$200,$100,$80,$40,$20,$10,8,4,2,1);

   bits       : array[0..7]  of byte  = ($80,$40,$20,$10,8,4,2,1);

var     ToWrite,ToRead          : array[0..BufferSize-1] of char;
        sourceFile,tempFile     : FILE;
        WritePos,WriteBit,BufBitCount,TotalBit,CodeIndex : integer;
        BufferPos               : integer;
        ToCode                  : Word;
        CountCode, FlagCode     : Word;
        ReadRemain              : integer;
        WhiteHead,BlackHead,base : TreePtr;
        ToCheck                   : Boolean;

{ check header of file, if not correct then cannot decode }
function IsHuffman : Boolean;
var    BufStr : String;
begin
  BlockRead(sourceFile,BufStr[1],20);
```

```
  BufStr[0] := Chr(20);
  if (pos('HUFFMAN_FILE',BufStr) or pos('BIT FAX',BufStr)) = 0 then
     begin
     MessageDlg('Not a Huffman File',mtError,[mbCancel],0);
     CloseFile(sourceFile);
     CloseFile(tempFile);
     Result := false;
     exit;
     end;
  Result := true;
end;

procedure CreatList(var p:TreePtr);
begin
  new(p);
  p^.bit := $FF;
  p^.cnt := -32768; { $FFFF }
  p^.Left := nil;
  p^.Right := nil;
end;

procedure Add_RightNode(var p:TreePtr);
var tmp : TreePtr;
begin
  CreatList(tmp);
  p^.Right := tmp;
end;

procedure Add_LeftNode(var p:TreePtr);
var tmp : TreePtr;
begin
  CreatList(tmp);
  p^.Left := tmp;
end;
{ check each bit in every byte, is it black or white }
procedure Check_Bit;
var  ch : char;
begin
  if ((ord(ToRead[BufferPos])) and (bits[BufBitCount])) <> 0 then
     ToCode := (ToCode or ToCompare[CodeIndex]);
  inc(BufBitCount);
  if BufBitCount = 8 then
     begin
     BufBitCount := 0;
     inc(BufferPos);
     if BufferPos = ReadRemain then
       begin
       BufferPos := 0;
       BlockRead(sourceFile,ToRead,BufferSize,ReadRemain);
       if ReadRemain = 0 then
          MessageDlg('Error in decode HUFFMAN CODE',mtError,[mbCancel],0)
       end;
  ToRead[BufferPos] := Chr(inverse[ord(ToRead[BufferPos])]);
  end;
end;

{ check is it EOL code }
procedure CheckEol;
begin
  ToCode := 0;
  for CodeIndex:=0 to 11 do
```

```
      begin
      Check_Bit;
      end;
  if ToCode <> EOLcode.code then
     while true do
       begin
       Check_Bit;
       inc(CodeIndex);
       if CodeIndex  = 16 then
          CodeIndex := 0;
       if ToCode > 0 then
          break;
       end;
  FlagCode := Wht;
end;

procedure Check_Code;
begin
  if (Ord(ToRead[BufferPos]) and bits[BufBitCount]) <> 0 then
     base := base^.Right
     else
      base := base^.Left;
  inc(BufBitCount);
  if BufBitCount = 8 then
     begin
     BufBitCount := 0;
     inc(BufferPos);
     if BufferPos = ReadRemain then
        begin
       BufferPos := 0;
       BlockRead(sourceFile,ToRead,BufferSize,ReadRemain);
       end;
     ToRead[BufferPos] := Chr(inverse[ord(ToRead[BufferPos])]);
     end;
end;

procedure GenTree(head:TreePtr; color:integer);
var   iCode,iBit,x : byte;
      index : TreePtr;
begin
  for iCode:=0 to 64 do
      begin
      index := head;
      for iBit:=0 to (Terminate[iCode,color].bit)-1 do
          begin
        if ((Terminate[iCode,color].code and ToCompare[iBit]))<>0 then
           begin
           if index^.Right = nil then
           Add_RightNode(index);
           index := index^.Right;
           end
           else
               begin
             if index^.Left = nil then
             Add_LeftNode(index);
             index := index^.Left;
             end;
        end;      { for iBit }
       inc(iBit);
       if iCode <> 64 then
          index^.cnt := iCode
```

```
        else
          index^.cnt := $0FFF;
        index^.bit := Terminate[iCode,color].bit;
        end;       { for iCode }
  for iCode:=0 to 26 do
        begin
        index := head;
        for iBit:=0 to MakeUp[iCode,color].bit-1 do
            begin
          if (MakeUp[iCode,color].code and ToCompare[iBit]) <> 0 then
             begin
             if index^.Right = nil then
             Add_RightNode(index);
             index := index^.Right;
             end
             else
               begin
               if index^.Left = nil then
               Add_LeftNode(index);
               index := index^.Left;
               end;
          end;      { for iBit }
        index^.cnt := (iCode+1)*64;
        index^.bit := MakeUp[iCode,color].bit;
        end;       { for iCode }
  inc(iCode);
end;

procedure InitHead;
begin
  CreatList(WhiteHead);
  CreatList(BlackHead);
  GenTree(WhiteHead,Wht);
  GenTree(BlackHead,Blk);
end;

{ convert black,white Huffman code into black and white bitmap }
procedure ConvertToBitmap;
var ix :integer;
begin
  if FlagCode = Wht then
     begin
     ix := 0;
     while (ix < CountCode) do
         begin
         inc(WriteBit);
         if WriteBit = 8 then
            begin
            WriteBit := 0;
            inc(WritePos);
            if WritePos = BufferSize then
               begin
               BlockWrite(tempFile,ToWrite,BufferSize);
               for WritePos := 0 to BufferSize-1 do
                   ToWrite[WritePos] := #0;
               WritePos := 0;
               end;
            end;
         inc(ix);
         end;
     FlagCode := Blk;
```

```
    end       { if FlagCode = Wht }
    else
      begin
      ix := 0;
      while (ix < CountCode) do
       begin
       ToWrite[WritePos]:=Chr(Ord(ToWrite[WritePos]) or bits[WriteBit]);
       inc(WriteBit);
       if WriteBit = 8 then
          begin
          WriteBit := 0;
          inc(WritePos);
          if WritePos = BufferSize then
             begin
             BlockWrite(tempFile,ToWrite,BufferSize);
             for WritePos := 0 to BufferSize-1 do
                 ToWrite[WritePos] := #0;
             WritePos := 0;
             end;
          end;
       inc(ix);
       end;      {while ix}
       FlagCode := Wht;
    end;    {else}
end;

procedure Check_HuffmanCode;
var  ch : char;
     tmp : TreePtr;
begin
  CountCode := 0;
  CodeIndex := 0;
  if FlagCode = Wht then
     tmp := WhiteHead
     else
      tmp := BlackHead;
  Base := tmp;
  while true do
    begin
    Check_Code;
    inc(CodeIndex);
    if CodeIndex >= 14 then
       begin
       MessageDlg('Error in decode HUFFMAN CODE',mtError,[mbOK],0);
       ToCheck := YES;
       TotalBit := 1728;
       exit;
       end;
    if (base^.cnt <> -1) and (base^.cnt <> 0) then
       if base^.bit = CodeIndex then
          begin
          CountCode := base^.cnt;
          if CountCode > 63 then
             begin
             if CountCode = $0FFF then
                begin
                TotalBit := 1728;
                ToCheck := YES;
                exit;
                end;
             CodeIndex := 0;
```

```
            base := tmp;
            while true do
              begin
              Check_Code;
              inc(CodeIndex);
              if base^.cnt <> -1 then
                 if base^.bit = CodeIndex then
                    begin
                    CountCode := CountCode + base^.cnt;
                    TotalBit := TotalBit + CountCode;
                    ConvertToBitmap;
                    exit;
                    end;
              end;    { while }
            end       { if count>63 }
            else
              begin
              TotalBit := TotalBit + CountCode;
              ConvertToBitmap;
              exit;
              end;
       end;           { if base^.bit }
    end;              { while true }
end;

{ main procedure, file's name is sent from UWaitFax.Pas to decode from
  reverse-bit huffman file, after decoding..data is in form of bitmap
  ans saved in temporary file 'seeme.bmp' then display on 'ViewForm' }
procedure SeeMe(name : String);
begin
  AssignFile(sourceFile, name);
  {$I-} Reset(sourceFile,1); {$I+}
  if IOResult <> 0 then
     begin
     MessageDlg('Error in opening '+name,mtError,[mbCancel],0);
     exit;
     end;

  AssignFile(tempFile,'seeme.bmp');  { temp file to save bitmap data }
  {$I-} ReWrite(tempFile,1); {$I+}
  if IOResult <> 0 then
     begin
     MessageDlg('Error in opening temp file',mtError,[mbCancel],0);
     CloseFile(sourceFile);
     exit;
     end;

  if not IsHuffMan then
     exit;

  { begin of huffman data }
  seek(sourceFile,26);
  for WritePos := 0 to BufferSize-1 do
      ToWrite[WritePos] := #0;
  WritePos := 0;

  initHead;
  WritePos := 0;
  WriteBit := 0;
  ToCode := 0;
  BufBitCount := 0;
  BufferPos := 0;
```

```
  TotalBit := 0;
  BlockRead(sourceFile,ToRead,BufferSize,ReadRemain);
  ToRead[BufferPos] := chr(inverse[ord(ToRead[BufferPos])]);

  CheckEOL;
  while true do
    begin
    Check_HuffmanCode;
    if TotalBit = 1728 then
       begin
       if ToCheck then
          break;
       CheckEol;
       TotalBit := 0;
       end;
    end;
  if WritePos < BufferSize then
     BlockWrite(tempFile,ToWrite,WritePos);

  CloseFile(sourceFile);
  CloseFile(tempFile);
  dispose(WhiteHead);
  dispose(BlackHead);

  { after reverse bits and decode data, save bitmap in 'seeme.bmp' ,then
    show them on 'ViewForm' }
  ViewForm.Image1.Visible := False;
  ViewForm.FileName := 'seeme.bmp';
  ViewForm.ShowModal;
  DeleteFile('seeme.bmp');
end;
```

โปรแกรม UBITMAP.PAS

```
unit Ubitmap;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;

type
  TViewForm = class(TForm)
    Image1: TImage;
    procedure FormCreate(Sender: TObject);
    procedure DisplayPic;
    procedure FormPaint(Sender: TObject);
  private
    { Private declarations }
    x,y  : Word;
    Buf  : array[0..49] of byte;
    Done : Word;
    fTxt : file;
  public
    { Public declarations }
    FileName : String;

  end;

var
  ViewForm: TViewForm;

implementation

{$R *.DFM}

procedure TViewForm.FormCreate(Sender: TObject);
var
  DotWidth : Word;
  i : Word;
begin
  Image1.Visible := False;
  DotWidth := 1;
  Canvas.Pen.Width := DotWidth;
  Canvas.Pen.Color := clBlack;
  ViewForm.Height := 200;
  ViewForm.Width := 400;
  Image1.Width:=1728;
  Image1.Height:=1143;
end;

{ read block of data (50 bytes) from files (seeme.pas) and check each
  bit, if bit is '1' then show dot, if bit is '0' then not-show, if
  reach 1728 bits then begin new line }
procedure TViewForm.DisplayPic;
var   i : Word;
begin
  MessageDlg('waiting',mtInformation,[mbOK],0);
  assignfile(fTxt,FileName);       {filename is 'seeme.bmp'}
  {$I-}  ReSet(fTxt,1);   {$I+}
  x:=0; y:=0;
```

```
while not Eof(fTxt) do
  begin
  BlockRead(fTxt,Buf,sizeof(Buf),Done);
  for i:=0 to done-1 do
      begin
      if x > 1720 then
         begin
         inc(y);
         x := 0;
         end;
       if (Buf[i] and 1) <> 0 then
           Image1.Canvas.Pixels[X+7,Y] := clBlack;
       if (Buf[i] and 2) <>0 then
           Image1.Canvas.Pixels[X+6,Y] := clBlack;
       if (Buf[i] and 4) <>0 then
          Image1.Canvas.Pixels[X+5,Y] := clBlack;
       if (Buf[i] and 8) <>0 then
          Image1.Canvas.Pixels[X+4,Y] := clBlack;
       if (Buf[i] and 16) <>0 then
          Image1.Canvas.Pixels[X+3,Y] := clBlack;
       if (Buf[i] and 32) <>0 then
          Image1.Canvas.Pixels[X+2,Y] := clBlack;
       if (Buf[i] and 64) <> 0 then
          Image1.Canvas.Pixels[X+1,Y] := clBlack;
       if (Buf[i] and 128) <>0 then
          Image1.Canvas.Pixels[X,Y] := clBlack;
       x := x+8
       end;
    end;
  closefile(fTxt);

end;

procedure TViewForm.FormPaint(Sender: TObject);
begin
  Image1.Visible := True;
  DisplayPic;
end;

end.
```

ภาคผนวก ข

ตารางแสดงความสัมพันธ์ระหว่างรหัสแอสกี้และรหัสภาษาไทย จะแทนที่รหัสภาษาไทยลงไปยังรหัสแอสกี้เดิมในส่วนที่เป็นรหัสที่ไม่ใช่อักษรภาษาอังกฤษ

		ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ	ญ	ฎ	ฏ
รหัสแอสกี้(ฐาน 16)	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	ฐ	ฑ	ฒ	ณ	ด	ต	ถ	ท	ธ	น	บ	ป	ผ	ฝ	พ	ฟ
รหัสแอสกี้(ฐาน16)	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
	ภ	ม	ย	ร	ฤ	ล	ฦ	ว	ศ	ษ	ส	ห	ฬ	อ	ฮ	ฯ
รหัสแอสกี้(ฐาน16)	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
	ะ		า													
รหัสแอสกี้(ฐาน16)	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
	เ	แ	โ	ใ	ไ	ๅ	็									
รหัสแอสกี้(ฐาน16)	E0	E1	E2	E3	D4	D5	D6	D7	D8	E9	EA	EB	EC	ED	EE	EF
	๐	๑	๒	๓	๔	๕	๖	๗	๘	๙						
รหัสแอสกี้(ฐาน16)	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

ภาคผนวก ค

คำสั่งแฟกซ์โมเด็มระดับ 2 ที่ปรากฏในโปรแกรม

Baxic "AT" Command Set

+++ ให้เปลี่ยนจากโหมดการส่งข้อมูล เป็นโหมดคำสั่ง และในขณะที่อยู่ในโหมดคำสั่ง ผู้ใช้สามารถติดต่อโดยตรงกับโมเด็มโดยใช้กลุ่มคำสั่ง "AT" และถ้าต้องการกลับสู่โหมดการส่งข้อมูลให้ใช้คำสั่ง AT0

ATA ให้เข้าสู่โหมดการตอบรับ

ATEn Command mode local echo of keyboard commands(กำหนดโหมดการแสดงผลจากการใช้คีย์บอร์ด)

n = 0 Echo off (ไม่ให้แสดงผลการใช้คีย์บอร์ด)

n = 1 Echo on (ให้แสดงผลการใช้คีย์บอร์ด)

ATHn ควบคุมสถานะการวางสาย

n = 0 ให้สร้างสถานะการวางสาย มีผลเหมือนคำสั่ง ATH

n = 1 ให้สร้างสถานะการยกแฮนด์เซ็ต

ATMn ควบคุมเสียงของโมเด็ม

n = 0 ไม่ให้แสดงเสียง

n = 1 ให้แสดงเสียงจนกระทั่งตรวจพบคลื่นพาห์

n = 2 ให้แสดงเสียงตลอดเวลา

n = 3 ให้แสดงเสียงหลังจากส่งเลขตัวสุดท้ายออกไป และให้หยุดเมื่อตรวจพบคลื่นพาห์

ATSr = n เซ็ตค่าให้รีจิสเตอร์ Sr ให้มีค่าตามกำหนด (n มีค่าตั้งแต่ 0 ถึง 255) ในโปรแกรมมีการกำหนดรีจิสเตอร์ดังต่อไปนี้

S0 = n กำหนดจำนวนการเรียก n (ring) ก่อนที่โมเด็มจะตอบรับโดยอัตโนมัติ แต่ถ้า n = 0 จะหมายถึง ไม่สามารถตอบรับโดยอัตโนมัติได้

S7 = n ตั้งเวลา(วินาที) สำหรับให้โมเด็มรอคลื่นพาห์ ค่าเริ่มต้นเป็น 60 วินาที

S8 = n ตั้งค่าหน่วงเวลา(วินาที) เมื่อตรวจพบสัญลักษณ์ "," ในการหมุนหมายเลขปลายทาง ค่าเริ่มต้นเป็น 0.02 วินาที และค่าหน่วงเวลาในการหยุดรอระหว่างคำสั่ง repeat (>)

ATVn Verbal/Numeric result codes(กำหนดโหมดการส่งผลตอบสนองจากโมเด็ม)

n = 0 ส่งผลตอบสนองในรูปตัวเลข

n = 1 ส่งผลตอบสนองในรูปข้อความ

สำหรับค่าของผลตอบสนองแสดงไว้ในตาราง ผ.1

ATXn กำหนดโหมดการแสดงผลตอบสนอง ดูได้จากตาราง ผ.1

n = 0-7

ATZ รีเซ็ตโมเด็ม

Result Code (ATVn)		ATXn							
ATV0	ATV1	X0	X1	X2	X3	X4	X5	X6	X7
0	OK	V	V	V	V	V	V	V	V
1	CONNECT	V	V	V	V	V			
2	RING	V	V	V	V	V	V	V	V
3	NOCARRIER	V	V	V	V	V	V	V	V
4	ERROR	V	V	V	V	V	V	V	V
5	CONNECT 1200								
6	NO DIAL TONE			V		V	V	V	V
7	BUSY				V	V	V	V	V
8	NO ANSWER				V	V	V	V	V
9	RINGING				V	V	V	V	V

NOTE: ตารางนี้กล่าวถึงเฉพาะการแสดงผลตอบสนองเป็นตัวเลขหรือข้อความ และการกำหนดความสามารถของโมเด็มในการตอบสนองดังกล่าว

V หมายถึงความสามารถที่มีได้

ตาราง ผ.1 Result Code Options

Extended 'AT&" Command Set

&Cn กำหนดโหมดการตรวจจับคลื่นพาห์

n = 0 ให้มีการตรวจตลอดเวลา

n = 1 ให้มีการตรวจเฉพาะเมื่อมีโอกาสที่จะมีคลื่นพาห์เข้ามา

&Dn โหมดการตรวจสอบสัญญาณ DTR(Data Terminal Ready)

n = 0 ไม่ทำการตรวจสอบ โดยสมมติว่ามี DTR อยู่ตลอดเวลา

n = 1 ถ้า DTR เปลี่ยนจากสถานะ on เป็น off จะทำให้ไม่มีการหมุนหมายเลขใดๆ เกิดขึ้น

n = 2 ถ้า DTR มีสถานะ off จะสร้างสถานะการวางสาย

n = 3 ถ้า DTR มีสถานะ off จะสร้างสถานะการวางสาย และรีเซ็ตโมเด็ม

"AT+F" Command Set

+FAA (Select Fax Auto Answer Mode : โหมดตอบรับอัตโนมัติ)

รูปแบบ : AT+FAA = n

n = 0 ตอบรับเมื่อแฟกซ์โมเด็มเลือกใช้เฉพาะโหมดที่กำหนดโดยคำสั่ง +FCLASS

n = 1 ตรวจรับข้อมูลและการเรียกอัตโนมัติ และตอบรับตามความเหมาะสม

ค่าเริ่มต้น: +FAA = 0

ในโหมดตอบรับอัตโนมัติ โมเด็มจะปรับตัวเองโดยอัตโนมัติและตอบรับทันที

+FASERR (Fax T.30 Error Response: ผลแสดงความผิดพลาดตามมาตรฐานโทรสาร T.30)

รูปแบบ: AT+FAXERR = n

n คือ รหัสแสดงสถานะการวางสาย

แสดงถึง สาเหตุของการวางสาย และถูกกำหนดโดยโมเด็มที่อยู่ในระหว่างการสิ้นสุดในแต่ละขั้นตอน โมเด็มจะปรับให้พารามิเตอร์นี้เป็น 0 เมื่อเริ่มต้นแต่ละขั้นตอน ค่าของ n แสดงในตารางผลตอบสนอง

n	Class 1 Definition
	Call Placement and Termination
0	Normal and proper end of connection
1	Ring detect without successful handshake
2	Call aborted from +FK or <CAN>
3	No loop current
	Transmit Phase A and Miscellaneous Errors
10	Unspecified phase A error
11	No answer (T.30 T1 timeout)
	Transmit Phase B Hangup Codes
20	Unspecified phase B error
21	Remote cannot receive or send
22	Command received error in transmit phase B
23	Command received invalid command received
24	Response received error
25	DCS sent three times without response
26	DIS/DTC received three times;DCS not recoginized
27	Failure to train
28	Response received invalid response
	Transmit Phase C Hangup Codes
40	Unspecified phase C error
43	DTE to DCE data underflow
	Transmit Phase D Hangup Codes
50	Unspecified phase D error

ตารางที่ ผ.2 แสดงรหัสของการวางสาย

51	Response received error
52	No response to MPS repeated three times
53	Invalid response to MPS
54	No response to EOP repeated three times
55	Invalid response to EOP
56	No response to EOM repeated three times
57	Invalid response to EOM
58	Unable to continue after PIN or PIP
	Receive Phase B Hangup Codes
70	Unspecified receive phase B error
71	Response received error
72	Command received error
73	T.30 T2 timeout, expected page not received
74	T.30 T1 timeout after EOM received
	Received Phase C Hangup Codes
90	Unspecified receive phase C error
91	Missing EOL after 5 seconds
92	Unused code
93	DCE to DTE buffer overflow
94	Bad CRC or frame (ECM or BFT modes)
	Receive Phase C Hangup Codes
00	Unspecified receive phase D error
01	Response received invalid response received
02	Command received invalid response received
03	Unable to continue after PIN or PIP

ตารางที่ ผ.2 (ต่อ) แสดงรหัสของการวางสาย

+FHNG (Call Termination Statur Response : ผลตอบสนองสถานะการเรียกติดต่อ)

รูปแบบ : +FHNG : n

n คือรหัสในการวางสาย แสดงดังตารางที่ ผ.2 โดยผลตอบสนองนี้โมเด็มส่งให้กับ DTE เพื่อแสดงถึงสาเหตุในการวางสายและจะเก็บไว้ในพารามิเตอร์ +FAXERR ด้วย

+FCFR (Confinnation to Receive Response : การยืนยันการรับผลตอบสนอง

รูปแบบ : AT+FCFR

โมเด็มท้องถิ่น (iocal modem) จะส่งผลตอบสนองนี้ไปยัง DTE(หมายถึงเครื่องคอมพิวเตอร์) หลังจากได้รับสัญญาณ DCS และ TFC ที่ถูกต้องจากเครื่องโทรสารที่อยู่ไกลออกไป

+FCLASS (Service Class Identification and Control : แสดงลักษณะและการควบคุม)

รูปแบบ AT+FCLASS? เป็นการถามถึงกลุ่มของคำสั่งที่ใช้ในขณะนี้

AT+FCLASS=? เป็นการถามถึงความสามารถของโมเด็มในการใช้คำสั่งในระดับต่างๆ

AT+FCLASS=n เป็นการกำหนดระดับของคำสั่งที่ใช้ในโมเด็ม

คำสั่งนี้ใช้สำหรับการกำหนดและถามโหมดการทำงานของแฟกซ์โมเด็มและความสามารถซึ่งโมเด็มจะตอบกลับดังนี้

1 : EIA/TIA-578 ระดับ 1

2 : กำหนดโดยโรงงาน (SP-2386-A ระดับ 2)

3 : TIA/EIA-592 ระดับ 2.0

+FCON (Fax Connection Response : ผลตอบสนองการติดต่อของเครื่องโทรสาร)

รูปแบบ : +FCON

โมเด็มจะแสดงผลนี้เพื่อบอกให้รู้ว่า การติดต่อระหว่างเครื่องโทรสารประสบความสำเร็จโดยจะถูกส่งโดยโมเด็มท้องถิ่นไปยัง DTE หลังจากที่ได้รับแฟลก (flag) แรกของ HDLC

+FCR (Set Capability to Receive : การกำหนดความสามารถในการรับ)

รูปแบบ : AT+FCR = n

n = 0 โมเด็มจะไม่รับข้อมูลและไม่สามารถโพล (poll)

n = 1 โมเด็มจะสามารถรับข้อมูลข่าวสารได้

พารามิเตอร์นี้จะถูกพิจารณาในระหว่างเฟส A และ D

+FDCC = vr,br,wd,ln,df,ec,bf,st กำหนดความสามารถของโมเด็มในโหมดการรับข้อมูล สำหรับรหัสของพารามิเตอร์ย่อยในขบวนการสามารถดูได้จากตาราง ผ.3

+FDIS (Set Current Session Negotiation Parameter : การกำหนดพารามิเตอร์แสดงความสามารถของโมเด็มตามการตกลงกันในขบวนการ)

รูปแบบ : AT+FDIS = vr,br,wd,ln,df,ec,bf,st

คำสั่งนี้จะเป็นการให้ DTE(หมายถึงคอมพิวเตอร์) กำหนดค่าของ DIS เฟรมโดยตรงไปให้กับ DCE (หมายถึงโมเด็ม) ในโหมดการส่งข้อมูล รหัสของพารามิเตอร์ย่อยในขบวนการ T.30 แสดงในตารางที่ ผ.3

Label	Function	Values	Description
vr	vertical resolution	1	Fine, 196 lines per inch
		0	Normal, 98 lines per inch

ตารางที่ ผ.3 แสดงค่าพารามิเตอร์ย่อย ใน T.30

br	data transfer speed[2]	0	2400 bps, V.27ter
		1	4800 bps, V.27ter
		2[1]	7200 bps, V.29 or V.17
		3[1]	9600 bps, V.29 or V.17
		4[1]	12000 bps, V.33 or V.17
		5[1]	14400 bps, V.33 or V.17
wd	page width	0	1728 pixels in 215 mm.
		1[1]	2048 pixels in 255 mm.
		2[1]	2432 pixels in 303 mm.
		3[1]	1216 pixels in 151 mm.
		4[1]	864 pixels in 107 mm.
ln	page length	0	A4, 297 mm.
		1[1]	B4, 364 mm.
		2[1]	Unlimited length
df	data compression format	0	1-dimensional, modified Huffman
		1[1]	1-dimensional, modified Read
		2[1]	2-dimensional, uncompressed
		3[1]	2-dimensional, modified Read
ec	error correction	0	Disable ECM
		1[1]	Enable error correction mode, 64 bytes/frame
		2[1]	Enable error correction mode,256 bytes/frame
bf	binary file transfer	0	Disable binary file transfer
		1[1]	Enable binary file transfer
st	scan time per line	0	0 ms for both normal and fine resolution
		1	5 ms for both normal and fine resolution
		2	10 ms for normal, 5 ms for fine resolution
		3	10 ms for both normal and fine resolution
		4	20 ms for normal, 10 ms for fine resolution
		5	20 ms for both normal and fine resolution
		6	40 ms for normal, 20 ms for fine resolution
		7	40 ms for both normal and fine resolution

[1] Optional

[2] The encoding specitied for the DIS frame in CCITT T.30 noes not allow all speeds to be specitied exactly. The identihcation of some BR speeds will therefore be manufacturer-specific

ตารางที่ ผ.3 (ต่อ)แสดงค่าพารามิเตอร์ย่อย ใน T.30

+FDR (Receive Phase C Data : การรับข้อมูลในเฟส C)

รูปแบบ : AT+FDR

คำสั่งนี้เป็นการเริ่มต้นหรือการรับข้อมูลเฟส C ต่อไป โดยโมเด็มจะแสดงค่าพารามิเตอร์ที่ใช้ด้วย ID และ NSS เฟรมถ้าเป็นไปได้

+FDT (Transmit Phase C Data : การส่งข้อมูลในเฟส C)

รูปแบบ : AT+FDT[=] n n = พารามิเตอร์ย่อย df,br,wd,ln

ในเฟส B คำสั่งนี้จะบอกให้โมเด็มทำการติดต่อกับเครื่องโทรสารที่อยู่ห่างออกไปและส่ง DCS ไปด้วย ส่วนในเฟส C จะเป็นการเริ่มต้นการส่งข้อมูลหรือให้ส่งข้อมูลต่อไป ค่าของพารามิเตอร์ย่อยแสดงไว้ในตารางที่ ผ.3

+FET (End the Page of Document : สิ้นสุดหน้าหรือข้อมูล)

รูปแบบ : AT+FET = n n = 0-15

เป็นคำสั่งแสดงการสิ้นสุดหน้าหรือข้อมูลที่ทำการส่ง ถ้าอยู่ในโหมดไม่มีการควบคุมความผิดพลาด (non-error-control operation) โมเด็มจะส่ง RTC ไปให้กับเครื่องโทรสารที่อยู่ไกล และเข้าสู่เฟส D โดยการส่งรหัส post-page ตามข้อกำหนดใน มาตรฐาน T.30

+FET (Post-Page Message Response : ผลตอบสนองข้อความแสดง post-message)

รูปแบบ : +FET : n

ผลตอบสนองนี้สร้างโดยโมเด็มท้องถิ่น โดยสร้างตามข่าวสาร pcst rnessage ที่รับเครื่องโทรสารที่อยู่ห่างออกไป และคำสั่งนี้จะทำหลังจากที่ทำคำสั่ง +FDR เรียบร้อยแล้ว ค่า n แสดงดังตาราง ผ.4

PPM Code	T.30 Mnemonic	Description
0	[PPS-]MPS	Another page ncxt,same document
1	[PPS-]EOM	Another document next
2	[PPS-]EOP	No more pages or documcnts
3	PPS-NULL	Another partial page next
4	[PPS-]PRI-MPS	Another page.procedurc interrupt
5	[PPS-]PRI-EOM	Another document.procedure interrupt
6	[PPS-]PRI-EOP	All done,procedure interrupt

ตารางที่ ผ.4 แสดงค่าของ n

+ **FLID** (Set Local ID String : กำหนดค่า local ID string ที่จะใช้ใน TSI/CSI เฟรม)

รูปแบบ : AT+FLID = "s" s คือรหัสแอสกี้ 20 ตัว

เพื่อบอกให้แฟกซ์โมเด็มรู้ว่าตัวเองเป็นใคร การกำหนด local ID ต้องกำหนดในเครื่องหมาย " " ดังนั้น เครื่องหมาย "(แอสกี้ 34) จึงไม่สามารถเป็นส่วนหนึ่งของ ID string และรูปแบบของ ID string โดยปกติจะเป็นหมายเลขโทรศัพท์

AT+FLID = "" (null string) หมายถึงไม่มีการส่งเฟรม TSI หรือ CSI

+FPTS (Set Page Transfer Status : กำหนดสถานะการส่งข้อมูล)

รูปแบบ ; AT+FPTS = *ppr* *ppr* มีค่า 1-5

ค่าเริ่มต้น : ppt = 1

คำสั่งนี้จะกำหนดค่า post message ซึ่งจะต้องดูจาก (copy-checking) หรือสัญญาณแสดงคุณภาพ

Value	Mnemonic	Dcfinition
1	MCF	Page good
2	RTN	Page bad, Retrain requested.
3	RTP	Page good. Retrain requested.
4	PIN	Page bad. Interrupt requested.
5	PIP	Page good. Interrupt requested.

ตารางที่ ผ.5 แสดงค่าของ *ppr*

+FPTS (Receive Page Transfer Status Response : การรับผลตอบสนองสถานะของการส่งข้อมูล)

รูปแบบ : +FPTS : *ppr,lc*[,blc,cblc[,lbc]]

ppr : ค่าของข่าวสาร post-page ซึ่งกำหนดเพื่อให้ +FPIS ใช้ในการกำหนดคำสั่งแสดงสถานะการส่งข้อมูล

lc : ตำแหน่งของบรรทัด

blc : ตำแหน่งของบรรทัดที่เสีย

cblc : เรียงลำดับบรรทัดที่เสีย

lbc : ตำแหน่งของไบท์ที่เสียหาย

ผลตอบสนองนี้จะถูกส่งจากแฟกซ์โมเด็มไปยัง DTE เมื่อสิ้นสุดเฟส C เพื่อรายงานสถานะการส่งข้อมูล

+FPTS (Transmit Page Transfer Starus Status Response : การส่งผลตอบสนองสถานะการส่งข้อมูล)

รูปแบบ : +FPTS : *ppr* *ppr* : ค่าของข่าวสาร post-page ซึ่งกำหนดเพื่อให้ +FPTS ใช้ในการกำหนดคำสั่งแสดงสถานะการส่งข้อมูล

โมเด็มจะส่งผลตอบสนองนี้ไปยัง DTE เพื่อแสดงถึงคุณภาพและข่าวสาร post-page จากเครื่องโทรสารที่อยู่ห่างออกไป (remote fax machine)

ภาคผนวก ง

แสดงความหมายของคำสั่งเดลไฟโดยทั่วไป ที่ปรากฏอยู่ในโปรแกรม

Note : เพื่อให้สามารถอ่านความหมายได้รวดเร็ว จึงได้กำหนดรูปแบบในการอธิบายไว้ดังนี้

iden(v,n[,res])[:type] prog:unit

กล่าวคือ

- iden หมายถึงชื่อโพรซีเดอร์หรือฟังก์ชัน
- v หมายถึงตัวแปรตามแบบข้อมูลที่จะกล่าวในคำอธิบาย และจะใช้ st,f และ p เป็นตัวแปร สตริง ไฟล์ และพอยน์เตอร์ ตามลำดับ ในหัวข้อนั้นๆ
- n,res และคำอื่นๆ หมายความว่าจะเป็นตัวแปรหรือค่าคงตัวก็ได้ ตามแบบที่จะกล่าวในคำอธิบาย
- [ ] หมายความว่าคำที่อยู่ในเครื่องหมายนี้จะมีหรือไม่ก็ได้
- type แบบข้อมูลของฟังก์ชัน
- prog จะระบุว่าเป็น PROC หมายถึงโพรซีเดอร์ หรือ FUNC หมายถึงฟังก์ชัน
- unit หากคำนั้นอยู่ในยูนิตอื่นนอกจาก System จะบอกชื่อยูนิตนั้น (ซึ่งเมื่อนำคำนี้ไปใช้ในโปรแกรม จะต้องระบุชื่อยูนิตภายใต้หัวข้อ USES)

Absolute

เป็นวิธีการกำหนดชนิดของตัวแปรโดยให้มีการจองเนื้อที่หน่วยความจำตามที่ระบุ หรือ ให้ใช้หน่วยความจำเดียวกันกับตัวแปรอื่นก็ได้

AssignFile(f,st) Proc :

กำหนดให้ตัวแปรไฟล์ f (ตามแบบของไฟล์ที่กำหนด) แทนชื่อไฟล์ในดิสค์ st

BlockRead(f,v,cnt[,res]) Proc :

อ่านข้อมูลของไฟล์ไม่กำหนดแบบ f ให้แก่ตัวแปรอาร์เรย์ของอักขระ v เป็นจำนวน cnt ไบต์ ให้ผลคือจำนวนที่อ่านได้จริงแก่ตัวแปรเลขจำนวนเต็ม res

BlockWrite(f,b,cnt[,res]) Proc :

เขียนข้อมูลในตัวแปรอาร์เรย์ของอักขระ v ลงไฟล์ไม่กำหนดแบบ f เป็นจำนวน cnt ไบต์ ให้ผลคือจำนวนที่เขียนได้จริงแก่ตัวแปรเลขจำนวนเต็ม res

Break Proc :

ให้ออกจากลูป for, while หรือ repeat

ChDir(st) Proc :

เปลี่ยนไดเรกทอรีเป็นตามที่กำหนดในสตริง st หรือรวมทั้ง st เป็นไดรฟ์

Chr(n) : Char Func :

ให้ค่าซึ่งเป็นผลจากการแปลงค่าเลขจำนวนเต็มไบต์ n เป็นข้อมูลแบบอักขระ เช่น Chr(65) ให้ค่าเป็นอักขระ 'A'

CloseFile(f) Proc :

ปิดไฟล์ f ซึ่งต้องเปิดอยู่

Copy(st,p,cnt) : String Func :

ให้ค่าในสตริง st จากตำแหน่ง p ไปเป็นจำนวน cnt อักขระ

Dispose(p) Proc :

ให้ยกเลิกชั้นข้อมูลที่กำหนดขึ้นด้วย New ซึ่งตัวแปรพอยน์เตอร์ p ชี้อยู่ในขณะนั้น

EOF(f) : Boolean Func :

ให้ค่าความเป็นจริงเมื่อไฟล์พอยน์เตอร์ของไฟล์ f อยู่ที่ท้ายไฟล์ (สุดไฟล์)

Exit Proc :

ให้ออกจากโปรแกรมย่อยนี้ แต่ถ้าใช้ในโปรแกรมหลักจะยุติการทำงานของโปรแกรม

ExtractFileName(st) : String Func : SysUtils

ให้ค่าเป็นชื่อและส่วนขยายของไฟล์ ในตัวแปรสตริง st ซึ่งมีค่าเป็น path+filename

FileSize(f) : LongInt Func :

ให้ขนาดของไฟล์ f (ขนาดตามแบบข้อมูล เช่น ถ้าเป็นไฟล์ของอักขระจะเป็นจำนวนอักขระ ถ้าเป็นไฟล์ของเรคอร์ดจะเป็นขนาดของเรคอร์ด

GetDir(D,st) Proc :

ให้ค่าเป็นไดเรกทอรี่ของไดรฟ์ปัจจุบัน D แก่ตัวแปรสตริง stD มีค่าได้เป็น 0 หมายถึงไดรฟ์ปัจจุบัน 1 หมายถึงไดรฟ์ A 2 หมายถึงไดรฟ์ B 3 หมายถึงไดรฟ์ C

Inc(v[,n]) Proc :

เพิ่มค่าในตัวแปรแบบมีลำดับที่ v ขึ้น n ถ้าไม่กำหนดค่า n จะเพิ่มทีละ 1

IntToStr(n) : String Func :

เปลี่ยนเลขจำนวน n เป็นสตริงของตัวเลข เช่น st := IntToStr(126); หมายถึง ให้ค่าแก่ตัวแปรสตริง st = '126'

IOresult : Integer Func :

ให้ Dos error code เมื่อมีการดำเนินกรรมวิธีกับไฟล์ (I/O stream) ค่า 0 คือไม่มี error

Length(st) : Integer Func :

ให้ค่าความยาวของสตริง st

MessageDlg(Msg, Atype,AButtons,HelpCtx) : Word Func : Dialogs

แสดง message dialog box ที่กลางหน้าจอ โดยจะมีชนิดของปุ่มให้ผู้ใช้ได้เลือก เช่น Yes, No, OK เป็นต้น เมื่อผู้ใช้เลือกแล้วก็จะส่งผลที่ผู้ใช้เลือกกลับไปยังโปรแกรมMsg เป็นข้อมูลสตริง string AType เป็นแบบ

ของ message box ที่จะแสดงบนหน้าจอ Abuttons เป็นแบบของปุ่มที่จะแสดงบนหน้าจอHelpCtx เป็นแบบของ help สำหรับสนับสนุน dilalog box นั้น

Proc :

New(p)

ให้กำหนดชั้นข้อมูลขึ้นในฮีพตามแบบข้อมูลที่กำหนดให้แก่ตัวแปรพอยน์เตอร์ p และให้ p ชี้ที่ชั้นข้อมูลนี้

Func :

Ord(n) : LongInt

ให้ลำดับที่ของข้อมูลแบบมีลำดับที่ n เช่น Ord('A') ให้ค่าเป็น 65

Func :

Pos(w,st) : Byte

ให้ตำแหน่งเริ่มต้นของสตริง w ในสตริง st

Proc :

ProcessMessages

ขัดจังหวะการทำงานของโปรแกรมเพื่อให้วินโดว์ตอบสนองต่อคำสั่งของผู้ใช้ได้ เช่น สั่งให้โปรแกรมตอบสนองต่อการกดแป้นพิมพ์หรือการเลื่อนเม้าส์ของผู้ใช้

Proc :

ReSet(f[,n])

เปิดไฟล์ทุกชนิด f เพื่ออ่านหรือเขียนคราวละ n เรคอร์ด ซึ่ง n จะใช้เมื่อ f เป็นไฟล์แบบไม่กำหนดแบบข้อมูลเท่านั้น เมื่อเปิดแล้วไฟล์พอยน์เตอร์จะอยู่ที่ต้นไฟล์

Proc :

Result

ใช้ในส่วนโปรแกรมย่อยที่เป็นฟังก์ชันเท่านั้น เพื่อให้ส่งผลของฟังก์ชันไปให้โปรแกรมหลัก

Proc :

ReWrite(f,n)

เปิดไฟล์ f เพื่อการเขียนคราวละ n เรคอร์ด ซึ่ง n จะใช้เมื่อ f เป็นไฟล์แบบไม่กำหนดแบบข้อมูลเท่านั้น ถ้าเป็นไฟล์เก่าข้อมูลเดิมจะถูกยกเลิก ถ้าเป็นไฟล์ใหม่จะสร้างไฟล์ว่างขึ้นในดิสก์ เมื่อเปิดแล้วไฟล์พอยน์เตอร์จะอยู่ที่ต้นไฟล์

Proc :

Seek(f,n)

เลื่อนไฟล์พอยน์เตอร์ของไฟล์ f จากตำแหน่งปัจจุบันไป n ไบต์

Func :

SizeOf(v) : Word

ให้ขนาดของตัวแปรหรือแบบข้อมูล v จำนวนเป็นไบต์

Func :

StrToInt(st) : LongInt

เปลี่ยนสตริงของตัวเลข st เป็นเลขจำนวน n

เช่น n := StrToInt('23'); หมายถึงให้ค่าแก่ตัวแปรเลขจำนวน n = 23

ภาคผนวก จ

แสดงความหมายของคำสั่งที่เกี่ยวกับออปเจค MSComm

1. คำสั่งสำหรับกำหนดสถานะของพอร์ตสื่อสารอนุกรม

MSComm.BaudRate	กำหนดค่าอัตราเร็วสูงสุดของการส่งข้อมูลผ่านพอร์ตสื่อสารอนุกรม
MSComm.DataBits	กำหนดจำนวนบิทข้อมูลสำหรับการสื่อสารอนุกรม
MSComm.FlowControl	กำหนดชนิดของการควบคุมการสื่อสารอนุกรม โดยอาจกำหนดเป็นแบบ RTS/CTS , XON/XOFF หรือไม่มีเลย
MSComm.ParityBits	กำหนดค่าพาริตี้บิทที่จะใช้ในการตรวจสอบความถูกต้องของข้อมูลในการสื่อสารข้อมูลแบบอนุกรม
MSComm.Port	กำหนดค่าเป็นหมายเลขพอร์ตสื่อสารอนุกรม
MSComm.RXFullCount	กำหนดขนาดของบัฟเฟอร์ของพอร์ตก่อนที่จะส่งสัญญาณไปบอกโปรแกรมว่ามีข้อมูลเข้ามา
MSComm.StopBits	กำหนดจำนวนบิทสุดท้ายสำหรับการสื่อสารอนุกรม

2.คำสั่งสำหรับเปิด/ปิดพอร์ต

MSComm.Open : Boolean	ให้โปรแกรมติดต่อกับพอร์ตสื่อสารอนุกรมผ่านออปเจค MSComm โปรแกรมจะส่งค่าเป็นจริง ถ้าสามารถติดต่อกับพอร์ตที่กำหนดได้สำเร็จ
MSComm.Close	ให้โปรแกรมเลิกการติดต่อกับพอร์ตสื่อสารอนุกรมที่ได้จองหรือเปิดไว้แล้ว

3.คำสั่งใช้อ่านข้อมูลจากพอร์ต และเขียนข้อมูลไปยังพอร์ต

MSComm.Read(Data:PChar; n:Word)	อ่านข้อมูลจากพอร์ตอนุกรมให้แก่ตัวแปร Data ซึ่งเป็นตัวแปรแบบพอยน์เตอร์ของตัวอักษร จำนวน n ไบต์ (ในความเป็นจริงแล้วจะอ่านจากบัฟเฟอร์ของพอร์ตที่ใช้เก็บข้อมูลเข้า)
MSComm.Write(Data:PChar; n:Word)	ให้เขียนข้อมูลจากตัวแปร Data จำนวน n ไบต์ไปยังพอร์ตอนุกรม เพื่อส่งไปในสายสัญญาณ (ในความเป็นจริงแล้วจะะทำการเขียนลงยังบัฟเฟอร์สำหรับเก็บข้อมูลออก)
MSComm.FlushRX	สั่งให้เคลียร์บัฟเฟอร์ที่ใช้เก็บข้อมูลที่ได้รับ
MSComm.FlushTX	สั่งให้ทำการส่งข้อมูลทั้งหมดที่ค้างอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลออกไปยังพอร์ต

3.การจัดการกับแมสเสจ (Message) เมื่อมีเกิดเหตุการณ์ (Event) ต่างๆ เกิดขึ้นกับ MSComm ดังนี้

MSComm.GetError	แปลงรหัสของ error ที่ส่งมาเป็นหมายเลขโดยวินโดวส์ ให้อยู่ในรูปข้อความ

MSComm.OnError ส่งแมสเสจแจ้งโปรแกรมเมื่อเกิดเหตุการณ์ (event) ที่มีข้อผิดพลาดขึ้นที่พอร์ตที่กำลังใช้งานอยู่ โปรแกรมจะได้รับแมสเสจเตือนเพื่อให้ทำการเรียกโพรซีเยอร์ที่กำหนดไว้

MSComm.OnReceive ส่งแมสเสจแจ้งเมื่อมีข้อมูลมารอที่บัฟเฟอร์เข้า เพื่อเตือนให้ผู้ใช้ทำการอ่านข้อมูลจากบัฟเฟอร์เข้า และจะทำการเรียกโพรซีเยอร์ที่ได้กำหนดไว้

MSComm.OnTransmitLow ส่งแมสเสจแจ้งเมื่อมีการเปลี่ยนโหมดการทำงานของแฟกซ์โมเด็มจากโหมดการส่งข้อมูลเป็นโหมดคำสั่ง

4.คำสั่งสำหรับกำหนดขนาดของบัฟเฟอร์

MSComm.TXBufSize กำหนดขนาดของบัฟเฟอร์ของ MSComm สำหรับเก็บข้อมูลที่จะส่ง

MSComm.RXBufSize กำหนดขนาดของบัฟเฟอร์ของ MSComm สำหรับข้อมูลที่ได้รับ

ภาคผนวก ง

แสดงความหมายของคำสั่งที่เกี่ยวกับออปเจค Sockets

Sockets.IP Addr	กำหนดหมายเลข IP ของฝ่ายที่จะทำการติดต่อด้วย
Sockets.NonBlocking	กำหนดโหมดของการทำงานของ socket ว่าเป็น non blocking mode หรือ blocking mode
Sockets.OnErrorOccurred	ส่งแมสเสจแจ้งโปรแกรมเมื่อเกิดข้อผิดพลาดของซ็อกเก็ต
Sockets.OnSessionClosed	ส่งแมสเสจแจ้งโปรแกรมเมื่อซ็อกเก็ตของอีกฝ่ายหนึ่งที่ติดต่อกันอยู่ขณะนั้นถูกปิด
Sockets.Peek	สั่งให้ทำการเรียกดูข้อมูลที่อยู่ในบัฟเฟอร์ โดยไม่ดึงข้อมูลที่เก็บไว้ในบัฟเฟอร์ออกมา ข้อมูลสูงสุด 255 ตัวอักษร
Sockets.Port	กำหนดหมายเลขของ socket port ที่จะใช้ในการติดต่อกับโปรแกรมประยุกต์ที่ต้องการ
Sockets.SAccept	จะใช้คู่กับคำสั่ง SListen โดยเมื่อมีผู้ขอติดต่อหลังจากคำสั่ง SListen แล้ว ฝ่ายที่สั่ง SListen จะสั่ง SAccept จะสร้างการติดต่อให้เกิดขึ้น
Sockets.SCancelListen	สั่งให้ยกเลิกการคอย
Sockets.SClose	สั่งให้ปิดซ็อคเก็ตที่ได้ทำการเปิดอยู่แล้ว
Sockets.SConnect	ใช้ร้องของการติดต่อกับฝ่ายที่คอยด้วยคำสั่ง SListen
Sockets.SListen	สั่งให้ทำการคอยการติดต่อจากอีกฝ่ายหนึ่ง ซึ่งจะเป็นเสมือนเซิร์ฟเวอร์ โดยจะทำการคอยเป็นเวลาที่ได้กำหนดไว้ใน Timeout แต่ถ้า Timeout = 0 จะกำการคอยต่อไปอย่างไม่มีกำหนด
Sockets.SocketNumber	แสดงหมายเลขซ็อคเก็ตที่ถูกกำหนดเมื่อมีการติดต่อเกิดขึ้น
Sockets.SReceive(SocketNumber, buffer, length)	สั่งให้ทำการอ่านข้อมูลจากซ็อคเก็ต SocketNumber มาเก็บไว้ยัง buffer เป็นจำนวน length ไบต์ คำสั่งนี้จะทำการคืนค่าจำนวนที่สามารถอ่านเข้ามาได้กลับให้อ่านข้อมูลจากซ็อกเก็ตให้แก่ตัวแปรหนึ่ง
Sockets.SSend(SocketNumber, data, numwrite)	สั่งให้การส่งข้อมูล data ไปยังซ็อคเก็ตหมายเลข SocketNumber เป็นจำนวน numwrite ไบต์
Sockets.Text	ให้ส่งข้อมูลจากตัวแปรสตริง ซึ่งจะส่งได้สูงสุด 255 ตัวอักษร
Sockets.Timeout	กำหนดเวลาที่ใช้ในการรอรับข้อมูลจากซ็อคเก็ต ถ้าหากซ็อคเก็ตไม่ส่งข้อมูลมาภายในเวลาที่กำหนดนี้ จะส่งสัญญาณเพื่อแสดงข้อผิดพลาด

ภาคผนวก จ

แสดงความหมายของคำสั่งที่เกี่ยวกับออปเจค Sockets

Sockets.IP Addr	กำหนดหมายเลข IP ของฝ่ายที่จะทำการติดต่อด้วย
Sockets.NonBlocking	กำหนดโหมดของการทำงานของ socket ว่าเป็น non blocking mode หรือ blocking mode
Sockets.OnErrorOccurred	ส่งแมสเสจแจ้งโปรแกรมเมื่อเกิดข้อผิดพลาดของซ็อกเก็ต
Sockets.OnSessionClosed	ส่งแมสเสจแจ้งโปรแกรมเมื่อซ็อกเก็ตของอีกฝ่ายหนึ่งที่ติดต่อกันอยู่ขณะนั้นถูกปิด
Sockets.Peek	สั่งให้ทำการเรียกดูข้อมูลที่อยู่ในบัฟเฟอร์ โดยไม่ดึงข้อมูลที่เก็บไว้ในบัฟเฟอร์ออกมา ข้อมูลสูงสุด 255 ตัวอักษร
Sockets.Port	กำหนดหมายเลขของ socket port ที่จะใช้ในการติดต่อกับโปรแกรมประยุกต์ที่ต้องการ
Sockets.SAccept	จะใช้คู่กับคำสั่ง SListen โดยเมื่อมีผู้ขอติดต่อหลังจากคำสั่ง SListen แล้ว ฝ่ายที่สั่ง SListen จะสั่ง SAccept จะสร้างการติดต่อให้เกิดขึ้น
Sockets.SCancelListen	สั่งให้ยกเลิกการคอย
Sockets.SClose	สั่งให้ปิดซ็อคเก็ตที่ได้ทำการเปิดอยู่แล้ว
Sockets.SConnect	ใช้ร้องของการติดต่อกับฝ่ายที่คอยด้วยคำสั่ง SListen
Sockets.SListen	สั่งให้ทำการคอยการติดต่อจากอีกฝ่ายหนึ่ง ซึ่งจะเป็นเสมือนเซิร์ฟเวอร์ โดยจะทำการคอยเป็นเวลาที่ได้กำหนดไว้ใน Timeout แต่ถ้า Timeout = 0 จะกำการคอยต่อไปอย่างไม่มีกำหนด
Sockets.SocketNumber	แสดงหมายเลขซ็อคเก็ตที่ถูกกำหนดเมื่อมีการติดต่อเกิดขึ้น
Sockets.SReceive(SocketNumber, buffer, length)	สั่งให้ทำการอ่านข้อมูลจากซ็อคเก็ต SocketNumber มาเก็บไว้ยัง buffer เป็นจำนวน length ไบต์ คำสั่งนี้จะทำการคืนค่าจำนวนที่สามารถอ่านเข้ามาได้กลับให้อ่านข้อมูลจากซ็อกเก็ตให้แก่ตัวแปรหนึ่ง
Sockets.SSend(SocketNumber, data, numwrite)	สั่งให้การส่งข้อมูล data ไปยังซ็อคเก็ตหมายเลข SocketNumber เป็นจำนวน numwrite ไบต์
Sockets.Text	ให้ส่งข้อมูลจากตัวแปรสตริง ซึ่งจะส่งได้สูงสุด 255 ตัวอักษร
Sockets.Timeout	กำหนดเวลาที่ใช้ในการรอรับข้อมูลจากซ็อคเก็ต ถ้าหากซ็อคเก็ตไม่ส่งข้อมูลมาภายในเวลาที่กำหนดนี้ จะส่งสัญญาณเพื่อแสดงข้อผิดพลาด

ภาคผนวก ฉ

แสดงความหมายของคำสั่งที่เกี่ยวกับออปเจค MSComm

1. คำสั่งสำหรับกำหนดสถานะของพอร์ตสื่อสารอนุกรม

MSComm.BaudRate	กำหนดค่าอัตราเร็วสูงสุดของการส่งข้อมูลผ่านพอร์ตสื่อสารอนุกรม
MSComm.DataBits	กำหนดจำนวนบิทข้อมูลสำหรับการสื่อสารอนุกรม
MSComm.FlowControl	กำหนดชนิดของการควบคุมการสื่อสารอนุกรม โดยอาจกำหนดเป็นแบบ RTS/CTS , XON/XOFF หรือไม่มีเลย
MSComm.ParityBits	กำหนดค่าพาริตี้บิทที่จะใช้ในการตรวจสอบความถูกต้องของข้อมูลในการสื่อสารข้อมูลแบบอนุกรม
MSComm.Port	กำหนดค่าเป็นหมายเลขพอร์ตสื่อสารอนุกรม
MSComm.RXFullCount	กำหนดขนาดของบัฟเฟอร์ของพอร์ตก่อนที่จะส่งสัญญาณไปบอกโปรแกรมว่ามีข้อมูลเข้ามา
MSComm.StopBits	กำหนดจำนวนบิทสุดท้ายสำหรับการสื่อสารอนุกรม

2.คำสั่งสำหรับเปิด/ปิดพอร์ต

MSComm.Open : Boolean	ให้โปรแกรมติดต่อกับพอร์ตสื่อสารอนุกรมผ่านออปเจค MSComm โปรแกรมจะส่งค่าเป็นจริง ถ้าสามารถติดต่อกับพอร์ตที่กำหนดได้สำเร็จ
MSComm.Close	ให้โปรแกรมเลิกการติดต่อกับพอร์ตสื่อสารอนุกรมที่ได้จองหรือเปิดไว้แล้ว

3.คำสั่งใช้อ่านข้อมูลจากพอร์ต และเขียนข้อมูลไปยังพอร์ต

MSComm.Read(Data:PChar; n:Word)	อ่านข้อมูลจากพอร์ตอนุกรมให้แก่ตัวแปร Data ซึ่งเป็นตัวแปรแบบพอยน์เตอร์ของตัวอักษร จำนวน n ไบต์ (ในความเป็นจริงแล้วจะอ่านจากบัฟเฟอร์ของพอร์ตที่ใช้เก็บข้อมูลเข้า)
MSComm.Write(Data:PChar; n:Word)	ให้เขียนข้อมูลจากตัวแปร Data จำนวน n ไบต์ไปยังพอร์ตอนุกรม เพื่อส่งไปในสายสัญญาณ (ในความเป็นจริงแล้วจะทำการเขียนลงยังบัฟเฟอร์สำหรับเก็บข้อมูลออก)
MSComm.FlushRX	สั่งให้เคลียร์บัฟเฟอร์ที่ใช้เก็บข้อมูลที่ได้รับ
MSComm.FlushTX	สั่งให้ทำการส่งข้อมูลทั้งหมดที่ค้างอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลออกไปยังพอร์ต

3.การจัดการกับแมสเสจ (Message) เมื่อมีเกิดเหตุการณ์ (Event) ต่างๆ เกิดขึ้นกับ MSComm ดังนี้

MSComm.GetError	แปลงรหัสของ error ที่ส่งมาเป็นหมายเลขโดยวินโดวส์ ให้อยู่ในรูปข้อความ

MSComm.OnError ส่งแมสเสจแจ้งโปรแกรมเมื่อเกิดเหตุการณ์ (event) ที่มีข้อผิดพลาดขึ้นที่พอร์ตที่กำลังใช้งานอยู่ โปรแกรมจะได้รับแมสเสจเตือนเพื่อให้ทำการเรียกโพรซีเยอร์ที่กำหนดไว้

MSComm.OnReceive ส่งแมสเสจแจ้งเมื่อมีข้อมูลมารอที่บัฟเฟอร์เข้า เพื่อเตือนให้ผู้ใช้ทำการอ่านข้อมูลจากบัฟเฟอร์เข้า และจะทำการเรียกโพรซีเยอร์ที่ได้กำหนดไว้

MSComm.OnTransmitLow ส่งแมสเสจแจ้งเมื่อมีการเปลี่ยนโหมดการทำงานของแฟกซ์โมเด็มจากโหมดการส่งข้อมูลเป็นโหมดคำสั่ง

4.คำสั่งสำหรับกำหนดขนาดของบัฟเฟอร์

MSComm.TXBufSize กำหนดขนาดของบัฟเฟอร์ของ MSComm สำหรับเก็บข้อมูลที่จะส่ง

MSComm.RXBufSize กำหนดขนาดของบัฟเฟอร์ของ MSComm สำหรับข้อมูลที่ได้รับ

เอกสารอ้างอิง

[1] ฉัตรชัย สุมามาลย์,นาวาตรี, การสื่อสารข้อมูลคอมพิวเตอร์และระบบเครือข่าย,บริษัทด่านสุธาการพิมพ์, ม.ป.ป.

[2] ศักดิศ สุวัชรังกูร และสันติ ลือวนิชวงศ์, การส่งโทรสารโดยผ่านแฟกซ์โมเด็ม, ปริญญานิพนธ์, 2537

[3] บุญเลิศ เอี่ยมทัศนา, เรียนรู้ภาษาปาสคาลด้วยเทอร์โบปาสคาล 4.0-5.0, บริษัท ซีเอ็ดยูเคชั่น จำกัด, 2532

[4] "สื่อสารข้อมูลโดยใช้ชุดโปรโตคอล TCP/IP ตอนที่ 1", ไมโครคอมพิวเตอร์ ฉ.113, หน้า 346-354

[5] Andrew Margolis, The Fax Modem Sourcebook, John Wiley&Sons Ltd., 1995

[6] Andrew Wozniewicz with Namir Shammas, Teach Yourself Borland Delphi in 21 Days, Sams Publishing, 1995

[7] Christian Schmitz-Moormann, 2864 Series Modem User's Manual, ZyXEL COMMUNICATIONS CORPORATION, 1995