ระบบประมวลผลสารสนเทศทางไกลในสภาพแวดล้อมกราฟิก

GRAPHICAL REMOTE INFORMATION PROCESSING SYSTEM

โดย

นายชินวัฒน์ ติวิธมไหศูรย์ 34102097

นายสุวิชัย แซ่ตั้ง 34108463

นายอัศวิน เชาวนกฤษณกุล 34109508

อาจารย์ที่ปรึกษา

อาจารย์วัชระ ฉัตรวิริยะ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบประมวลผลสารสนเทศทางไกลในสภาพแวดล้อมกราฟิก

ผู้จัดทำ นักศึกษาภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

1. นายชินวัฒน์ ติวิธมไหศูรย์ รหัสประจำตัว 34102097
2. นายสุวิชัย แซ่ตั้ง รหัสประจำตัว 34108463
3. นายอัศวิน เชาวนกฤษณกุล รหัสประจำตัว 34109508

.....อาจารย์ที่ปรึกษา

(อาจารย์วัชระ ฉัตรวิริยะ)

ระบบประมวลผลสารสนเทศทางไกลในสภาพแวดล้อมกราฟิก

ชินวัฒน์ ติวิธมไหศูรย์
สุวิชัย แซ่ตั้ง
อัศวิน เชาวนกฤษณกุล
อาจารย์วัชระ ฉัตรวิริยะ อาจารย์ที่ปรึกษา
ปีการศึกษา 2537

บทคัดย่อ

ในปัจจุบัน การติดต่อสื่อสารข้อมูลแบบออนไลน์มีการใช้งานมากขึ้น แต่ระบบที่ใช้งานอยู่นั้นทำงานในระบบปฏิบัติการดอส เป็นส่วนใหญ่ ซึ่งทำงานในรูปแบบตัวอักษร ทำให้มีข้อจำกัดอยู่หลายประการ สำหรับโครงงานระบบประมวลผลสารสนเทศทางไกลในสภาพแวดล้อมกราฟิกนี้ จัดทำขึ้นโดยใช้ระบบปฏิบัติการที่มีการติดต่อกับผู้ใช้แบบกราฟิก คือ โอเอสทู และ วินโดวส์ ซึ่งสามารถแก้ไขข้อจำกัดดังกล่าวได้ และมีเครื่องมือที่สนับสนุนการพัฒนาระบบมากกว่า ในโครงงานนี้ประกอบไปด้วย ส่วนหลักๆ 3 ส่วนด้วยกัน คือ ส่วนโฮส, ส่วนเทอร์มินัล และส่วนการติดต่อผ่านโมเด็ม ซึ่งส่วนเทอร์มินัล จะมีทั้งที่ทำงานบนโอเอสทู และบนวินโดวส์ เพื่อให้ผู้ใช้สามารถเลือกใช้งานได้ ส่วนการส่งข้อมูลระหว่างโฮส กับเทอร์มินัล จะส่งในรูปแบบคำสั่งกราฟิก ซึ่งจะทำให้ลดปริมาณข้อมูลที่จะส่งลง และทำให้การทำงานเร็วขึ้น ท้ายสุดเป็นตัวอย่างแอพพลิเคชัน ซึ่งสามารถนำไปประยุกต์ใช้ในงานอื่นๆ หรือ ใช้เป็นแนวทางในการพัฒนาต่อไปในอนาคตได้

Graphical Remote Information Processing System

Chinnawat Tivitmahaisoon

Suwichai Saetang

Atsawin Ch. Kritsanakul

Watchara Chatwiriya Advisor

1994

Abstract

Today, the online communication systems are growth in world-wide but most current systems work on DOS operating system which work in text mode. So, there are many limitations. Graphical Remote Information Processing System (GRIPS) are created on graphical operating systems, OS/2 and Windows, which can solve those limitations and there are many tools for development. This project comprises of three important parts, host, terminal and communication via modem. For terminal part, it works on OS/2 and Windows which user can choose. Data communication between host and terminal are graphical commands which can reduce quantity of data to send and transfer time. Finally, there is a sample application which can applied with other application or developed in better system in the future.

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

เชระบบปฏิบัติการหมการติดต่อกับผู้ใช้แบบ GUI (Graphical User Interface) เช่น โอเอสทู (OS/2) หรือ วินโดวส์ (Windows) เป็นต้น ระบบปฏิบัติการทั้งสองสามารถแก้ไขข้อจำกัดดังกล่าวได้หมด และมีเครื่องมือที่สนับสนุนการพัฒนาระบบอย่างครบครัน อำนวยความสะดวกในการพัฒนามากกว่า โดยเฉพาะส่วนที่ติดต่อกับผู้ใช้ ซึ่งมีการจัดเตรียมไว้ให้สำหรับผู้เขียนโปรแกรมเรียกใช้ได้โดยไม่ต้องพัฒนาเอง นอกจากนั้นโครงงานนี้ยังสามารถเป็นแนวทางในการพัฒนาต่อเนื่อง หรือนำไปประยุกต์ใช้ในงานอื่นต่อไปในอนาคตได้

1.2 ส่วนประกอบของโครงงาน

โครงงานนี้จะแบ่งส่วนประกอบของระบบออกเป็น 3 ส่วน คือ ส่วนแสดงข้อมูลและติดต่อกับผู้ใช้ ส่วนประมวลผลข้อมูล และส่วนที่ทำหน้าที่รับส่งข้อมูล โดยทั้งสองส่วนแรกจะเชื่อมต่อกันโดยผ่านการควบคุมไปยังส่วนที่สาม ซึ่งติดต่อกันผ่านเครือข่ายการสื่อสาร หรือผ่านการติดต่อแบบไฟล์

1.2.1 ส่วนแสดงผลข้อมูลและติดต่อกับผู้ใช้ หรือเทอร์มินัล (terminal)

ส่วนนี้เป็นส่วนทำงานในระบบปฏิบัติการแบบกราฟิก เช่น โอเอสทู หรือ วินโดวส์ เป็นต้น ซึ่งระบบปฏิบัติการเหล่านี้ มีมาตรฐานในการใช้คำสั่งต่างๆ ทางด้านกราฟิกเป็นมาตรฐานในลักษณะของการใช้ API (Application Programming Interface) ที่เรียกว่า GDI (Graphics Device Interface) ในวินโดวส์ และ GPI (Graphics Programming Interface) ในโอเอสทู เป็นส่วนหนึ่งของระบบปฏิบัติการ และมีเครื่องมือในการพัฒนาที่เพียบพร้อม ทำให้สามารถเขียนโปรแกรมโดยใช้ความสามารถของระบบปฏิบัติการได้อย่างเต็มที่ รวมทั้งเป็นการพัฒนาตามระบบติดต่อกับผู้ใช้ที่เป็นมาตรฐานอยู่แล้ว

โปรแกรมส่วนนี้จะมีหน้าที่แสดงข้อมูลที่ได้รับมาที่อยู่ในรูปของคำสั่งในโปรโตคอล (protocol) ที่กำหนดขึ้นเฉพาะ สำหรับการส่งข้อมูลกราฟิกในแบบ GDI และส่งการทำงานของผู้ใช้เช่นการเลือกระบบรายการ การกดปุ่มบนจอภาพ ในลักษณะของความหมายของการกระทำนั้นๆ ไปยังส่วนประมวลผลข้อมูล ซึ่งรูปแบบดังกล่าวเป็นการทำงานแบบเทอร์มินัลนั่นเอง ดังนั้นโปรแกรมในส่วนนี้จะเหมือนกันหมด สำหรับทุกการประยุกต์นำไปใช้งานในสารสนเทศที่มีรูปแบบต่างกัน

1.2.2 ส่วนประมวลผลข้อมูล หรือโฮส (host)

ส่วนนี้จะทำหน้าที่จัดเตรียมและประมวลผลข้อมูลที่จะ ทำการส่งที่อยู่ในรูปแบบตามโปรโตคอลที่กำหนดขึ้น และจัดส่งผ่านส่วนรับส่งข้อมูลไปยังส่วนแสดงผลข้อมูล และติดต่อกับผู้ใช้ รวมทั้งรับการติดต่อจากผู้ใช้ด้านเทอร์มินัลเพื่อประมวลผล และส่งผลลัพธ์กลับไปยังผู้ใช้ตามสคริปต์ (script) ที่จัดเตรียมไว้สำหรับโฮสนั้นๆ

1.2.3 ส่วนติดต่อและดำเนินการสื่อสาร

มีหน้าที่เป็นส่วนกลางในการติดต่อระหว่างสองโปรแกรมแรก ในลักษณะเป็นลำดับขั้น (layer) โดยโปรแกรมทั้งสองส่วนแรก จะส่งผ่านข้อมูลและการควบคุม มายังส่วนที่สามนี้ ซึ่งทำหน้าที่ดำเนินการติดต่อผ่านรูปแบบการติดต่อที่เลือก ซึ่งมีหลายลักษณะ ได้แก่

- *การสื่อสารผ่านอุปกรณ์โมเด็ม (modem)* เตรียมการติดต่อ ดำเนินการส่งข้อมูล การตรวจจับข้อผิดพลาดระหว่างการส่งข้อมูล และการยกเลิกการติดต่อ
- *การสื่อสารผ่านไฟล์* โดยการใช้ไฟล์เป็นตัวกลางในการติดต่อระหว่างโฮสและเทอร์มินัล ทำให้สามารถใช้ในระบบเครือข่ายเช่นแลน (LAN - Local Area Network) ได้

1.3 แนวคิดของโครงงาน

โครงงานพัฒนาระบบประมวลผลสารสนเทศทางไกลในสภาพแวดล้อมแบบกราฟิก GRIPS (Graphical Remoted Information Processing System) พัฒนาขึ้นโดยอาศัยแนวคิดในการใช้คำสั่งแบบ GDI เป็นสิ่งที่ใช้แสดงแทนสารสนเทศ (Information representation) ซึ่งมีความได้เปรียบในด้านความยืดหยุ่น ความรวดเร็วในการประมวลผล สามารถลดภาระของเทอร์มินัลในการควบคุมการแสดงผลและการติดต่อกับผู้ใช้ มีสิ่งที่เป็นแนวความคิดที่โดดเด่น คือ การรวมส่วนของคอนโทรล (control) ต่างๆ เข้าไปในโปรโตคอลระดับแอพพลิเคชั่น (Application Layer) ที่เรียกว่า GRIPP (Graphical Remote Information Processing Protocol) ที่เกี่ยวข้องกับการจัดสร้างและควบคุมการทำงานของคอนโทรลต่างๆ เช่น ปุ่มกด (push button) ช่องขีด (check box) พื้นที่ที่รับการกด (click area) เป็นต้น และโปรโตคอลในระดับบนนี้ แยกเป็นอิสระจากการติดต่อในระดับล่างตั้งแต่ระดับการเชื่อมต่อ (data link) ลงไป ทำให้สามารถมีรูปแบบการติดต่อระดับล่างได้หลายรูปแบบ ตามที่มีการออกแบบไว้จะสนับสนุนใน 2 รูปแบบ คือ การติดต่อผ่านอุปกรณ์โมเด็ม และการติดต่อผ่านไฟล์ในลักษณะโลคอล (local) โดยสรุปสามารถแยกแนวความคิดเด่นได้ 3 ประเด็นดังนี้

1.3.1 โปรโตคอลระดับบน

เป็นรวมการควบคุมการติดต่อกับผู้ใช้ (user interface) การควบคุมการติดต่อระหว่างโฮสและเทอร์มินัล เช่น การเข้าสู่การติดต่อ (login) เป็นต้น และการแสดงสารสนเทศในรูปคำสั่ง GDI เป็นโปรโตคอลที่แยกจากระดับล่างอย่างชัดเจน ลักษณะการใช้โปรโตคอลเช่นนี้ ทำให้สามารถพัฒนาส่วนของเทอร์มินัลได้ในสภาพแวดล้อมต่างกันได้ เช่น โอเอสทู และ วินโดวส์ เป็นต้น ลักษณะการทำงานเป็นการประยุกต์ใช้ความคิดของ RIP (Remote Information Protocol) ที่ใช้งานในระบบ BBS (Bulletin Board System) และระบบ HTML ที่ใช้ใน WWW (World Wide Web) มาใช้ในระบบที่มีสภาพแวดล้อมแบบกราฟิก

1.3.2 รูปแบบการติดต่อระดับล่าง

การติดต่อระดับล่างสามารถมีได้หลายรูปแบบ อย่างน้อยในระบบที่ออกแบบจะมี 2 แบบคือ

- การติดต่อผ่านอุปกรณ์โมเด็ม
- การติดต่อแบบโลคอลไฟล์

1.3.3 ลักษณะการประมวลผลสาระสนเทศ

การประมวลผลสารสนเทศจะทำที่ด้านโฮสเป็นส่วนใหญ่ คล้ายการทำงานแบบ ไคลเอนต์/เซอร์เวอร์ (Client/Server) ทำให้สามารถรักษาความปลอดภัยของข้อมูล ได้มากกว่าการส่งสารสนเทศมาประมวลผลที่เทอร์มินัล การส่งข้อมูลสารสนเทศจะอยู่ในรูปที่ถูกเปลี่ยนแปลงสำหรับการนำเสนอ เป็นคำสั่งสำหรับการแสดงผลแล้ว ผู้ใช้ด้านเทอร์มินัลไม่อาจจะนำไปใช้ได้เลยในทันที

1.4 ขอบเขตของโครงงาน

- โปรโตคอลที่ใช้ในการส่งข้อมูลแบบ GDI ที่ใช้ สนับสนุนการจัดการการติดต่อกับผู้ใช้รวมทั้งการจัดการภาพแบบเวคเตอร์ (vector) และบิตแมพ (bitmap) ตัวอักษร และข้อมูลสารสนเทศต่างๆ
- การประมวลการติดต่อกับผู้ใช้จะทำที่ด้านเทอร์มินัลเป็นส่วนใหญ่
- การประมวลผลสารสนเทศทำที่ด้านโฮสเป็นส่วนใหญ่
- สนับสนุนการทำงานของเทอร์มินัลในหลายแพลตฟอร์ม อย่างน้อย โอเอสทู และ วินโดวส์
- มีส่วนที่ตรวจสอบความผิดพลาดในการรับส่งข้อมูลผ่านการสื่อสารด้วยอุปกรณ์โมเด็ม

1.5 ประโยชน์ของโครงงาน

- ผู้ใช้สามารถใช้ระบบได้ง่ายขึ้นเนื่องจากทำงานใน GUI ที่เป็นมาตรฐาน
- สามารถนำแก่น (kernal) ของระบบไปใช้พัฒนาต่อเนื่องได้
- ผู้ใช้สามารถนำข้อมูลสารสนเทศไปใช้ร่วมกับระบบอื่นๆ ได้ง่าย
- ผู้ใช้มีโอกาสเลือกการใช้งานได้หลายแพลตฟอร์ม (multi-platform)

1.6 หัวข้อการทำงาน

- ศึกษาความเป็นไปได้ กำหนดปัญหาและขอบเขตของโครงงาน
- วางแผนการดำเนินงาน ขั้นตอนการดำเนินงาน
- ค้นคว้าเอกสารประกอบ หนังสืออ้างอิง และแหล่งข้อมูลอื่นประกอบโครงงาน เรื่องที่เกี่ยวข้อง ได้แก่
    1. วินโดวส์โปรแกรมมิง (Windows Programming), วินโดวส์ GDI-API (Windows GDI-API)
    2. โอเอสทูโปรแกรมมิง (OS/2 Programming), โอเอสทู GPI-API (OS/2 GPI-API)
    3. การสื่อสารข้อมูล (Data communication)
    4. โมเด็ม
- ดำเนินการเตรียม ติดตั้ง และศึกษาวิธีการใช้ระบบที่ใช้พัฒนา
- กำหนดและออกแบบโปรโตไทป์ของระบบเป้าหมาย
- ออกแบบโปรโตคอลที่ใช้ในการติดต่อ
- ออกแบบและจัดสร้างส่วนเทอร์มินัล ทั้งสภาพแวดล้อม โอเอสทู และ วินโดวส์
- ออกแบบและจัดสร้างส่วนโฮส

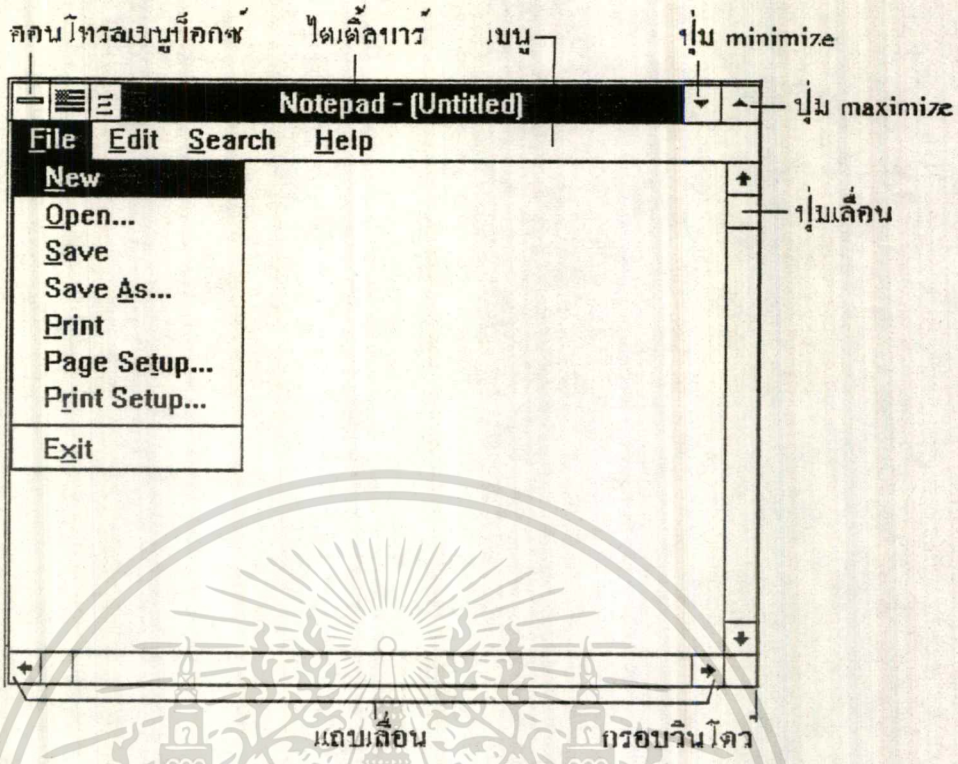
- ออกแบบและจัดสร้างส่วนติดต่อสื่อสาร
- จัดทำสคริปต์และจัดทำสารสนเทศสำหรับโปรโตไทป์
- การทดสอบระบบทั้งในส่วนประกอบ (component testing) ระบบโดยรวม (integration testing) และทดสอบระบบโดยผู้ใช้ (user testing)
- สรุปและประเมินผล

บทที่ 2
ทฤษฎีและหลักการ

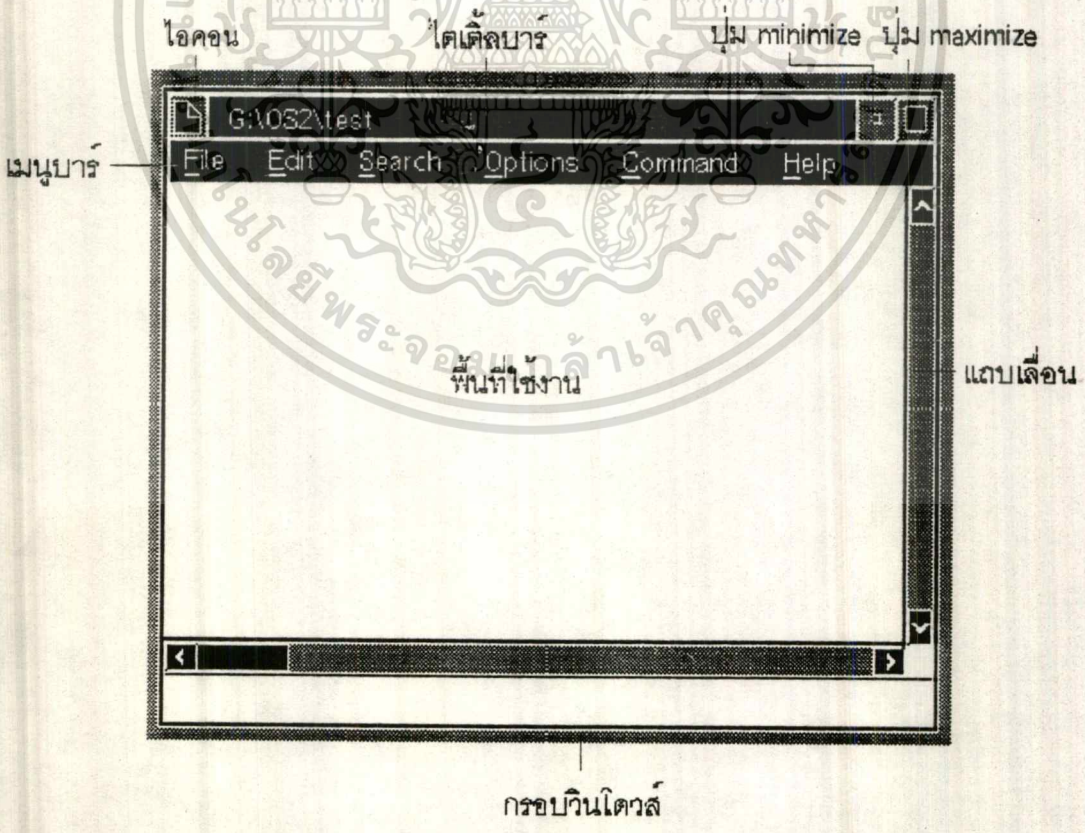
2.1 รูปแบบของแอพพลิเคชันที่ใช้การติดต่อกับผู้ใช้แบบกราฟิกในระบบโอเอสทูและวินโดวส์

แอพพลิเคชันที่ใช้ระบบติดต่อกับผู้ใช้แบบกราฟิก หรือ GUI (Graphical User Interface) นั้นล้วนมีความเหมือนกัน ในการใช้ความสามารถของระบบปฏิบัติการหรือ ระบบจัดการสภาพแวดล้อมแบบกราฟิกที่เป็นมาตรฐาน เป็นเครื่องมือในการติดต่อกับผู้ใช้ ทั้งระบบโอเอสทู และวินโดวส์ต่างเป็นระบบที่สร้างการติดต่อมาตรฐานระหว่างแอพพลิเคชันกับผู้ใช้ แอพพลิเคชันจึงประกอบไปด้วยส่วนประกอบต่างๆ ที่ดำเนินการติดต่อกับผู้ใช้ที่คล้ายคลึงกัน ได้แก่

- วินโดว์ (window)
- เมนู (menu)
- กรอบข้อความ (dialog box)
- เมสเสจ (message)



รูปที่ 2.1 ส่วนประกอบต่างๆ ของวินโดว์ในระบบวินโดวส์



รูปที่ 2.2 ส่วนประกอบต่างๆ ของวินโดว์ในระบบโอเอสทู

2.1.1 วินโดว์ (window)

เป็นสิ่งที่ติดต่อกับผู้ใช้อย่างพื้นฐานที่ทุกแอพพลิเคชันมักจะมีใช้เสมอ วินโดว์นั้นจะถูกกำหนดลักษณะที่แสดงขึ้นมาตามรูปแบบที่ผู้ใช้กำหนดตามสไตล์ (style) สำหรับแต่ประเภทของวินโดว์นั้นๆ ตั้งแต่ตอนสร้างวินโดว์ ในการสร้างวินโดว์นั้นแอลพลิเคชันจะเป็นผู้สร้าง แต่การจัดการวินโดว์นั้นจะเป็นหน้าที่ของระบบปฏิบัติการ ไม่ว่าจะเป็นการวาดวินโดว์บนจอภาพด้วยรูปแบบต่างๆ ที่แสดงสถานะขณะนั้น, ตำแหน่งของวินโดว์, การเปลี่ยนแปลงขนาดของวินโดว์ ฯลฯ พื้นที่ส่วนหนึ่งของวินโดว์ที่แอพพลิเคชันสามารถควบคุมรูปแบบการใช้งานต่างๆ อย่างเต็มที่เรียกว่า พื้นที่ทำงาน (client area) ซึ่งแต่ละแอพพลิเคชันนั้นสามารถมีวินโดว์ได้หลายวินโดว์ ดังนั้นพื้นที่ทำงานของแต่ละวินโดว์จึงแตกต่างกันไป จึงจำเป็นที่จะต้องมีการกำหนดให้เขียนส่วนตอบสนองของแต่ละวินโดว์ หรือพื้นที่ทำงานแต่ละอันโดยเฉพาะในรูปแบบของฟังก์ชันประจำวินโดว์ (window procedure) ซึ่งมีรูปแบบพิเศษต่างจากฟังก์ชันปกติทั่วไป

2.1.2 เมนู

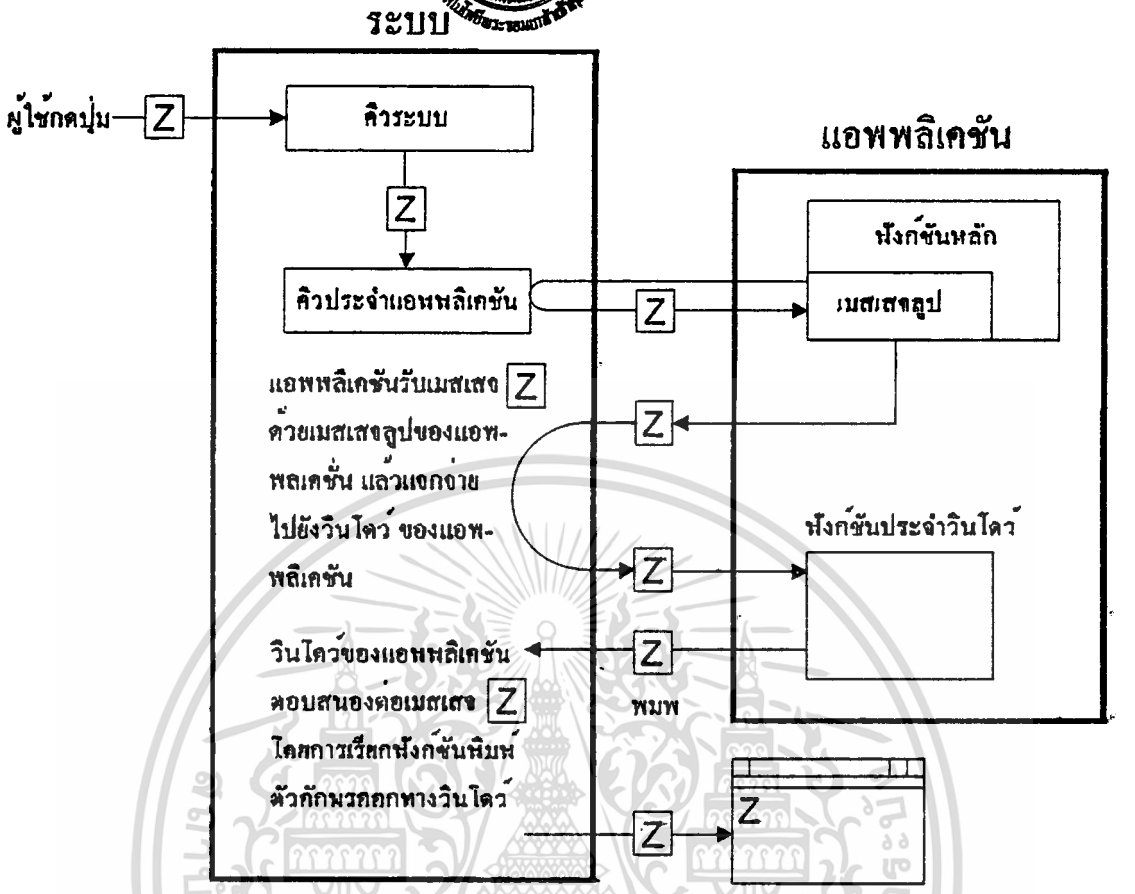
เป็นสิ่งที่ใช้รับข้อมูลหลักของแอพพลิเคชันที่ทำงานในระบบ เมนูประกอบไปด้วยรายการคำสั่งเรียงกันไปในลักษณะที่เป็นรายการหลัก และรายการย่อย ให้ผู้ใช้สามารถเลือกและใช้งานได้ง่าย โดยหน้าที่ในการจัดการ การแสดงเมนู และการเลือกรายการในเมนูเป็นหน้าที่ของระบบปฏิบัติการ หลักจากผู้ใช้เลือกเรียบร้อยแล้ว ระบบจะส่งคำสั่งที่ผู้ใช้เลือกผ่านมาทางเมสเสจคิว (message queue) เพื่อให้แอพพลิเคชั่นได้ตอบสนองต่อคำสั่งนั้นๆ

2.1.3 กรอบข้อความ

กรอบข้อความเป็นวินโดว์ชนิดหนึ่งที่ใช้งานชั่วคราว เพื่อให้แอพพลิเคชันได้ติดต่อกับผู้ใช้เช่น รับคำสั่งในรายละเอียดมากขึ้น ภายในกรอบข้อความจะประกอบไปด้วยคอนโทรล (controls เป็นวินโดว์ชนิดหนึ่ง) คอนโทรลแต่ละชนิดจะใช้สำหรับการติดต่อกับผู้ใช้แบบง่าย หนึ่งแบบ เช่น Edit เป็นวินโดว์ที่ให้ผู้ใช้ใส่ข้อความ ปุ่มกด (push button) เป็นวินโดว์สำหรับรับการกดเมาส์ (mouse) นอกจากกรอบข้อความที่สร้างขึ้นโดยแอพพลิเคชันแล้ว ยังมีกรอบข้อความที่ระบบปฏิบัติการได้จัดเตรียมไว้แล้วสำหรับการทำงานทั่วๆ ไป (common dialog) ซึ่งแอพพลิเคชันสามารถเรียกใช้โดยไม่ต้องมีส่วนที่จัดการกรอบข้อความ เพียงแต่ส่งพารามิเตอร์ (parameter) บางตัวไปเท่านั้น

2.1.4 เมสเสจ

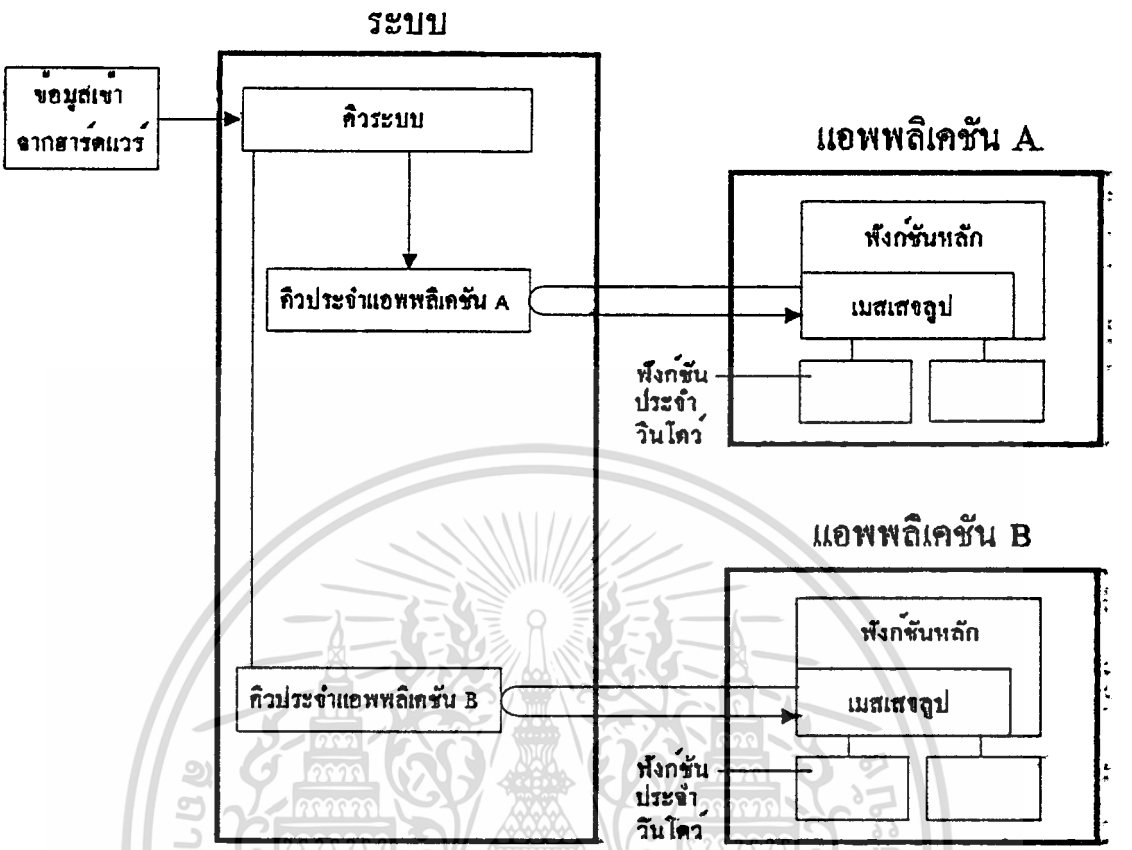
เนื่องจากแอพพลิเคชันต่างๆ นั้นรับข้อมูลเข้าโดยผ่านทางเมสเสจคิว ส่วนประกอบหลักของแอพพลิเคชันจึงต้องมีส่วนหนึ่งที่เรียกว่า เมสเสจลูป (message loop) เพื่อที่จะรับเมสเสจ และแจกจ่ายไปยังวินโดว์ที่เหมาะสม



รูปที่ 2.3 การจัดการเข้าทางแป้นพิมพ์

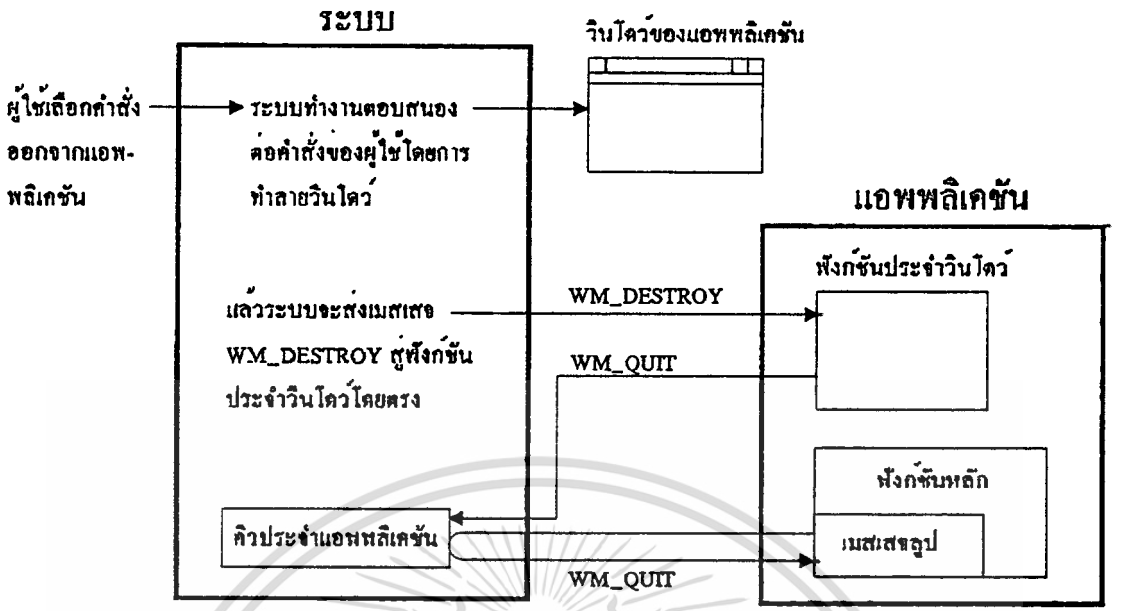
จากรูปที่ 2.3 เมื่อเรากดแป้นพิมพ์หนึ่งปุ่ม ระบบจะรับทราบและสร้างเมสเสจสำหรับแป้นพิมพ์ที่เหมาะสม จัดเก็บเข้าไปในคิวของระบบ (system queue) ซึ่งเป็นเมสเสจคิวของระบบทั้งระบบ แล้วส่งต่อไปยังคิวประจำแอพพลิเคชัน (application queue) ของแอพพลิเคชันที่เหมาะสม หลังจากนั้นเมสเสจลูปจะอ่านเมสเสจ และตีความ ซึ่งจะได้เป็นเมสเสจของตัวอักษรในรูปของรหัสอักขระ แล้วแจกจ่ายในลักษณะของเมสเสจจากแป้นพิมพ์ ไปยังฟังก์ชันประจำวินโดว์ที่เหมาะสม

ระบบไม่เพียงแค่สามารถจัดการเมสเสจให้แก่แอพพลิเคชันเดียวเท่านั้น แต่ยังสามารถรวบรวมและแจกจ่ายเมสเสจให้แก่แอพพลิเคชันหลายๆ แอพพลิเคชันพร้อมกันได้อีกด้วย ดังรูปที่ 2.4 จะเห็นได้ว่าระบบรวบรวมข้อมูลเข้าทั้งหมด แล้วแจกแจงส่งไปให้แต่ละแอพพลิเคชัน ในแต่ละแอพพลิเคชันจะมีเมสเสจลูปคอยรับและแจกจ่ายไปให้ฟังก์ชันประจำวินโดว์ของตน



รูปที่ 2.4 การจัดการข้อมูลสำหรับแอพพลิเคชันสองตัว

การส่งเมสเสจ นอกจากจะส่งทางคิวประจำแอพพลิเคชันแล้ว ยังสามารถส่งโดยตรงได้ด้วย เมสเสจบางชนิด เช่นเมื่อระบบปฏิบัติการตอบรับการทำลายวินโดว์ ก็จะส่งเมสเสจชื่อ WM_DESTROY ไปยังฟังก์ชันประจำวินโดว์โดยตรงไม่ผ่านเมสเสจคิวของระบบ หลังจากนั้นฟังก์ชันประจำวินโดว์ก็จะตอบสนอง โดยส่งสัญญาณไปยังฟังก์ชันหลักของแอพพลิเคชัน (main ในโอเอสทู หรือ WinMain ในวินโดวส์) ว่าวินโดว์ได้ถูกทำลายลงแล้ว แอพพลิเคชันควรจะเลิกทำงาน โดยการใช้ฟังก์ชันส่งเมสเสจที่ชื่อ WM_QUIT ไปยังคิวประจำแอพพลิเคชัน ดังรูปที่ 2.5 เมื่อเมสเสจลูปได้รับเมสเสจ WM_QUIT ก็จะเลิกการทำงาน



รูปที่ 2.5 การจัดการเมสเสจของวินโดว์

2.2 สถาปัตยกรรมของโอเอสทู

โอเอสทู เป็นระบบปฏิบัติการอีกระบบหนึ่งสำหรับพีซี ถึงแม้ในขณะนี้ยังมีผู้ใช้อยู่น้อย แต่ก็เป็นที่ยอมรับเพิ่มมากขึ้นในบางส่วนของกลุ่มผู้ใช้พีซี ปัจจุบันโอเอสทูได้พัฒนาไปถึงรุ่น 3.0 แล้ว

2.2.1 ประวัติของโอเอสทู

ระบบปฏิบัติการโอเอสทู เกิดขึ้นเมื่อปี 1984 บริษัทอินเทลวางตลาดซีพียู 80286 พร้อมกับบริษัทไอบีเอ็มวางตลาดคอมพิวเตอร์ไอบีเอ็มพีซีเอที (IBM PC/AT) ที่ใช้ไมโครโปรเซสเซอร์รุ่นนี้ ซีพียู 80286 มีโหมดปฏิบัติการ 2 โหมด คือ เรียลโหมด (real mode) และโปรเท็กโหมด (protected mode) ในเรียลโหมดแล้ว 80286 จะมีความสามารถเหมือนกับซีพียู 8086 เพียงแต่มีความเร็วมีมากกว่า แต่ในโปรเท็กโหมด 80286 มีความสามารถเพิ่มขึ้นหลายอย่าง

ความสามารถใหม่ที่สำคัญที่สุดเกี่ยวข้องกับหน่วยความจำ ประการแรก 80286 สามารถติดต่อกับหน่วยความจำได้ถึง 16 MB มากกว่า 8086 ถึง 16 เท่าตัว ประการที่สอง 80286 มีความสามารถป้องกันส่วนต่างๆ ของหน่วยความจำจากการติดต่อที่ได้รับอนุญาต (เป็นที่มาของชื่อ โปรเท็กโหมด) คุณสมบัติเหล่านี้ทำให้ 80286 เป็นซีพียูที่เหมาะสมกับงานแบบมัลติทาสก์ พื้นที่หน่วยความจำขนาดใหญ่ของ 80286 ทำให้โหลดโปรแกรมหลายๆ อย่างเข้าในหน่วยความจำได้ในเวลาเดียวกัน ความสามารถในการป้องกันหน่วยความจำ ทำให้ป้องกันพื้นที่หน่วยความจำของแต่ละโปรแกรม ไม่ให้รบกวนกัน

เมื่อต้องการนำเอาความก้าวหน้าของโปรเท็กโหมดมาใช้อย่างเต็มที่ จึงต้องมีระบบปฏิบัติการใหม่ ในปี 1984 บริษัทไอบีเอ็ม และบริษัทไมโครซอฟต์ ร่วมกันพัฒนาโอเอสทู เพื่อให้เป็นระบบปฏิบัติการสำหรับซีพียู 80286

ขึ้นไป หลังจากนั้น โอเอสทูรุ่น 1.0 ก็วางตลาดในปี 1987 ไม่มีการติดต่อแบบกราฟิก แต่ก็นับเป็นการเริ่มต้น และทำให้บริษัทต่างๆ ได้พัฒนาโปรแกรมสำหรับโอเอสทู นอกจากนั้น โอเอสทูยังมีดอสคอมแพทิบิลิตีบ็อกซ์ (DOS compatibility box) เพื่อรันโปรแกรมของเอ็มเอสดอส

ในปลายปี 1988 โอเอสทูรุ่น 1.1 ได้วางตลาด มีการปรับปรุงหลายอย่าง รวมทั้งการติดต่อกับผู้ใช้แบบกราฟิก และวินโดว์ หลังจากนั้นก็ได้วางตลาดรุ่น 1.2 ในปี 1989 และรุ่น 1.3 ในปี 1991 โดยได้พัฒนาขึ้นมาตามลำดับ

อย่างไรก็ดี ฮาร์ดแวร์ที่นำมาใช้กับโอเอสทู อย่างได้ผลนั้น มีราคาค่อนข้างแพง และถึงแม้จะมีโปรแกรมสำหรับโอเอสทูมากขึ้น แต่ก็ยังน้อยเมื่อเปรียบเทียบกับโปรแกรมสำหรับดอส ปัญหาเกี่ยวกับสมรรถภาพ และความไม่สามารถเข้ากันได้กับฮาร์ดแวร์ที่ไม่ใช่ของไอบีเอ็ม มีผลทำให้ผู้ใช้ส่วนมากมีท่าทีรีรออยู่ในการใช้ โอเอสทู

นอกจากนั้น เมื่อไมโครซอฟต์วางตลาด วินโดวส์ รุ่น 3.0 ในปี 1990 ซึ่งสามารถทำงานได้ดีในระบบที่มีราคาถูกกว่าที่โอเอสทูต้องการ ผู้ใช้ก็หันมาใช้วินโดวส์กันเป็นจำนวนมาก บริษัทไมโครซอฟท์ก็ตัดสินใจยุติการร่วมพัฒนาโอเอสทูกับไอบีเอ็ม ทำให้บริษัทไอบีเอ็มต้องพัฒนาโอเอสทูไปตามลำพัง

เดือนพฤษภาคมปี 1992 บริษัทไอบีเอ็มได้วางตลาดโอเอสทูรุ่น 2.0 ซึ่งได้พัฒนาขึ้นมาใหม่หมด โดยให้ทำงานบนเครื่องที่ใช้ซีพียู 80386 ซึ่งเป็นที่ยอมรับกันพอสมควร เพราะมีประสิทธิภาพดีขึ้น และต่อมาไม่นานก็ได้ออก รุ่น 2.1 มา โดยมีการเปลี่ยนแปลงไปจากรุ่น 2.0 เล็กน้อย โอเอสทูก็เป็นที่นิยมกันมากขึ้น มีโปรแกรมทำงานบน โอเอสทูวางตลาดมากขึ้นเรื่อยๆ

ปลายปี 1994 บริษัทไอบีเอ็มวางตลาดโอเอสทูรุ่น 3.0 ซึ่งทำงานได้เร็วขึ้นกว่าเดิมเป็นอันมาก ประกอบกับวินโดวส์ 95 ของไมโครซอฟท์ เลื่อนกำหนดการวางตลาด ทำให้ผู้ใช้หันมาใช้โอเอสทูกันมากมาย

2.2.2 คุณสมบัติของโอเอสทู

โอเอสทูมีคุณสมบัติเด่นๆ อยู่หลายอย่างด้วยกัน ในที่นี้จะกล่าวพอสังเขป

2.2.2.1 การติดต่อกับผู้ใช้แบบกราฟิก

โอเอสทูมีการติดต่อกับผู้ใช้แบบกราฟิก ซึ่งจะแสดงข้อมูลข่าวสารในวินโดว์ของจอภาพ ใช้ไอคอน (icon) เพื่อแทนโปรแกรม, ข้อมูล และงานต่างๆ โดยใช้เมนูเพื่อเสนอทางเลือกต่างๆ ต่อผู้ใช้ GUI นั้นออกแบบมาเพื่อใช้กับเมาส์ ซึ่งทำให้การใช้คอมพิวเตอร์ง่ายขึ้น มากกว่าการพิมพ์คำสั่งลงไป

โอเอสทูรุ่นแรกๆ มี GUI ที่เรียกว่าพรีเซนเตชันแมนเนเจอร์ (presentation manager) ซึ่งคล้ายกับ GUI ของไมโครซอฟต์วินโดวส์ GUI ส่วนในรุ่น 2.0 ขึ้นมา จะเป็นเวิร์กเพลซเชลล์ (workplace shell)

เวิร์กเพลซเชลล์ เป็น GUI เชิงวัตถุ (object-oriented GUI) การกระทำต่างๆ ของผู้ใช้ทั้งหมดจะเกิดขึ้นบนจอภาพที่เรียกว่าเดสก์ทอป (desktop) บนเดสก์ทอปจะใช้ไอคอนแทนออบเจ็กต์ต่างๆ ของระบบ คือ โปรแกรม, ข้อมูล และฮาร์ดแวร์จะจัดเป็นออบเจ็กต์ การทำงานของผู้ใช้ทั้งหมด จะทำได้โดยการจัดการออบเจ็กต์ต่างๆ ซึ่งปกติจะใช้เมาส์

2.2.2.2 ระบบไฟล์แบบ HPFS

โอเอสทูรุ่น 1.2 เป็นต้นมา สามารถใช้ระบบไฟล์ ไฮเพอร์ฟอร์แมนซ์ไฟล์ซิสเต็ม (high performance file system, HPFS) ระบบไฟล์ HPFS เป็นการทดแทนระบบไฟล์แบบ แฟต (file allocation table, FAT) ที่เอ็มเอสดอส และไมโครซอฟต์วินโดวส์ 3.x ใช้อยู่

ระบบ HPFS มีจุดเด่นอยู่หลายประการ คือ

- สามารถใช้ชื่อไฟล์ยาวถึง 254 ตัวอักษร ทำให้ผู้ใช้ตั้งชื่อไฟล์ได้สะดวกขึ้น
- เก็บข้อมูลเกี่ยวกับตำแหน่งของไฟล์ใกล้กับตัวไฟล์เอง การจัดแบบนี้ทำให้หัวเขียน/อ่านเคลื่อนไหวน้อยที่สุด และทำให้ความเร็วของดิสก์สูงสุด
- เรียงลำดับชื่อไฟล์ในไดเรกทอรีตามลำดับตัวอักษรโดยอัตโนมัติ ทำให้โปรแกรมหาไฟล์ที่ต้องติดต่อได้ง่ายขึ้น
- มีแอตทริบิวต์ส่วนขยาย (extended attribute) ทำให้ข้อความบรรยายต่างๆ เช่น ชื่อผู้เขียนเรื่องสรุปเข้าร่วมกับแต่ละไฟล์ได้โดยไม่ต้องเป็นส่วนของไฟล์เอง
- มีการรวบรวมข้อมูลดิสก ที่ทำให้เข้าถึงแบบสุ่ม (random access) มีประสิทธิภาพกว่าที่ต้องอ่านไฟล์ทั้งหมดตามลำดับ (sequential) การติดต่อดิสก์แบบนี้ปกติเป็นที่ต้องการของโปรแกรมฐานข้อมูล

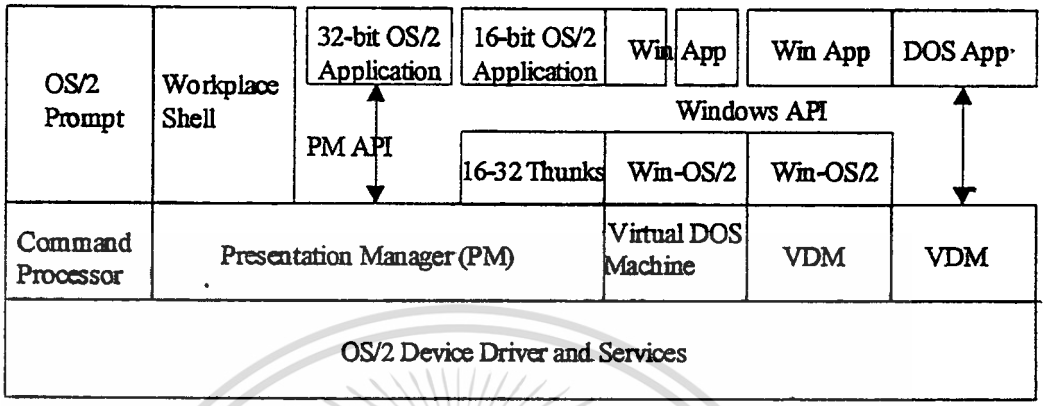
โอเอสทูสามารถใช้ได้ทั้งระบบ HPFS หรือ FAT เมื่อติดตั้งโอเอสทู ก็มีทางเลือกว่า จะใช้ระบบไฟล์แบบใด โดยถ้าใช้แบบ FAT ก็สามารถใช้ร่วมกับเอ็มเอสดอสได้

2.2.2.3 ความเข้ากันได้กับเอ็มเอสดอส และวินโดวส์

โปรแกรมสำหรับเอ็มเอสดอสส่วนมากสามารถทำงานใน ดอสเซสชัน (DOS session) ของโอเอสทูได้โดยสามารถมีดอสเซสชันได้หลายๆ ตัว และแต่ละตัวเป็นอิสระจากดอสเซสชันอื่นๆ ในแต่ละดอสเซสชัน โปรแกรมจะเห็นถึงซีพียู 8086 และหน่วยความจำร่วมของแต่ละโปรแกรมเอง ซึ่งสามารถทำได้ด้วยความสามารถของซีพียู 80386 ที่เลียนแบบซีพียู 8086 หลายๆ ตัว แต่ละดอสเซสชันจะเรียกว่า เวอร์ชวลดอสแมชีน (virtual DOS machine, VDM)

นอกจากนี้ ในตัวโอเอสทูรุ่น 2.0 เป็นต้นมา จะมีไมโครซอฟท์วินโดวส์อยู่ด้วย โดยในรุ่น 2.0 และ 2.1 จะมีวินโดวส์รุ่น 3.0 และ 3.1 ตามลำดับ ส่วนโอเอสทูรุ่น 3.0 จะสามารถใช้ร่วมกับวินโดวส์รุ่น 3.1 หรือ 3.11 ได้

2.2.3 โครงสร้างของโอเอสทู



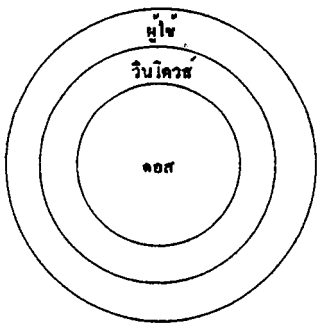
รูปที่ 2.6 โครงสร้างของโอเอสทู

จากรูปเป็นโครงสร้างของโอเอสทูในลำดับชั้นต่างๆ โดยระดับล่างสุดจะเป็นระดับของเคอร์แนล (kernel) ซึ่งเป็นระดับของระบบ มีหน้าที่ควบคุมการทำงานของระบบ ให้เป็นไปอย่างเรียบร้อย เช่นการจัดการไฟล์ เป็นต้น ระดับต่อมาจะเป็นระดับบริการ (service) โดยจะมีหลายชนิดเพื่อตอบสนองแอพพลิเคชันหลายแบบ ส่วนระดับบนสุดก็จะเป็นระดับ แอพพลิเคชัน ซึ่งมีได้หลายแบบคือ โอเอสทู, โอเอสทูรุ่น 1.X, ดอส และวินโดวส์ ดังได้กล่าวมาแล้ว

2.3 สถาปัตยกรรมของวินโดวส์

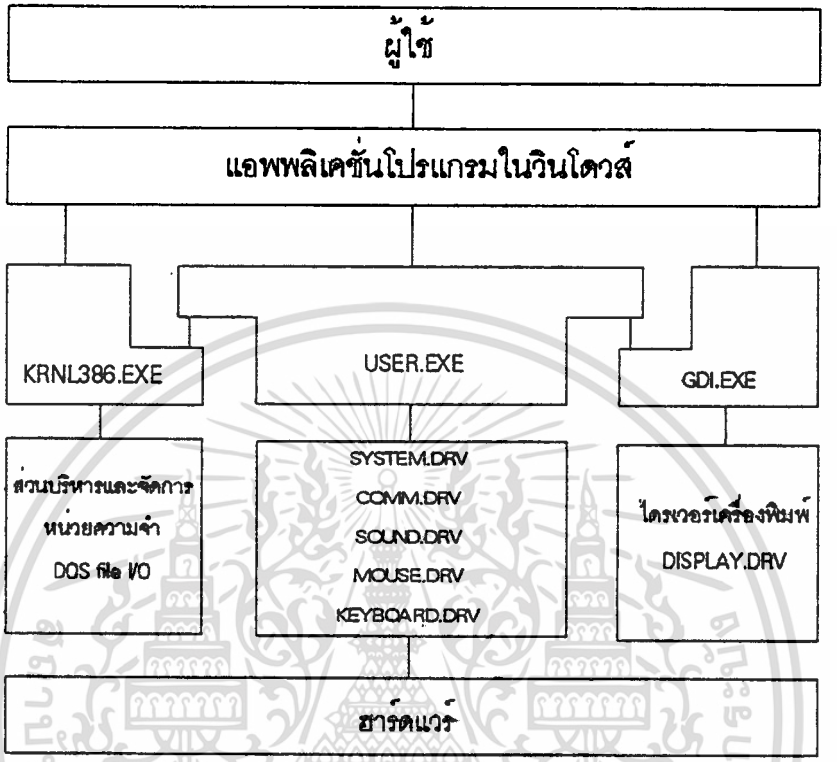
2.3.1 ลักษณะทั่วไป

วินโดวส์ตั้งแต่เวอร์ชัน 3.X ลงไป เป็นโปรแกรมที่ทำงานอยู่บนระบบปฏิบัติการดอส โดยทำหน้าที่เป็นส่วนเชื่อมการติดต่อระหว่างผู้ใช้กับดอส เป็นระบบจัดการสภาพแวดล้อมแบบกราฟิก ซึ่งอยู่ในตำแหน่งที่ห่อหุ้มดอส และชั้นอยู่ระหว่างดอสกับผู้ใช้ดังรูปที่ 2.7



รูปที่ 2.7 ระดับชั้นของดอส วินโดวส์ และผู้ใช้

2.3.2 โครงสร้าง



รูปที่ 2.8 ระดับชั้นของวินโดวส์

จากรูปเป็นโครงสร้างของวินโดวส์ในลำดับชั้นต่าง ๆ ระหว่างผู้ใช้กับฮาร์ดแวร์ เริ่มต้นจากผู้ใช้จะติดต่อกับเครื่องโดยผ่านทางแอพพลิเคชัน ซึ่งเรียกใช้รูทีน (routine) บริการต่าง ๆ จากวินโดวส์อีกทีหนึ่ง โดยแอพพลิเคชันจะเข้าถึงไลบรารีหลักทั้ง 3 คือ Kernel, User และ GDI ซึ่งต่างไม่ขึ้นกับฮาร์ดแวร์ ซึ่งแต่ละไลบรารีมีรายละเอียดคือ

2.3.2.1 User

เป็นไลบรารีที่บรรจุฟังก์ชันที่ใช้ในการจัดการวินโดว์และจัดการสภาพแวดล้อมทั่วไป

2.3.2.2 Kernel

เก็บฟังก์ชันบริการต่างๆ เช่น การจัดระบบจัดการงานแบบหลายงาน การบริหารจัดการหน่วยความจำ และการจัดการทรัพยากร

2.3.2.3 GDI

เก็บฟังก์ชันการติดต่อกับอุปกรณ์แสดงผลกราฟิก

2.3.3 การจัดการวินโดว์

การจัดการวินโดว์ คือการจัดการเกี่ยวกับเมสเสจของวินโดว์นั้นๆ โดยจะมีการทำงานเกี่ยวกับเมสเสจ เช่น การสร้างเมสเสจ, การแปล และตีความเมสเสจ เป็นต้น

2.3.3.1 เมสเสจ

เมสเสจเป็นอินพุทหรือข้อมูลเข้าเพียงทางเดียวของแอพพลิเคชันวินโดว์ เป็นตัวแทนต่อทุกๆ เหตุการณ์ ทั้งหมดที่ต้องการตอบสนอง เมสเสจเป็นนตัวแปรโครงสร้างชนิดหนึ่งที่ประกอบไปด้วยส่วนค่าอ้างอิงของเมสเสจนั้นกับส่วนพารามิเตอร์ของเมสเสจ โดยที่พารามิเตอร์จะขึ้นกับเมสเสจชนิดนั้น

2.3.3.2 การสร้างและประมวลผลเมสเสจ

วินโดว์จะมีการสร้างเมสเสจขึ้นทุกครั้งที่มีอินพุทเข้ามายังระบบ ไม่ว่าจะเป็นการเลื่อนเมาส์ การกดคีย์หรืออื่นๆ เพื่อบอกแก่แอพพลิเคชันหรือตัววินโดว์เองเหตุการณ์ที่เกิดขึ้น โดยวินโดว์จะนำเมสเสจที่สร้างเหล่านี้ใส่เข้าไปในคิวของระบบ จากนั้นก็ส่งผ่าน ไปยังคิวของแอพพลิเคชันที่เหมาะสมต่อไป ลักษณะของคิวของแอพพลิเคชันนี้เป็นแบบเข้าก่อนออกก่อน โดยที่การเข้าก็คือ วินโดว์ดึงเมสเสจจากคิวของระบบมาใส่ให้แก่คิวของแอพพลิเคชัน ส่วนการนำออกคือ แอพพลิเคชันเป็นตัวดึงออกไปเองโดยการเรียกใช้ฟังก์ชัน GetMessage จากนั้นก็แจกแจงและส่งไปยังฟังก์ชันประจำวินโดว์ต่างๆ ด้วยฟังก์ชัน DispatchMessage

มีบางเมสเสจที่วินโดว์ต้องการส่งไปยังฟังก์ชันประจำตัววินโดว์โดยตรง แทนที่จะวางลงในคิว อาจเรียกเมสเสจพวกนี้ว่าเป็น เมสเสจลัดคิว เมสเสจลัดคิวนี้จะเป็นเมสเสจที่มีผลกับวินโดว์นั้นโดยเฉพาะวินโดว์ และแอพพลิ เคชันจะส่งเมสเสจชนิดนี้ด้วยฟังก์ชัน SendMessage ซึ่งจะเป็นการส่งเมสเสจไปยังฟังก์ชันประจำตัววินโดว์โดยตรง อีกทั้งฟังก์ชันประจำตัววินโดว์นั้นยังสามารถส่งค่าการประมวลผลกลับมาได้อีกด้วย

ตัวอย่างของเมสเสจลัดคิวอย่างหนึ่งก็คือ การใช้ฟังก์ชัน CreateWindow นอกจากจะเป็นการสร้างวินโดว์แล้ว ยังเป็นการบังคับให้วินโดว์ส่งเมสเสจ WM_CREATE ซึ่งเป็นเมสเสจลัดคิวไปยังฟังก์ชันประจำตัววินโดว์ แล้วรอจนกว่าฟังก์ชันนั้นจะจัดการกับเมสเสจเสร็จ เมสเสจ WM_CREATE จะไม่ผ่านคิวแอพพลิเคชันเลย

แต่ก็ไม่ใช่ว่ามีเพียงวินโดว์เท่านั้นที่สามารถสร้างเมสเสจขึ้นว่าได้ ตัวแอพพลิเคชันเองก็สามารถสร้างเมสเสจแล้วส่งไปยังคิวของตนเองและคิวของแอพพลิเคชันอื่นๆ ได้เช่นกัน

แอพพลิเคชันจะใช้ฟังก์ชัน GetMessage ในลูปของฟังก์ชัน WinMain เพื่อดึงเมสเสจออกจากคิวของแอพพลิเคชันเอง ฟังก์ชัน GetMessage จะทำงานโดยเริ่มจากดูว่าในควมีเมสเสจหรือไม่ หากมีก็จะดึงเมสเสจที่อยู่แรกสุดไป แต่หากไม่มีเมสเสจอยู่ในคิว ก็จะเกิดการรอเมสเสจขึ้น และปล่อยการควบคุมกลับคืนไปให้วินโดว์ เพื่อแบ่งการควบคุมให้แอพพลิเคชันอื่นสามารถทำงานต่อไปได้

เมื่อฟังก์ชัน WinMain ของแอพพลิเคชันได้รับเมสเสจจากคิวแล้ว ก็ต้องเรียกใช้ฟังก์ชัน Dispatch-Message เพื่อแจกแจงเมสเสจ และส่งเมสเสจนั้นไปยังฟังก์ชันประจำตัววินโดว์ต่างๆ การทำเช่นนี้จะเป็นการถ่ายเทการควบคุมให้กับฟังก์ชันประจำวินโดว์ที่จะประมวลผลเมสเสจนั้น หลังจากที่ทำงานที่เกี่ยวข้องกับเมสเสจนี้เรียบร้อยแล้วก็ถ่ายเทการควบคุมกลับไปให้ฟังก์ชันหลักในแอพพลิเคชัน เพื่อดึงเอาเมสเสจต่อไปในคิวมาประมวลผลต่อ

ไปข้อควรระวัง ฟังก์ชันประจำวินโดว์ไม่ควรคาดว่าเมสเสจใดจะเข้ามาก่อน เพราะวินโดว์สามารถส่งเมสเสจมาในลำดับอย่างใดก็ได้

ส่วนการรับข้อมูลจากคีย์บอร์ดนั้น วินโดว์จะรับเข้ามาในลักษณะของรหัสของคีย์ที่กด ดังนั้นจึงต้องมีการใช้ฟังก์ชัน TranslateMessage เพื่อแปลงรหัสของคีย์เหล่านั้นให้เป็นรหัสของคีย์เหล่านั้นให้เป็นรหัสของตัวอักษร เมื่อแปลงได้แล้ว ฟังก์ชันนี้ก็จะส่งเมสเสจของตัวเองเข้าไปในคิว พร้อมทั้งรหัสตัวอักษรที่ผู้ใช้กดเข้ามาเป็นพารา-มิเตอร์ของเมสเสจนั้น

2.3.3.3 การแปลและตีความเมสเสจ

โดยทั่วไปแล้วจะมีการใช้ฟังก์ชัน TranslateMessage นี้กับทุกเมสเสจที่เข้ามา โดยที่หากไม่ใช่เมสเสจที่เกี่ยวกับคีย์บอร์ดก็จะไม่มีผลอันใด

ตัวอย่างต่อไปนี้ก็จะเป็นการแสดงถึงหน้าตาของเมสเสจลูปที่อยู่ในฟังก์ชัน WinMain ของแอพพลิเคชัน เมสเสจลูปจะดึงเมสเสจออกจากคิวและแจกจ่ายดังนี้

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

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

ส่วนแอพพลิเคชันที่มีการใช้คีย์ทันใจ (accelerator key) ของเมนู ก็ต้องมีการสร้างตารางคีย์ทันใจที่ใช้ไว้ในไฟล์รีซอร์ส จากนั้นก็ต้องมีการโหลดตารางนี้เข้ามาโดยใช้ฟังก์ชัน LoadAccelerator โดยจะทำให้ลักษณะของลูปในการรับและส่งผ่านเมสเสจเป็นดังนี้

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

ที่ต้องมีการตรวจสอบว่าค่าที่ได้กลับมาจากการเรียกใช้ฟังก์ชัน TranslateAccelerator นั้น ก็เพราะว่า จากเมสเสจที่แปลงแล้วนั้น ไปยังฟังก์ชันประจำวินโดว์เอง ดังนั้นจึงไม่ต้องมาเรียกใช้ฟังก์ชัน TranslateMessage และฟังก์ชัน DispatchMessage อีก

2.3.3.4 การตรวจสอบเมสเสจ

ฟังก์ชัน PeekMessage ใช้เพื่อมองหาและตรวจสอบเมสเสจบางเมสเสจในระหว่างการทำงานที่กินเวลาโดยไม่ต้องการมีผลกระทบกระเทือนต่อการเป็นอยู่ของเมสเสจ เมื่อเรียกใช้ฟังก์ชันนี้ ถ้ามีเมสเสจในคิวฟังก์ชันนี้จะให้ค่าที่ไม่เป็นศูนย์กลับมา ทำให้เราสามารถประมวลผลเมสเสจได้ โดยไม่ต้องกลับไปที่ลูปหลักของฟังก์ชัน WinMain ประโยชน์ของฟังก์ชันนี้ก็คือ เมื่อมีการทำงานบางอย่างที่ต้องกินเวลามาก เช่น การอ่านเขียนดิสก์ ก็สามารถใช้ฟังก์ชันนี้ในการตรวจสอบว่า ขณะนี้ผู้ใช้มีการกดคีย์เพื่อยกเลิกการทำงานหรือไม่

2.3.3.5 การส่งและส่งผ่านเมสเสจ

ฟังก์ชัน PostMessage และ SendMessage มีหน้าที่ในการส่งเมสเสจไปยังฟังก์ชันประจำวินโดว์ของตัวเองหรือฟังก์ชันประจำวินโดว์ของแอพพลิเคชันอื่น นอกจากนี้ก็ยังมีฟังก์ชัน PostAppMessage ที่ทำงานแบบเดียวกันกับฟังกชัน PostMessage แต่เป็นการส่งผ่านโดยอาศัยแฮนเดิลโมดูลของแอพพลิเคชันแทน

ฟังก์ชัน PostMessage จะส่งเมสเสจผ่านไปทางคิวของแอพพลิเคชัน(เราจะขอเรียกการส่งแบบนว่าการส่งผ่าน) โดยที่ผู้ส่งจะยังไม่เสียการควบคุม และผลของเมสเสจที่ส่งไปนั้นก็ยังไม่เกิดขึ้นจนกว่าแอพพลิเคชันนั้นๆ จะดึงเมสเสจที่ส่งไปนั้นขึ้นมา ส่วน SendMessage จะส่งเมสเสจลัดคิวไปยังฟังก์ชันประจำวินโดว์ที่ต้องการทันที (โดยเราจะขอเรียกการส่งแบบนี้ว่าการส่ง) โดยที่ไม่ต้องคิวของแอพพลิเคชัน ซึ่งการส่งเมสเสจไปแบบนี้ ผู้ที่ส่งจะสูญเสียการควบคมไปให้ฟังก์ชันประจำวินโดว์ที่ส่งเมสเสจให้ และเมื่อฟังก์ชันนั้นทำงานเสร็จแล้วก็จะคืนการควบคุมกลับมา ส่วนค่าที่ได้กลับมาของการเรียกใช้ SendMessage นั้นคือค่าที่ฟังก์ชันประจำวนโดว์นั้นส่งค่ากลับมา แต่ถ้าเป็นค่าที่ได้กลับมาจากฟังก์ชัน PostMessage เป็นเพียงว่าสามารถส่งเมสเสจนั้นไปได้สำเร็จหรือไม่

2.3.3.6 การหลีกเลี่ยงการเกิดติดตาย (dead lock) ที่เกิดจากเมสเสจ

แอพพลิเคชันอาจเกิดเดดล็อกได้ ถ้าแอพพลิเคชันที่ได้การควบคุมมาจากฟังก์ชันอื่นทาง SendMessage แล้วส่งต่อการควบคุมไปอีก ซึ่งการส่งต่อการควบคุมต่อไปนั้น อาจไม่ต้องใช้ SendMessage ก็ได้ การใช้ฟังก์ชันข้างล่างเหล่านี้ก็เปรียบเสมือนการผ่านการควบคุมไปยังแอพพลิเคชันอื่น

- DialogBox
- DialogBoxIndirect
- DialogBoxIndirectParam
- DialogBoxParam
- GetMessage
- PeekMessage
- Yield

ฟังก์ชันที่เรียกใช้งาน SendMessage เพื่อส่งเมสเสจไปให้อีกฟังก์ชันทำงานนั้น จะไม่สามารถทำงานต่อไปได้จนกว่าฟังก์ชันที่รับเมสเสจนั้นจะทำงานเสร็จและให้ค่ากลับมา แต่ถ้าฟังก์ชันนั้นเรียกใช้งานฟังก์ชันต่างๆ ช้างบนก็

จะเกิดกรณีเดดล็อกขึ้น ดังนั้น ก่อนการเรียกใช้งานฟังก์ชันเหล่านั้น ควรตรวจสอบดู้เสียก่อนว่าเมสเสจที่ได้รับนั้นมาจาก SendMessage หรือไม่ โดยใช้ฟังก์ชัน IsSendMessage ถ้าให้ค่ากลับมาเป็นจริง ก็หมายความว่าเป็นเมสเสจที่ได้มาจาก SendMessage ให้เรียกใช้งาน ReplyMessage ก่อนที่จะส่งการควบคุมต่อไป

2.3.3.7 ฟังก์ชันประจำวินโดว์

ฟังก์ชันประจำวินโดว์ เป็นฟังก์ชันที่ทำหน้าที่ประมวลผลเมสเสจของวินโดว์ที่ใช้คลาสนั้น เมสเสจจะถูกส่งมาโดยวินโดว์ เมื่อมีอินพุท (เช่น เมาส์ คีย์บอร์ด หรือ ไทเมอร์) เข้ามา หรือไม่ วินโดว์ต้องการให้วินโดว์ทำงานบางอย่าง เช่น วาดพื้นที่ใช้งานใหม่ นอกจากนี้มีหน้าที่ในการรับอินพุทจากเมสเสจต่างๆ แล้วฟังก์ชันประจำวินโดว์ยังมีหน้าที่ในการรับข้อมูลเมื่อระบบถูกเปลี่ยนแปลงโดยแอพพลิเคชันอื่น เช่น มีการแก้ไขไฟล์ WIN.INI หรือการทำตามคำร้องขอบางอย่าง เช่น การเปลี่ยนแปลงเมนูก่อนแสดงให้ผู้ใช้เห็น หรือการรับทราบว่าขณะนี้ตัวเองได้เป็นวินโดว์ที่แอคทีฟแล้ว รวมทั้งข้อมูลอย่างอื่นๆ อีกมากมาย

แม้ว่าเมสเสจส่วนใหญ่จะถูกส่งมาจากวินโดว์ แต่วินโดว์ด้วยกันเองก็สามารถส่งเมสเสจถึงกันได้(หรือจากตัวเองก็ได้) ทั้งนี้เพื่อจะได้ทราบความเป็นไปของวินโดว์อื่นๆ ตามปกติแล้วการรับเมสเสจของฟังก์ชันประจำวินโดว์จะเป็นไปเรื่อยๆ จนกว่าวินโดว์นั้นจะถูกทำลาย (ถ้าคลาสนั้นยังมีวินโดว์อื่นใช้งานอยู่อีก เมสเสจต่างๆ ก็ยังส่งไปยังวินโดว์ที่ยังมีชีวิตเหล่านั้นอยู่)

เนื่องจากการทำงานของแอพพลิเคชันนั้น ขึ้นอยู่กับเมสเสจที่แต่ละวินโดว์ได้รับ ดังนั้นตัวฟังก์ชันประจำวินโดว์จึงเป็นส่วนที่บังคับการดำเนินไปของวินโดว์นั้นๆ เช่น ในแอพพลิเคชันที่มีการเปิดไฟล์ เมื่อผู้ใช้เลือกคำสั่ง Open ฟังก์ชันประจำวินโดว์ก็จะเป็นตัวสั่งการว่า ต่อไปให้ทำการเปิดไฟล์ตามที่ผู้ใช้ต้องการ หรือผู้ใช้มีการเลื่อนสกรอลลบาร์ ฟังก์ชันประจำวินโดว์ก็ต้องเลื่อนข้อมูลที่อยู่นอกวินโดว์ขึ้นมาให้ผู้ใช้ดูตามต้องการ

ส่วนเมสเสจที่เข้ามายังฟังก์ชันประจำวินโดว์แต่ไม่ได้ใช้ประโยชน์อะไรนั้น ต้องมีการส่งต่อไปให้ฟังก์ชัน DefWindowProc เป็นตัวประมวลผลเมสเสจเหล่านั้น โดยเฉพาะอย่างยิ่งเมสเสจที่เกิดจากนอกพื้นที่ใช้ งานของวินโดว์ (ซึ่งเป็นงานของวินโดว์ เช่น การคลิกที่ไตเติ้ลบาร์ หรือ การลดขนาดวินโดว์) ดังนั้นใน switch case ของทุกฟังก์ชันประจำวินโดว์ควรมี DefWindowProc นี้อยู่เสมอ

2.3.3.8 เมสเสจของวินโดว์

เมสเสจเป็นกลุ่มของข้อมูลอย่างหนึ่ง ที่ถูกส่งไปยังฟังก์ชันประจำวินโดว์ต่างๆ ที่ทำงานในวินโดว์ขณะนั้น ประกอบด้วย 4 ส่วนคือ แฮนเดิลของวินโดว์ที่ต้องการส่งไป ตัวเมสเสจ ID พารามิเตอร์ขนาด 16 บิต และสุดท้ายคือพารามิเตอร์ขนาด 32 บิต สำหรับพารามิเตอร์สองตัวหลังนั้นจะเป็นข้อมูลของอะไรก็จะขึ้นกับชนิดของเมสเสจนั้น ซึ่งบางเมสเสจก็ไม่ได้ใช้พารามิเตอร์ครบทั้งสองตัว

ลักษณะของฟังก์ชันประจำวินโดว์ที่มีเมสเสจเป็นอากิวเมนต์ดังนี้

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

hWnd เป็นพารามิเตอร์ที่เก็บแฮนเดิลของวินโดว์ที่รับเมสเสจนี้ wMsg เป็นตัวเมสเสจที่ส่งมา wParam เป็นพารามิเตอร์เพิ่มเติมของเมสเสจที่ส่งมา โดยเป็นข้อมูล 32 บิต ตัวอย่างของการใช้งานพารามิเตอร์สองตัวหลัง เช่น ถ้ามีการเลื่อนเมาส์ก็จะเกิดเมสเสจ WM_MOUSEMOVE และ wParam ก็จะมีข้อมูลของปุ่มที่ถูกกด ส่วน lParam ก็จะเป็นตำแหน่งของเมาส์ที่ถูกเลื่อนมา

2.3.3.9 วงจรชีวิตของวินโดว์

หน้าที่หลักของวินโดว์คือ การรับอินพุทและการแสดงเอาต์พุท ดังนั้นวงจรชีวิตของวินโดว์จึงเริ่มขึ้นเมื่อแอพพลิเคชันต้องการรับอินพุทและเอาต์พุท และจะสิ้นสุดเมื่อไม่ต้องการหรือจลแอพพลเคชัน โดยปกติแล้ววินโดว์ก็จะคงอยู่จนกว่าจะจบแอพพลเคชัน แต่วินโดว์ที่ใช้งานเฉพาะบางอย่างก็จะมีชีวิตสั้นๆ เช่น วินโดว์ของกรอบข้อวาม

วินโดว์เริ่มขึ้นจากขั้นตอนการสร้าง โดยการส่งคลาสของวินโดว์ที่ทำการขึ้นทะเบียนไว้แล้วให้กับฟังก์ชัน CreateWindow จากนั้นวินโดว์ก็จะไปเตรียมข้อมูลเกี่ยวกับวินโดว์ใหม่นั้น แล้วให้ค่าตัวเลขกลับมาสู่แอพพลิเคชัน สำหรับใช้ในการอ้างถึงวินโดว์ภายหลัง เรียกตัวเลขนี้ว่า แฮนเดิลของวินโดว์

เมสเสจแรกที่ถูกส่งไปยังฟังก์ชันประจำวินโดว์คือ WM_CREATE ซึ่งเป็นเมสเสจที่บอกว่าให้ทำการเตรียมตัวบางอย่าง ก่อนการแสดงวินโดว์ เช่น การจองหน่วยความจำ หรือเปิดไฟล์ข้อมูลเตรียมไว้ ข้อมูลที่ส่งมาด้วยก็จะอยู่ใน lParam เป็นพอยน์เตอร์ไปยังโครงสร้าง CREATSTRUCT ซึ่งเก็บข้อมูลเกี่ยวกับวินโดว์นั้นอยู่ เมสเสจ WM_CREATE และ WM_NCCREATE นั้ถูกส่งให้วินโดว์โดยผ่านคิวของแอพพลิเคชัน ซึ่งก็หมายความว่าจะทำงานก่อนลูปหลักของแอพพลิเคชัน

การที่จะเริ่มใช้งานวินโดว์ได้นั้น จะต้องแสดงวินโดว์นั้นขึ้นมาเสียก่อน หากวินโดว์ไม่ได้ถูกสร้างด้วยสไตล์ WS_VISIBLE แล้ว ก็ต้องใช้ฟังก์ชน ShowWindow ในการแสดง แต่สำหรับวินโดว์หลักของแอพพลิเคชันแล้ว ไม่ควรกำหนดให้เป็นสไตล์ WS_VISIBLE ควรใช้ ShowWindow แทน โดยใช้ตัวแปร nCmdShow เป็นพารามิเตอร์ของฟังก์ชันเพื่อบอกว่าควรแสดงหรือไม่

เมื่อวินโดว์สิ้นสุดการทำงานหรือจลแอพพลิเคชันแล้ว ก็ต้องยกเลิกหรือทำลายวินโดว์นั้น ด้วยฟังก์ชัน DestroyWindow โดยฟังก์ชันจะลบวินโดว์นั้นออกจากหน้าจอ และส่งเมสเสจ WM_DESTROY และ WM_NCDESTROY ไปยังฟังก์ชันประจำวินโดว์ (อีกทางหนึ่งที่ฟังก์ชันของวินโดว์จะได้รับเมสเสจนี้คือ มีเมสเสจ WM_CLOSE ส่งเข้าไปยังฟังก์ชัน DefWindowProc)

สำหรับวินโดว์ที่เป็นวินโดว์หลักของแอพพลิเคชันควรจะถูกทำลายเป็นวินโดว์สุดท้าย และควรมีการบอกให้จอแอพพลิเคชันด้วย โดยเมื่อวินโดว์หลักได้รับ WM_DESTROY แล้วก็ให้เรียกใช้ฟังก์ชัน PostQuitMessage เพื่อส่ง WM_QUIT ไปยังคิวของแอพพลิเคชัน และเมื่อเมนลูปอ่านเมสเสจขึ้นมาก็จะเป็นการจบแอพพลิเคชัน

2.4 การสร้างแอพพลิเคชัน

ทั้งการสร้างแอพพลิเคชันบนวินโดวส์ และโอเอสทูนั้นล้วนมีขั้นตอนเหมือนกัน เนื่องจากใช้ภาษาในการทำโปรแกรมเดียวกันคือภาษาซี นอกจากนั้นส่วนประกอบซึ่งเป็นซอร์สไฟล์ประเภทต่างๆ นั้นยังเหมือนกันด้วย

2.4.1 ส่วนประกอบทางโปรแกรมของแอพพลิเคชัน

ส่วนประกอบที่รวมกันเป็นแอพพลิเคชันนั้นสามารถแบ่งออกเป็นส่วนๆ ได้คือ

2.4.1.1 ซอร์สไฟล์ (source file)

เป็นส่วนที่เป็นโปรแกรมของแอพพลิเคชัน ซึ่งเขียนด้วยภาษาซีหรือซีพลัสพลัส (C/C++) รวมทั้งภาษาแอสเซมบลี้ (assembly) ซึ่งประกอบไปด้วยฟังก์ชันหลักของแอพพลิเคชัน ฟังก์ชันประจำของแต่ละวินโดว์ และฟังก์ชันในการใช้งานอื่นๆ ที่ประกอบกันเป็นแอพพลิเคชัน ไฟล์ในกลุ่มนี้ได้แก่ไฟล์ที่มีนามสกุลเป็น .C, .CPP และ .ASM เป็นต้น

2.4.1.2 รีซอสสคริปต์ไฟล์ (resource script file)

รีซอสสคริปต์ไฟล์ ปกติจะมีนามสกุลเป็น .RC เป็นไฟล์ที่เก็บรูปแบบของทรัพยากร (resource) ต่างๆ ที่ใช้ในการสร้างแอพพลิเคชัน เช่น เมนูและกรอบข้อความ รวมทั้งกำหนดว่าบิตแมพ (bitmap) และไอคอน (icon) ที่ต้องการใช้ในแอพพลิเคชันถูกเก็บไว้ในไฟล์ชื่ออะไร และจะกำหนดชื่อหรือค่าไว้ด้วย เพื่อให้แอพพลิเคชันสามารถเรียกใช้ได้อย่างถูกต้อง

2.4.1.3 ไฟล์ลักษณะโมดูล (module definition file)

ไฟล์นี้จะมีนามสกุลเป็น .DEF มีหน้าที่ในการช่วยโปรแกรมลิงค์ (linker) สร้างไฟล์ที่ใช้ในการสร้างไฟล์ทำงานของแอพพลิเคชันในรูป .EXE โดยไฟล์นี้จะบอกคุณสมบัติต่างๆ ของโค้ดเซกเมนต์ (code segment) และดาต้าเซกเมนต์ (data segment) ของโปรแกรม ขนาดของฮีพ (heap) และสแตก (stack) รวมทั้งข้อมูลอื่นๆ

2.4.1.4 เมคไฟล์ (make file)

เมคไฟล์เป็นตัวกำกับการทำงานของโปรแกรมต่างที่ใช้ในการสร้างแอพพลิเคชัน โดยประกอบไปด้วยคำสั่งต่างๆ ที่จะสั่งตัวแปลภาษา (compiler) ทำงานเมื่อเกิดการเปลี่ยนแปลงของไฟล์ต้นฉบับ

2.4.2 ขั้นตอนการสร้างแอพพลิเคชัน

การสร้างแอพพลิเคชันประกอบไปด้วยขั้นตอนดังนี้

2.4.2.1 เขียนโปรแกรมต้นฉบับ

2.4.2.2 สร้างทรัพยากรของแอพพลิเคชัน

ด้วยโปรแกรมสร้างทรัพยากรแบบต่างๆ

2.4.2.3 เขียนสคริปต์สำหรับทรัพยากร

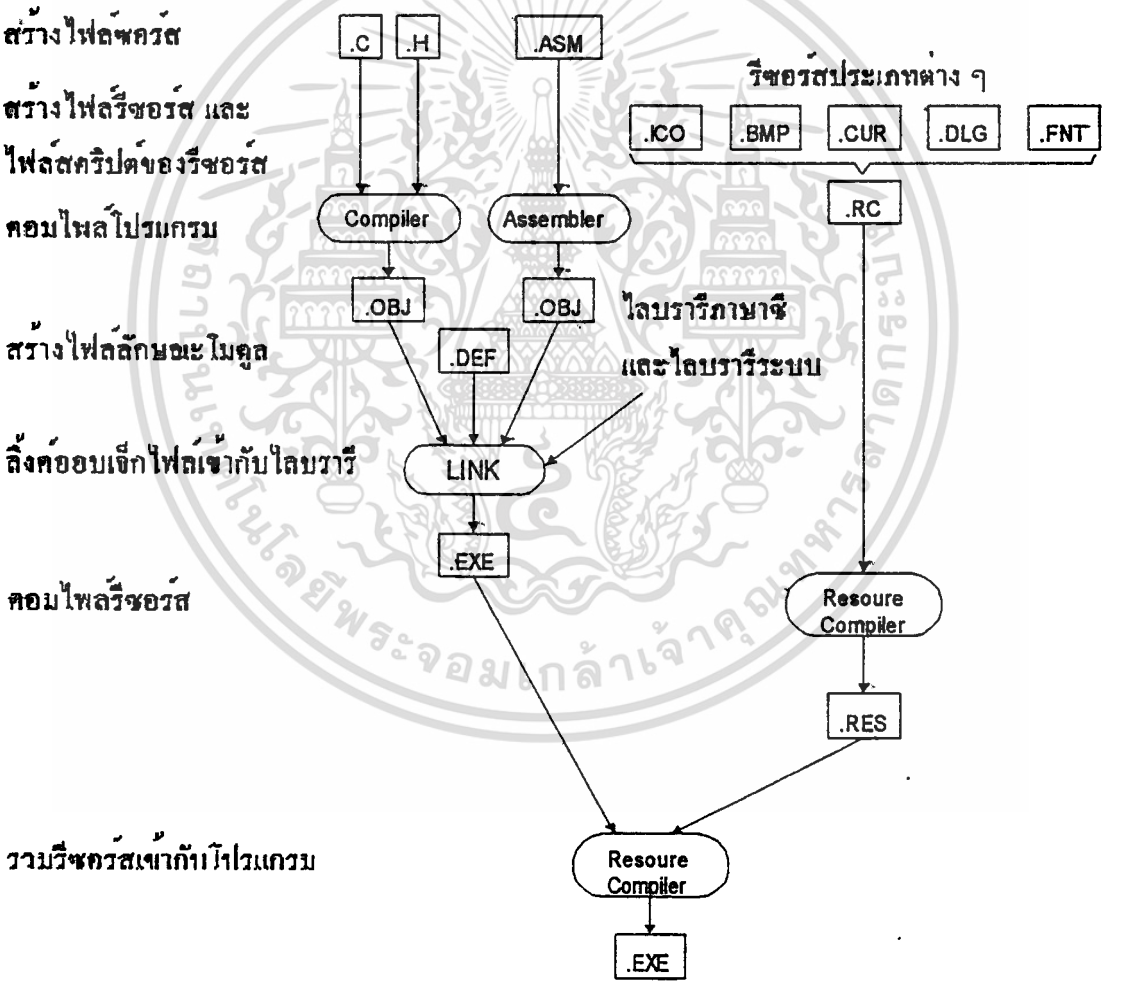
เพื่อชี้แจงการใช้ทรัพยากรต่างๆ เพื่อบรรจุไว้ในแอพพลิเคชัน รวมทั้งการตั้งชื่อทรัพยากรต่างๆ ที่ใช้

2.4.2.4 เขียนไฟล์กำหนดลักษณะของโมดูล

เพื่อกำหนดลักษณะต่างๆ ของแอพพลิเคชัน

2.4.2.5 ใช้โปรแกรมที่ช่วยในการสร้างแอพพลิเคชันจากเมคไฟล์

ซึ่งจำดำเนินการเรียกใช้ตัวแปรภาษา ตัวแปรภาษาสคริปต์ของทรัพยากร แล้วลิ้งค์รวมกันเป็นแอพพลิเคชัน



รูปที่ 2.9 ขั้นตอนการสร้างแอพพลิเคชันแบบวินโดว์

2.5 การจัดการกราฟิก

การจัดการกราฟิกในโอเอสทูนั้นจะทำผ่านส่วนที่เรียกว่า GPI ส่วนในวินโดวส์นั้นจะทำผ่านส่วนที่เรียกว่า GDI ซึ่งมีรายละเอียดดังนี้

2.5.1 GPI (Graphics Programming Interface)

PI ในโอเอสทู พรีเซนเตชันแมนเนเจอร์ (OS/2 Presentation Manager, PM) มีฟังก์ชันต่างๆ มากกว่า 200 ฟังก์ชัน ซึ่งสามารถใช้สร้าง, แสดง หรือพิมพ์กราฟิก ได้อย่างมีประสิทธิภาพ

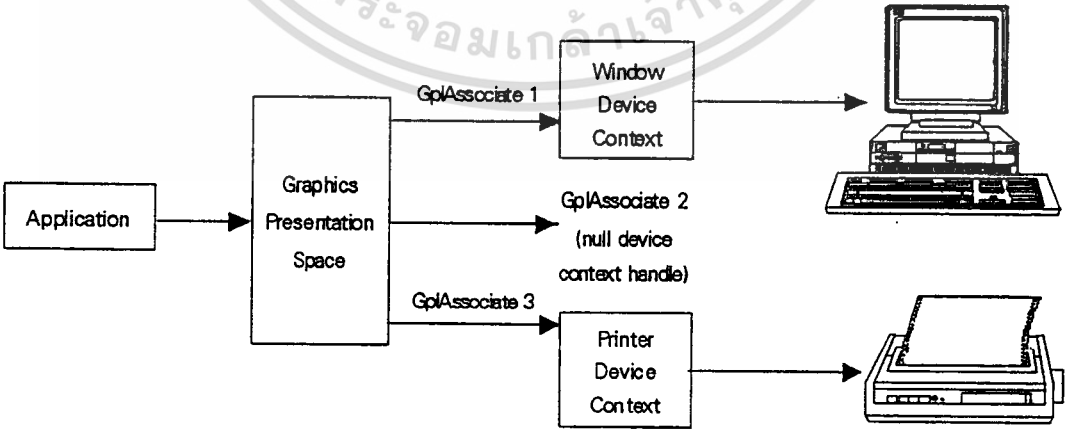
ฟังกชันส่วนใหญ่ของ GPI จะเป็นประเภทไม่ขึ้นกับอุปกรณ์ (device-independence) ซึ่งจะติดต่อกับฮาร์ดแวร์ผ่านทาง พรีเซนเตชันสเปซ (presentation spaces), ดีไวซ์คอนเทกต์ (device context) และกราฟิกพริมิทีฟ (graphics primitives) ต่างๆ

2.5.1.1 *พรีเซนเตชันสเปซ และดีไวซ์คอนเทกต์*

พรีเซนเตชันสเปซ เป็นโครงสร้างข้อมูลอย่างหนึ่ง ซึ่งเปรียบเสมือนกระดาษว่างๆ แผ่นหนึ่ง ซึ่งใช้สำหรับวาดกราฟิกต่างๆ ก่อนที่จะส่งไปยังอุปกรณ์แสดงผล โดยที่อุปกรณ์แสดงผลเหล่านั้นอาจจะเป็นจอภาพ, เครื่องพิมพ์ หรือบิตแมพในหน่วยความจำ ก็ได้ โปรแกรมที่ทำงานบน PM จะต้องส่งคำสั่งวาดกราฟิกไปยังพรีเซนเตชันสเปซเสมอ

ตัวโอเอสทู และไดรเวอร์ จะทำการนำข้อมูลกราฟิกต่างๆ ที่อยู่ในพรีเซนเตชันสเปซ มาแสดงออกยังอุปกรณ์ที่ผู้ใช้ต้องการ โดยผ่านทางดีไวซ์คอนเทกต์ที่เชื่อมต่อพรีเซนเตชันสเปซเข้ากับอุปกรณ์นั้นๆ

ดีไวซ์คอนเทกต์ เป็นโครงสร้างข้อมูลเช่นกัน โดยจะมีหน้าที่แปลงข้อมูลกราฟิกจากพรีเซนเตชันสเปซไปเป็นคำสั่งที่ใช้สำหรับสั่งอุปกรณ์แสดงผล รูปที่ 2.10 แสดงการทำงานของคำสั่งกราฟิกในโปรแกรม ซึ่งจะผ่านทางพรีเซนเตชันสเปซ, ดีไวซ์คอนเทกต์ และอุปกรณ์แสดงผล



รูปที่ 2.10 การทำงานของคำสั่งกราฟิก

2.5.1.2 กราฟิกพริมิทีฟ

กราฟิกพริมิทีฟ เปรียบเสมือนชิ้นส่วนย่อย (Building Block) ของโปรแกรมแบบ PM ซึ่งใช้สำหรับสร้างภาพ กราฟิกพริมิทีฟประกอบไปด้วยหลายชนิดด้วยกัน

2.5.1.2.1 ลักษณะของพริมิทีฟ (Primitive Attributes)

กราฟิกพริมิทีฟแต่ละตัว จะมีลักษณะเฉพาะ (attribute) จำนวนหนึ่ง ซึ่งกำหนดลักษณะต่างๆ ของ พริมิทีฟนั้น เช่น พริมิทีฟเส้น จะมีลักษณะของสี, ความกว้าง หรือความประ เช่น เส้นจุด, เส้นประ, เส้นทึบ หรือเส้นโปร่ง ซึ่งสิ่งเหล่านี้จะรวมเรียกว่า ลักษณะของพริมิทีฟ

โดยปกติแล้ว ลักษณะของพริมทีฟจะใช้กำหนดกลุ่มของพริมิทีฟ แทนทีจะเป็นแต่ละพริมิทีฟ เช่น ถ้ามีการกำหนดให้ลักษณะของเส้นเป็นเส้นประ เส้นทุกเส้นหลังจากนั้นก็จะเป็นเส้นประ เป็นต้น

ทุกๆ ลักษณะของพริมิทีฟ จะมีค่าปกติอยู่ ซึ่งจะกำหนดเมื่อมีการสร้างพรีเซนเตชันสเปซ และเมื่อมีการเปลี่ยนลักษณะ ค่าเหล่านั้นก็จะคงอยู่จนกระทั้งมีการเปลี่ยนใหม่ ดังได้กล่าวแล้วข้างต้น

2.5.1.2.2 พริมิทีฟเส้น และเส้นโค้ง (Line and Arc Primitives)

พริมิทีฟเส้น และเส้นโค้ง เป็นองค์ประกอบเบื้องต้นของโปรแกรมส่วนใหญ่ โปรแกรมวาดภาพอย่างง่ายสามารถใช้เพียงพริมิทีฟเส้น และเสนโค้งเป็นองค์ประกอบได้ หรือแม้ในโปรแกรมที่ซับซ้อน เช่น โปรแกรมแคด (CAD) ก็สามารถใช้พริมิทีฟเหล่านี้เป็นองค์ประกอบ เพื่อวาดภาพที่ซับซ้อนได้

ลักษณะของพริมิทีฟเส้น และเส้นโค้ง จะเก็บอยู่ในโครงสร้างชื่อ LINEBUNDLE ซึ่งจะมี ความกว้าง, ชนิดของเส้น, สี และการผสม (mix) (ดูหัวข้อ 2.5.1.3)

2.5.1.2.3 พริมิทีฟพื้นที่ (Area Primitives)

คำว่า พื้นที่ ในที่นี้มีความหมายถึงรูปร่างที่ปิด (มีเส้นล้อมรอบ) ซึ่งอาจจะเกิดขึ้นจากพริมิทีฟเส้น และเส้นโค้ง การสร้างพริมิทีฟพื้นที่ จะใช้ฟังก์ชันที่สามารถกำหนด และวาดพื้นที่ได้

พริมทีฟพื้นที่จะเก็บในโครงสร้างชื่อ AREABUNDLE ลักษณะของพริมิทีฟพื้นที่ ประกอบไปด้วย สัญลักษณ์ในการระบาย และจุดอ้างอิง (ในการระบายพื้นที่ สามารถกำหนดสัญลักษณ์ที่ใช้ระบายได้ เช่น ระบายเป็นรูปสามเหลี่ยมเล็กๆ หลายๆ รูปเรียงต่อกัน ซึ่งต้องกำหนดจุดเริ่มต้นระบาย), ชนิดของการระบาย, สีระบาย, สีฉากหลัง และการผสม ของสีระบาย และสีฉากหลัง

พื้นที่ อาจกล่าวได้อีกอย่างว่า กลุ่มของพื้นที่ เพราะการสร้างพื้นที่ จะต้องเริ่มด้วยฟังก์ชัน GpiBeginArea และจบด้วยฟังก์ชัน GpiEndArea เสมอ โดยระหว่างสองฟังก์ชันนี้ จะใช้พริมิทีฟเสนและ เส้นตรงอย่างไรก็ได้ ซึ่งจะทำให้สามารถสร้างพริมิทีฟพื้นที่ได้อย่างสะดวก

2.5.1.2.4 พริมิทีฟรูปหลายเหลี่ยม (Polygon Primitives)

รูปหลายเหลี่ยม (polygon) หมายถึงรูปปิดที่ล้อมรอบด้วยเส้นตรงหลายๆ เส้น พริมิทีฟรูปหลายเหลี่ยม หมายถึง รูปหลายเหลี่ยม ที่ประกอบด้วยการระบายสี หรือมีเส้นล้อมรอบด้วย

ฟังก์ชัน GpiPolygons ใช้สร้างพริมิทีฟรูปหลายเหลี่ยม ซึ่งประกอบด้วยเส้นตรงหลายๆ เส้นล้อมรอบ โดยต้องใช้นอกกลุ่มของพื้นที่ (ดูหัวข้อ 2.5.1.2.3) อย่างไรก็ตาม พริมิทีฟพื้นที่สามารถติดต่อกัน, ตัดกัน หรือไม่เกี่ยวข้องกันก็ได้

2.5.1.2.5 พริมิทีฟข้อความ และรูปแบบตัวอักษร (Character String Primitives and Fonts)

พริมิทีฟข้อความ ใช้สำหรับสร้างข้อความ (text) ส่วนรูปแบบตัวอักษร คือ ชุดของลักษณะตัวอักษร ซึ่งตัวอักษรแต่ละตัวในชุด จะมีรูปร่างคล้ายๆ กัน และจะมีความสูง และน้ำหนักของเส้นเท่าๆ กัน

ฟังก์ชันเกี่ยวกับพริมิทีฟข้อความ และรูปแบบตัวอักษร จะสามารถใช้เพื่อหารูปแบบตัวอักษรทั้งหมดที่มี, เลือกกำหนดรูปแบบตัวอักษร, วาดข้อความตามรูปแบบตัวอักษรที่เลือก รวมทั้งแก้ไขข้อความได้

พริมิทีฟข้อความ และรูปแบบตัวอักษร จะเก็บในโครงสร้างชื่อ CHARBUNDLE โดยลักษณะของพริมิทีฟข้อความประกอบด้วย ชุดของตัวอักษร, ความละเอียด, เซล (cell), มุมเอียง, ทิศทาง, การจัดตัวอักษร, สี และการผสม

2.5.1.2.6 พริมิทีฟเครื่องหมาย (Marker Primitives)

พริมิทีฟเครื่องหมาย คือรูปที่ใช้สำหรับแสดงจุดที่ต้องการเน้น เช่น จุดสูงสุดของกราฟ เป็นต้น ใน GPI จะมีรูปเครื่องหมายอย่างง่ายให้ใช้หลายชนิดด้วยกัน เช่น สี่เหลี่ยม, วงกลม, รูปดาว, รูปข้าวหลามตัด, กากะบาด เป็นต้น

พริมิทีฟเครื่องหมาย จะเก็บลักษณะไว้ในโครงสร้างชื่อ MARKERBUNDLE ซึ่งจะประกอบไปด้วย รูปร่างของเครื่องหมาย, ขอบของเครื่องหมาย, กลุ่มของเครื่องหมาย, สี และการผสม

2.5.1.3 ลักษณะสี และการผสม (Color and Mix Attributes)

สี และการผสม เป็นลักษณะที่กำหนดให้กับพริมิทีฟ ลักษณะสีจะกำหนดสีของ เส้น, เส้นโค้ง, ตัวอักษร หรือภาพ ส่วนลักษณะการผสม จะกำหนดว่าสีของแต่ละพริมิทีฟนั้น จะรวมกับสีของภาพที่วาดอยู่ก่อนอย่างไร เช่น วาดทับ, นำสีมาผสมกัน เป็นต้น ซึ่งจะทำให้ลักษณะของรูปที่วาดใหม่ลงไปนั้น ขึ้นอยู่กับรูปที่วาดอยู่ก่อนด้วย

บางพริมิทีฟจะมีทั้งลักษณะสีวาด และสีฉากหลัง เช่น พริมิทีฟข้อความ จะใช้ลักษณะสีวาดกำหนดสีของตัวอักษร และใช้ลักษณะสีฉากหลังกำหนดสีของฉากหลังที่อยู่รอบๆ ตัวอักษร และพริมิทีฟที่มีลักษณะสีวาด และสีฉากหลัง จะมีลักษณะการผสมของสีวาด และของสีฉากหลังด้วย

2.5.1.4 บิตแมพ (Bit Maps)

บิตแมพ คือรูปภาพที่เก็บในหน่วยความจำ การสร้างบิตแมพสามารถทำได้โดยใช้โปรแกรม Icon Editor หรือโปรแกรมอื่นๆ หรือใช้ฟังก์ชัน GPI การเก็บบิตแมพสามารถเก็บในดีไวซ์คอนเทกต์แบบหน่วยความจำ (memory device context) หรือเก็บเป็นไฟล์ในดิสก์ บิตแมพที่ใช้ใน PM สามารถเป็นแบบขาวดำ หรือเป็นแบบสีก็ได้

โดยปกติแล้ว โปรแกรมสามารถใช้บิตแมพเพื่อทำงานเหล่านี้ได้

- เก็บ และแสดงผล รูปภาพ, ไอคอน และ ตัวชี้ (pointer)
- สร้างสัญลักษณ์การระบาย สำหรับ พริมิทีฟพื้นที่ และพาธ (ดูหัวข้อ 2.5.1.6)
- แสดงภาพเคลื่อนไหว สำหรับระบบมัลติมิเดีย (multimedia)

2.5.1.5 เมตาไฟล์ (Metafiles)

เมตาไฟล์ คือไฟล์ที่เก็บคำสั่งวาดภาพระดับต่ำ (ในที่นี้ คำสั่งระดับต่ำ หมายถึงคำสั่งที่ย่อยที่สุด) ที่ใช้สำหรับวาดภาพ

โปรแกรมที่ทำงานบนโอเอสทู สามารถใช้คำสั่งวาดภาพแบบรีเทนชัน (retention) ซึ่งทำให้โปรแกรมสามารถเก็บคำสั่งวาดภาพนั้น เพื่อให้โปรแกรมอื่นที่ทำงานบน PM สามารถนำมาใช้ได้ หรือแม้แต่โปรแกรมที่ไม่ทำงานบน PM ก็สามารถนำมาใช้ได้ถ้าโปรแกรมนั้นสนับสนุนมาตรฐานการแลกเปลี่ยนแบบ มิกซด์ออบเจ็กต์ดอกคิวเมนต์คอนเทนต์อะคริเทคเจอร์ (Mixed Object Document Content Architecture (MO:DCA)) ภาพที่จะนำมาแลกเปลี่ยนกันระหว่างโปรแกรมเหล่านี้ จะเก็บในรูปแบบ เมตาไฟล์

2.5.1.6 พาธ (Paths)

พาธ เป็นวัตถุที่ประกอบไปด้วย ลำดับของพริมิทีฟต่างๆ และฟังก์ชันลักษณะต่างๆ ที่อยู่ระหว่างฟังก์ชัน GpiBeginPath และ GpiEndPath โปรแกรมสามารถใช้พาธเพื่อ กำหนดพื้นที่, กำหนดการตัดรูป (clipping) ที่ซับซ้อน, การวาดเส้นล้อมรอบ หรือเพื่อวาดเส้นหนา เป็นต้น ในความหมายตรง พาธจะเป็นเพียงการวาดเส้น ที่สามารถกำหนดความหนาได้ และการตัดรูปที่เป็นวงกลม, วงรี หรือรูปร่างอื่นๆ ที่ไม่เป็นรูปสี่เหลี่ยมมุมฉาก ประโยชน์อีกอย่างของพาธ คือการวาดตัวอักษรแบบโปร่ง (outline)

2.5.1.7 รีเจียน (Regions)

รีเจียน คือวัตถุรูปสี่เหลี่ยมมุมฉากที่อยู่ในดีไวซ์ ซึ่งสามารถใช้สำหรับตัดรูปผลลัพธ์, วาดรูปสี่เหลี่ยมมุมฉากหลายๆ รูป หรือวาดรูปหลายเหลี่ยม ที่มีส่วนตัดกันเป็นสี่เหลี่ยมมุมฉาก รีเจียนสามารถใช้สำหรับการวาดหน้าจอใหม่ ในวินโดว์ หรือบางส่วนในภาพได้

รีเจียนจะไม่ขึ้นอยู่กับอุปกรณ์ และจะกำหนดโดยตำแหน่งในดีไวซ์ แต่ละรีเจียน จะสร้างขึ้นสำหรับ ดีไวซ์ที่กำลังเชื่อมต่อกับพรีเซนเตชันสเปซอยู่ รีเจียนถือเป็นส่วนหนึ่งของโครงสร้างระบบ โดยจะอยู่ในดีไวซ์คอนเทกต์

2.5.1.8 รีเทนด์กราฟิก (Retained Graphics)

ในโอเอสทู จะมีกราฟิกอยู่ 2 ชนิด คือ รีเทนด์ และ นอนรีเทนด์ (nonretained) รีเทนด์กราฟิกจะเก็บในรูปแบบของพรีเซนเตชันสเปซ ซึ่งจะเก็บกราฟิกทั้งหมด ไว้สำหรับแสดงผล หรือแก้ไข

โปรแกรมสามารถวาดภาพโดยใช้ นอนรีเทนด์กราฟิก โดยใช้ฟังก์ชันกราฟิก ภาพเหล่านั้น จะปรากฏบนอุปกรณ์แสดงผลทันที (เช่น บนวินโดว์) แต่ถ้าบางส่วนของภาพโดนลบ หรือระบบต้องการให้มีการวาดภาพใหม่ โปรแกรมจะต้องใช้ฟังก์ชันเดิมเพื่อวาดรูปเดิมอีก ซึ่งอาจจะต้องวาดหลายครั้งก็ได้ ซึ่งเป็นข้อเสียของนอนรีเทนด์กราฟิก

ในทางตรงข้าม รีเทนด์กราฟิกมีจุดเด่นอยู่หลายประการ เช่น ใช้งานได้ง่าย, มีความยืดหยุ่นสูง, แก้ไขได้ง่าย, มีคำสั่งกราฟิกมากกว่า รวมทั้งมีการจัดแยกประเภทไว้เป็นอย่างดี สิ่งที่อาจจะเป็นจุดด้อยของรีเทนด์กราฟิก คือ ใช้หน่วยความจำมากกว่าในการเก็บข้อมูล

โปรแกรมที่ทำงานบน PM จะเก็บกราฟิกในรูปแบบเซกเมนต์ (segments) ซึ่งปกติจะหมายถึงการจัดกลุ่มและการเก็บของกราฟิกพริมิทีฟต่างๆ รวมทั้งลักษณะของพริมิทีฟด้วย อย่างไรก็ตาม กราฟิกเซกเมนต์ อาจจะหมายถึงอย่างอื่นในบางครั้ง

2.5.1.9 คอร์เรลเลชัน (Correlation)

คอร์เรลเลชัน คือการตรวจสอบว่าพริมิทีฟใด หรือกลุ่มของพริมิทีฟใด อยู่ในตำแหน่งที่จะต้องแสดงผลบนจอภาพ โดยปกติแล้ว คอร์เรลเลชัน จะใช้ในโปรแกรมที่ทำงานแบบกราฟิก เช่น เพื่อเลือกพริมิทีฟ หรือกลุ่มของพริมิทีฟ ที่จะแสดงผลบนจอภาพ อย่างไรก็ตาม โปรแกรมที่ไม่ทำงานบนกราฟิก ก็อาจจะใช้คอร์เรลเลชันก็ได้ เช่น ในโปรแกรมที่ทมีการคำนวณ และอนุญาตให้ผู้ใช้สามารถเลือกการกระทำทางคณิตศาสตร์ได้

2.5.1.10 การตัดรูป และการตรวจขอบเขต (Clipping and Boundary Determination)

การตัดรูป คือกระบวนการที่แสดงผลลัพธ์ของการวาดกราฟิก เฉพาะที่อยู่ในขอบเขตที่กำหนดไว้เท่านั้น ส่วนกราฟิกที่อยู่นอกขอบ จะไม่มีการแสดงผล

โดยปกติแล้ว โปรแกรมจะทำการกำหนดขอบเขตเอง แต่บางครั้ง PM ก็จะกำหนดขอบเขต เช่น กำหนดขอบเขต เพื่อทำการวาดเฉพาะบนหน้าต่าง หรือเพื่อวาดเฉพาะที่ต้องแสดงผลใหม่เท่านั้น เป็นต้น

การตรวจขอบเขต เป็นการหาสี่เหลี่ยมมุมฉากที่เล็กที่สุด ที่ล้อมรอบกราฟิกที่กำหนด ทั้งนี้เพื่อหาตำแหน่งเท่าที่จำเป็นต่อการวาดหน้าจอใหม่ แทนที่จะวาดทั้งจอ เพื่อให้การวาดหน้าจอเร็วขึ้น

2.5.1.11 โคออร์ดิเนทสเปซ, การเปลี่ยนแปลงรูปแบบ และฟังก์ชัน (Coordinate Spaces and Transformation Operations and Functions)

การเปลี่ยนแปลงรูปแบบ ทำให้โปรแกรมสามารถควบคุมขนาด และตำแหน่งของกราฟิก บนอุปกรณ์แสดงผลต่างๆ นอกจากนั้น GPI ยังมีฟังก์ชันสำหรับย่อ/ขยาย, เปลี่ยนแปลง, หมุน และเอียง ทั้งบนกราฟิก และบน

ข้อความ ฟังก์ชัน GPI ส่วนใหญ่จะแสดงผลลัพธ์ไปยังพื้นที่เสมือนที่หนึ่ง ซึ่งเรียกว่า เวิร์ลด์โคออร์ดิเนตสเปซ (world coordinate space)

ตารางที่ 2.1 แสดงโคออร์ดิเนตสเปซทั้ง 5 ชนิดที่ใช้ในการเปลี่ยนแปลงใน GPI

ชื่อ	ความหมาย
World coordinate space	พิกัดที่โปรแกรมใช้สำหรับวาดกราฟิก
Model space	พิกัดเสมือน ที่ใช้สร้างรูปผ่านทางโมเดลทรานสฟอร์มเมตริก
Default page cooridnate space	พิกัดที่ใช้สำหรับแสดงรูปโดยไม่มีการย่อ/ขยาย
Page coordinate space	พิกัดเสมือน ที่ใช้สร้างรูปผ่านทางทรานสฟอร์มเมตริกปกติ
Device coordinate space	พิกัดของอุปกรณ์

2.5.1.12 การส่งงานไปพิมพ์ และการจัดการ (Print Job Submission and Manipulation)

การพิมพ์ คือการสร้างผลลพธ์บนแผ่นกระดาษ โปรแกรมสามารถส่งงานไปพิมพ์ผ่านทางสพูลเลอร์ (spooler) ซึ่งจะนำงานพิมพ์นั้นๆ ไปส่งยังเครื่องพิมพ์ตามกำหนด หรือโปรแกรมสามารถส่งงานไปพิมพ์ยังเครื่องพิมพ์โดยตรงก็ได้ แต่การส่งงานไปพิมพ์โดยตรงนั้น โปรแกรม (รวมทั้งผู้ใช้) จะต้องคอยจนกระทั่งงานนั้นๆ พิมพ์เสร็จ จึงจะทำงานต่อได้

ถ้าโปรแกรมส่งงานไปพิมพ์ผ่านทางสพูลเลอร์ โปรแกรมสามารถจัดการงานนั้นๆ ได้ในขณะที่งานนั้นๆ ยังอยู่ในคิวอยู่ เช่น สามารถสั่งให้พิมพ์มากกว่า 1 ชุด หรือสั่งให้ลบงานนั้นๆ ออกจากคิว เป็นต้น

2.5.2 GDI (Graphics Device Interface)

GDI นั้นเป็นส่วนโมดูลหนึ่งของวินโดวส์ที่ทำหน้าที่บริหารและจัดการคำสั่งทางกราฟิก

2.5.2.1 นิยามทั่วไป

ตัวโมดูล GDI นี้ เป็นไลบรารีที่ภายในประกอบไปด้วยฟังก์ชันสำหรับการเรียกใช้จำนวนมาก ซึ่งรองรับความสามารถในการวาดจุด เส้น สี่เหลี่ยม ส่วนโค้ง วงรี และวงกลม เป็นต้น นอกจากนั้น GDI ยังตอบสนองต่อการแปลคำสั่งทางกราฟิกเหล่านี้ แก่บรรดาไดรเวอร์แสดงผลทั้งหมด ประกอบไปด้วย หน้าจอ, เครื่องพิมพ์ และเครื่องวาด (plotter) ดังที่กล่าวมา GDI นั้นเป็นการติดต่อที่ไม่ขึ้นกับอุปกรณ์ (device-independent interface) ดังนั้นโปรแกรมกราฟิกที่เขียนขึ้น ตามมาตรฐานของวินโดวส์ จึงสามารถทำงานได้กับเครื่องพิมพ์ เครื่องวาดหรือการแสดงผลกราฟิกที่ติดตั้งไว้แบบใดๆ ก็ได้

2.5.2.2 จุดประสงค์ของ GDI

GDI จะตรวจสอบจากบรรดารายชื่อของไดรเวอร์ที่ติดตั้งไว้แล้วว่า จะติดต่อกับอุปกรณ์ฮาร์ดแวร์แต่ละชนิดที่เป็นส่วนประกอบของระบบคอมพิวเตอร์นั้นอย่างไร ตัวอย่างเช่น อุปกรณ์แบบราสเตอร์ (raster) ได้แก่ จอภาพโมโนโครม (monochrome) และจอสี นั้นจะต้องมีการจัดการควบคุมแตกต่างจากอุปกรณ์แบบเวคเตอร์ (vector) เช่นเครื่องวาด แม้แต่ระหว่างจอภาพแสดงผลด้วยกันก็ตาม ลักษณะคุณสมบัติความแตกต่างจะต้องนำมาพิจารณาด้วย เพราะว่าความละเอียด, สีและอัตราส่วนแสดงผล (aspect ratios) นั้นต่างกัน สภาพแวดล้อม ที่ไม่ขึ้นกับอุปกรณ์ของ GDI นี้ทำให้วินโดวส์มีความสามารถในการย้ายการพัฒนาโปรแกรม (portability) ที่ภาษาในการทำโปรแกรมหลายภาษาไม่สามารถทำได้ รวมทั้งความสามารถในการจัดการกับพิกเซล (pixel) ที่ทรงพลัง

2.5.2.3 ปฏิบัติการแบบพิกเซล

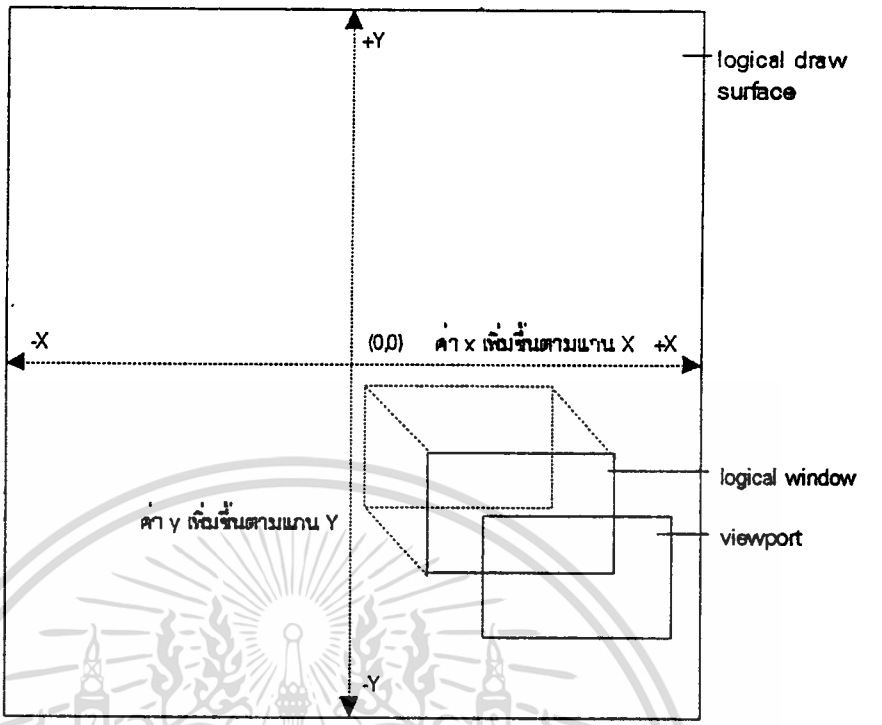
โดยปกติ GDI จะทำงานในโหมดพิกัดแบบพิกเซล (pixel coordinate mode) หรือโหมดข้อความ ซึ่งตัวมันจะดึงข้อมูลเกี่ยวกับไดรเวอร์อุปกรณ์ และปรับผลลัพธ์กราฟิกของมันเพื่อให้รูปร่างที่มีอัตราส่วนถูกต้อง และอยู่ในความละเอียดที่ถูกต้องด้วยสำหรับอุปกรณ์ฮาร์ดแวร์ นอกจากแบบนี้แล้วยังมีโหมดการแปลง (mapping mode) ที่เหลืออีก 7 แบบที่สามารถใช้ได้ในหน่วยมาตราเมตริก (metric), มาตราอังกฤษ และหน่วยที่ผู้ใช้กำหนดขึ้น

2.5.2.4 ข้อมูลอุปกรณ์

รายละเอียดที่เกี่ยวข้องกับอุปกรณ์ฮาร์ดแวร์ต่างๆ ที่ติดตั้งในระบบนั้น GDI สามารถดึงมาพิจารณาได้โดยการใช้ฟังก์ชัน GetDeviceCaps ซึ่งฟังก์ชันนี้จะส่งข้อมูลลักษณะเกี่ยวกับอุปกรณ์ฮาร์ดแวร์จำนวน 24 ค่า กลับมา ข้อมูลเหล่านี้ได้แก่ ความละเอียดในแนวตั้งและแนวนอน, อัตราส่วนภาพ, จำนวนเพลนสี (color plane) เป็นต้น นอกจากนี้ยังมีข้อมูลรายละเอียดซึ่งบ่งบอกความสามารถของอุปกรณ์ฮาร์ดแวร์นั้นว่าสนับสนุนคำสั่ง ในการวาดวงรี, วงกลม หรืออื่นๆ ได้โดยไม่ต้องอาศัยความช่วยเหลือจาก GDI ได้หรือไม่ ถ้าไม่มีความสามารถดังกล่าวนี้ วินโดวส์จะทำให้แทนโดยใช้ชุดคำสั่งทางซอฟต์แวร์ที่เหมือนกันผ่านทาง GDI

ตัวอย่างเช่น จอแสดงผลแบบไอบีเอ็มวีจีเอ (IBM VGA) มีขนาดในแนวนอน 240 มิลลิเมตรและขนาดในแนวตั้ง 180 มิลลิเมตร จะให้อัตราส่วนแนวนอนและแนวตั้งเป็น 36 หมายความว่าถ้าวาดจุดจำนวน 10 จุด ในทิศทางใดๆ ก็ตามจะได้เส้นความยาวเท่ากัน

ในโหมดการแปลงปกตินั้น GDI จะมีจุดเริ่มต้นที่มุมซ้ายด้านบนของวินโดว์ ส่วนค่าในแกนนอน (แกน X) มีค่าบวกเพิ่มขึ้นไปทางด้านขวา และค่าในแกนตั้ง (แกน Y) มีค่าบวกเพิ่มขึ้นไปในแนวดิ่งลงตามที่แสดงไว้ในรูปที่ 2.11



รูปที่ 2.11 ระบบพิกัดปกติในวินโดวส์

2.5.2.5 วินโดว์กับวิวพอร์ต (viewport)

ตัวแปรที่เกี่ยวข้องกับระบบพิกัดในวินโดว์คือ จุดกำเนิดหรือจุดเริ่มต้นของวินโดว์, คู่พิกัดในวินโดว์, จุดกำเนิดของวิวพอร์ต และคู่พิกัดในวิวพอร์ต ทั้งหมดจะมีความสัมพันธ์กันกับโหมดของการแปลงอย่างมาก

จากรูปที่ 2.11 ด้านหลังสุดเป็นพื้นผิวสำหรับวาดในเชิงลอจิค (logical draw surface) ที่ถูกอ้างจากโปรแกรม ซึ่งเป็นระนาบที่ใช้ในการวาดลงไป ระนาบนี้มีขนาด 64x64 กิโลหน่วยลอจิค โดยจุดศูนย์กลางของระนาบเป็นจุดเริ่มต้นของแกน XY ที่ (0,0) แผ่นเหนือขึ้นมาเป็นลอจิคัลวินโดว์ (logical window) ซึ่งมี หน่วยนับลอจิคเดียวกันกับระนาบสำหรับวาด เมื่อมีการเคลื่อนย้ายวินโดว์ด้วยคำสั่งเปลี่ยนตำแหน่งจุดเริ่มต้น ก็จะใช้หน่วยเดียวกันกับระนาบสำหรับวาด ส่วนขอบเขต หรือความกว้างคูณความยาวของวินโดว์ ที่จะครอบคลุมแผ่นระนาบนั้นจะขึ้นอยู่กับอัตราการแปลงระหว่างหน่วยลอจิค ไปเป็นหน่วยทางกายภาพ ซึ่งอัตราการแปลงนี้ ขึ้นอยู่กับในแต่ละโหมดของการแปลง เช่น ถ้าเป็น 2:1 จะทำให้เห็นภาพระนาบได้กว้างขึ้น แสดงว่าภาพถูกย่อให้เล็กลง ในทางกลับกัน ถ้าเปลี่ยนเป็นอัตรา 1:2 จะทำให้เห็นภาพที่ระนามได้น้อยลง แสดงว่าภาพถูกขยายขึ้น ขณะที่วินโดว์ ยังมีกรอบขนาดเท่าเดิม

ส่วนบนสุดคือวิวพอร์ต ซึ่งเป็นกรอบสี่เหลี่ยมเช่นเดียวกับวินโดว์ แต่หน่วยนับจะต่างกัน หน่วยนับของ วิวพอร์ตจะเป็นจุด เวลาสั่งเคลื่อนย้ายกรอบ จะเปลี่ยนตำแหน่งไปตามจุดที่กำหนดโดยแกนค่าบวก XY ของ กรอบวิวพอร์ตทีอยู่ทางด้านซ้ายและบนตามลำดับ ซึ่งแตกต่างไปจากการเคลื่อนย้ายวินโดว์ ในกรณีของวินโดว์นั้น จะเป็นไปตามหน่วยของอัตราที่เปรียบเทียบ โดยปกติเมื่อเริ่มต้นวิวพอร์ตจะอยู่ซ้อนทับบนวินโดว์พอดี เมื่อสั่งให้เคลื่อนย้ายวินโดว์เมื่อไร วิวพอร์ตก็จะย้ายตามไปด้วย เพราะว่าวิวพอร์ตวินโดว์เป็นจุดอ้างอิงเสมอ

2.5.2.6 โหมดการแปลง

ทุกฟังก์ชันด้านกราฟิกของ GDI นั้นมีความสัมพันธ์ขึ้นกับโหมดการแปลงที่เลือก ดังที่กล่าวมาแล้วสำหรับโหมดปกตินั้นคือ MM_TEXT แต่จะจุดจะแทนในหน่วยพิกเซลหนึ่งพิกเซล การแปลงนั้นกระทำจากหน่วยในจินตนาการไปหน่วยความเป็นจริงทางกายภาพ สำหรับในโหมด MM_TEXT การแปลงจำทำในอัตราส่วน 1:1 ทำให้รูปร่างที่ได้จากการวาดนั้น ขึ้นอยู่กับอุปกรณ์แสดงผล เช่น ถ้าวาดรูปวงกลม บนหน้าจอที่มีจุดภาพเป็นรูปสี่เหลี่ยมผืนผ้า รูปวงกลมจะกลายเป็นรูปวงรี แต่ถ้าจุดภาพของหน้าจอเป็นสี่เหลี่ยมจัตุรัสก็จะได้รูปวงกลม

ในโหมด MM_LOMETRIC, MM_HIMETRIC, MM_LOENGLISH, MM_HIENGLISH และ MM_TWIPS จะให้รูปทรงที่เหมือนจริงคือจะไม่ขึ้นกับอุปกรณ์ที่แสดงผล โดยสองแบบแรกมีหน่วยเป็นมิลลิเมตร และสามแบบถัดมามีหน่วยเป็นนิ้ว ส่วน MM_ISOTROPIC นั้น อนุญาตให้ผู้ใช้สามารถกำหนดอัตราการแปลง จากหน่วยลอจิคเป็นหน่วยกายภาพขึ้นเองได้ และสุดท้ายคือ MM_ANISOTROPIC นั้นก็เช่นเดียวกันกับ MM_ISOTROPIC แต่อัตราการแปลงในแต่ละแกนนั้นเป็นอิสระต่อกัน ไม่จำเป็นต้องเท่ากัน

การเปลี่ยนโหมดการแปลงสามารถทำได้โดยการเรียกใช้ฟังก์ชัน SetMapMode พร้อมกับระบุค่าของโหมดต่างๆ ที่วินโดวส์ได้จัดเตรียมไว้แล้วตามตารางที่ 2.2

ตารางที่ 2.2 โหมดการแปลงแบบต่างๆ ในวินโดวส์

แบบการแปลง	หน่วยลอจิค	ค่าบวกแกน X	ค่าบวกแกน Y
MM_TEXT	1 พิกเซล	ด้านขวา	ด้านล่าง
MM_LOMETRIC	0.1 มม.	ด้านขวา	ด้านบน
MM_HIMETRIC	0.01 มม.	ด้านขวา	ด้านบน
MM_LOENGLISH	0.01 นิ้ว	ด้านขวา	ด้านบน
MM_HIENGLISH	0.001 นิ้ว	ด้านขวา	ด้านบน
MM_TWIPS	1/1440 นิ้ว	ด้านขวา	ด้านบน
MM_ISOTROPIC	กำหนดเอง x เท่ากับ y	เลือกได้	เลือกได้
MM_ANISOTROPIC	กำหนดเอง x ไม่เท่ากับ y ได้	เลือกได้	เลือกได้

2.5.2.7 คอนเท็กซ์ของอุปกรณ์ (DC, Device Context)

คอนเท็กซ์ของอุปกรณ์หรือ DC นั้นเป็นตัวกลางในการติดต่อระหว่างแอพพลิเคชันกับอุปกรณ์แสดงผลกราฟิกโดยมีวินโดวส์เป็นตัวกลางอีกทีหนึ่ง นั่นคือแอพพลิเคชันจะติดต่อกับอุปกรณ์ผ่านทางวินโดวส์ โดยใช้ คอนเท็กซ์ของอุปกรณ์เหมือนกับเป็นสตรีม (stream) มาตรฐานที่ใช้ในภาษาซี เช่น การนำเข้า และแสดงออกทางหน้าจอ

โดยใช้ฟังก์ชัน fprintf โดยผ่านสตรีม stdio หรือผ่านไปยังเครื่องพิมพ์โดยใช้สตรีมชื่อ stdprn เช่นเดียวกับ ในวินโดวส์ คอนเท็กของอุปกรณ์ เป็นข้อกำหนดหรือลักษณะการติดต่ออย่างกว้าง ๆ ที่วินโดวส์มีต่อแอพพลิเคชัน

2.5.2.8 คอนเท็กซ์แสดงผล (Display Context)

การจัดการกับการแสดงผลของวินโดวส์นั้น ใช้สิ่งที่เรียกว่า คอนเท็กซ์แสดงผล ซึ่งเป็นชนิดหนึ่งของ คอนเท็กซ์อุปกรณ์ ตัวคอนเท็กซ์แสดงผลนี้เองที่ทำให้มองแต่ละวินโดว์ได้ว่าอยู่ที่พื้นผิวหรือพื้นทีที่ต่างกัน แอพพลิเคชันที่ได้คอนเท็กซ์แสดงผลนี้ ก็สามารถจัดการเกี่ยวกับวินโดว์นั้นได้ทุกประการ โดยฟังก์ชันที่เกี่ยวกับเอาต์พุตไปสู่วินโดว์นั้นอยู่ภายใต้การควบคุม หรือเรียกใช้งานผ่าน GDI อีกทีหนึ่ง

สำหรับชนิดของคอนเท็กซ์แสดงผลมีทั้งหมด 4 ชนิดด้วยกัน คือแบบทั่วไป (common), แบบคลาส (class), แบบส่วนตัว (private) และแบบวินโดว์ ซึ่งสามแบบแรกนั้นยินยอมให้มีการวาดลงในพื้นที่ใช้งานของวินโดว์เท่านั้น แต่ประเภทสุดท้ายสามารถวาดลงไปที่ใดของวินโดว์ก็ได้ การกำหนดว่าวินโดว์ใดจะได้คอนเท็กซ์แสดงผลชนิดใดไปนั้น วินโดวส์กำหนดให้โดยดูจากคลาสของวินโดว์นั้น

2.5.2.8.1 คอนเท็กซ์แสดงผลแบบทั่วไป

คอนเท็กซ์แสดงผลชนิดนี้จะถูกกำหนดให้กับวินโดว์ ที่ใช้สไตล์ที่ไม่มีการระบุชนิดของคอนเท็กซ์แสดงผล ซึ่งเมื่อวินโดว์มีคอนเท็กซ์แสดงผลชนิดนี้แล้วก็สามารถที่จะวาดลงไปในพื้นที่ใช้งานของวินโดว์ตัวเองได้

2.5.2.8.2 คอนเท็กซ์แสดงผลแบบคลาส

คอนเท็กซ์แสดงผลชนิดนี้เป็นคอนเท็กซ์ส่วนรวมระหว่างวินโดว์ที่ใช้คลาสเดียวกัน โดยอุปกรณ์ต่างๆ ที่วินโดว์ตั้งให้คอนเท็กซ์แสดงผลแบบนี้ จะตกทอดไปถึงวินโดว์ต่อไปที่ร้องขอใช้คอนเท็กซ์แสดงผลด้วย การที่จะให้คลาสใช้คอนเท็กซ์แสดงผลแบบนี้ สามารถทำได้โดยการกำหนดรูปแบบ CS_CLASSDC ให้ตอนขึ้นทะเบียนคลาส

2.5.2.8.3 คอนเท็กซ์แสดงผลแบบส่วนตัว

การสร้างคอนเท็กซ์แสดงผลแบบส่วนตัวทำได้โดยกำหนด CS_OWNDC ในตอนสร้างวินโดว์ ทำให้วินโดว์นั้นมีคอนเท็กซ์แสดงผลใช้ส่วนตัว ไม่ปะปนกับคอนเท็กซ์ส่วนรวม เมื่อใช้คอนเท็กซ์แสดงผลแบบนี้แล้วมีการร้องขอเพียงครั้งแรกครั้งเดียวเท่านั้น และสามารถใช้ไปได้ตลอด ไม่สามารถยกเลิกได้

คอนเท็กซ์แสดงผลชนิดนี้ใช้งานง่ายและสะดวก เพราะไม่ต้องมีการร้องขอทุกครั้งที่ต้องการวาดภาพ แต่มีข้อเสียคือ ทำให้เปลืองหน่วยความจำในการที่ต้องเก็บข้อมูลของวินโดว์นั้นๆ เพียงวินโดว์เดียวตลอดเวลา แม้ว่าวินโดว์นั้นไม่ได้อยู่ในกระบวนการวาดภาพอยู่

2.5.2.8.4 คอนเท็กซ์แสดงผลแบบวินโดว์

สำหรับแบบนี้ แอพพลิเคชันสามารถวาดที่ตำแหน่งใดๆ บนวินโดว์ก็ได้ ไม่จำกัดเฉพาะในพื้นที่ใช้งานเหมือน 3 แบบแรก จุดอ้างอิงของคอนเท็กซ์แสดงผลแบบนี้จะอยู่ที่มุมบนด้านซ้ายสุดของวินโดว์

2.5.2.9 แคช (cache) จองคอนเท็กซ์แสดงผล

แคชของคอนเท็กซ์แสดงผลคือ หน่วยความจำส่วนที่ใช้เก็บคอนเท็กซ์แสดงผลแบบทั่วไป ซึ่งสามารถเก็บได้ทั้งหมด 5 ชุด หมายความว่าสามารถมีวินโดว์ที่ใช้คอนเท็กซ์แสดงผลแบบทั่วไปได้พร้อมๆ กันเป็นจำนวน จำกัดอยู่ 5 วินโดว์ ดังนั้นวินโดว์ที่ใช้คอนเท็กซ์แสดงผลแบบทั่วไปหรือแบบวินโดว์จะต้องปล่อยคอนเท็กซ์แสดงผล ทันทีที่ไม่ใช้แล้ว

หากมีการร้องขอใช้คอนเท็กซ์แสดงผลในขณะที่ถูกใช้ไปหมดแล้วทั้ง 5 ชุด จะทำให้เกิดอาการติดตายขึ้น มีข้อความเตือนผู้ใช้ และหลุดออกจากการทำงาน สำหรับคอนเท็กซ์แสดงผลแบบส่วนตัวและแบบคลาสนั้น มีหน่วยความจำต่างหากในการเก็บไม่เกี่ยวกับส่วนแคชนี้

2.5.2.10 เมสเสจ WM_PAINT

เมื่อมีการเปลี่ยนแปลงเกิดขึ้นกับวินโดว์ ไม่ว่าจะเป็นการย้าย, การเปลี่ยนขนาด หรือถูกวินโดว์อื่นทับ วินโดวส์จะบันทึกตำแหน่งที่ต้องมีการวาดใหม่เหล่านั้นเอาไว้ และเมื่อมีโอกาสก็จะส่ง WM_PAINT มา เพื่อให้แอพพลิเคชันทำการวาดตรงพื้นที่ที่บันทึกไว้นี้ใหม่ (โดยใช้ฟังก์ชัน BeginPaint และ EndPaint)

ฟังก์ชันที่เกี่ยวข้องกับการวาดใหม่นี้มีสองฟังก์ชันคือ InvalidateRect และ InvalidateRgn ซึ่งจะใช้ในการบันทึกพื้นที่ที่ผู้ใช้ต้องการวาดใหม่ โดยสามารถบันทึกไว้ได้หลายๆ แห่งพร้อมกัน เมื่อฟังก์ชันประจำวินโดว์ ได้รับ WM_PAINT ก็จะทำการวาดส่วนต่างๆ เหล่านั้น (ทั้งที่ผู้ใช้เป็นผู้กำหนด และที่วินโดวส์กำหนดเอง) ใหม่ในทีเดียว การทำเช่นนี้ทำให้วินโดวส์ไม่ต้องวาดใหม่ทั้งพื้นที่ใช้งาน ซึ่งสามารถประหยัดเวลาในการปรับ เปลี่ยนหน้าจอได้มาก

การโมฆะบางพื้นที่ในส่วนของพื้นที่ที่ต้องถูกวาดใหม่ สามารถทำได้โดยใช้ฟังก์ชัน ValidateRect และ ValidateRgn ส่วนอีกฟังก์ชันที่สามารถสร้างเมสเสจ WM_PAINT และส่งไปยังวินโดว์ต่างๆ ได้ทันทีคือฟังก์ชัน UpdateWindows ซึ่งการส่ง WM_PAINT ของฟังก์ชันนี้จะส่งโดยตรงไปยังฟังก์ชันประจำวินโดว์ ทำให้ไม่ต้องเข้าแถวต่อจากเมสเสจในคิวอื่นๆ ส่วนมากจะใช้ฟังก์ชันนี้กับวินโดว์ที่เพิ่งเริ่มสร้าง

2.5.2.11 การขอคอนเท็กซ์ของอุปกรณ์

ทุกๆ ฟังก์ชันกราฟิกของ GDI นั้นต้องการแฮนเดิล (handle) ไปยังคอนเท็กซ์ของอุปกรณ์ทั้งสิ้น จึงต้องมีการขอคอนเท็กซ์ของอุปกรณ์ก่อนการวาดเสมอ การของคอนเท็กซ์ของอุปกรณ์สำหรับการวาดรูปนั้น มีด้วยกันหลายวิธี อุปกรณ์ต่อพ่วงที่ติดตั้งไว้แต่ละชนิดก็จะมีคอนเท็กซ์เป็นของตัวเอง ตัวอย่างเช่น จอภาพ, วินโดว์, เครื่องพิมพ์ สำหรับอุปกรณ์ที่ใช้มากที่สุดคือวินโดว์ นั้นหมายถึงส่วนที่เป็นสี่เหลี่ยมทั้งหมด ซึ่งประกอบไปด้วย กรอบ, พื้นที่แสดงผล และไตเติ้ลบาร์ พื้นที่ทั้งหมดนี้สามารถวาดลงไปได้ แต่โดยทั่วไปนั้นส่วนกรอบ และไตเติ้ลบาร์จะปล่อยให้วินโดวส์เป็นตัวจัดการ ส่วนที่จะจัดการจึงทำเฉพาะพื้นที่แสดงผลเท่านั้น

2.5.2.11.1 การขอคอนเท็กซ์ในเมสเสจ WM_PAINT

การจัดการพื้นที่แสดงผล เริ่มต้นด้วยการขอคอนเท็กซ์ของอุปกรณ์ที่เมสเสจ WM_PAINT ด้วยฟังก์ชัน BeginPaint ซึ่งเป็นวิธีที่มีประสิทธิภาพมากที่สุด เพราะว่าวินโดวส์จะส่งเฉพาะพื้นที่ ที่ไม่สมบูรณ์มาให้วาดพื้นที่ดังกล่าวนี้ได้แก่ พื้นที่ว่างที่เกิดจากผู้ใช้เคลื่อนย้ายวินโดว์มาทับกันแล้วเคลื่อนย้ายออก หรือเกิดจากระบบ อันเนื่องจากมีการย้าย, เคลื่อน, สร้างวินโดว์ใหม่ การขอคอนเท็กซ์ของอุปกรณ์ด้วยวิธีนี้จะทำการวาดเฉพาะส่วนที่ ขาดหายไปเท่านั้น ซึ่งทำให้วินโดวส์ทำการวาดหน้าจอได้เร็วกว่าการวาดใหม่ทั้งหมด เมื่อจอขั้นตอนของการวาดแล้ว จะต้องปล่อยคอนเท็กซ์ให้เป็นอิสระด้วยฟังก์ชัน EndPaint

2.5.2.11.2 การขอคอนเท็กซ์ในช่วงที่ไม่มีเมสเสจ WM_PAINT

การขอคอนเท็กซ์ด้วย BeginPaint ควรจะทำที่เมสเสจ WM_PAINT เท่านั้น สำหรับในช่วงเวลาที่ไม่มีเมสเสจ WM_PAINT นั้นสามารถทำการวาดในช่วงเวลานี้ได้ด้วยการขอคอนเท็กซ์ของอุปกรณ์ด้วยฟังก์ชัน GetDC และจบด้วย ReleaseDC ฟังก์ชันทั้งคู่จะได้คอนเท็กซ์ของอุปกรณ์สำหรับวาดบนพื้นที่ทำงานทั้งหมด การวาดใดๆ ที่เกิดขึ้น ถือว่าเป็นการวาดใหม่ทั้งวินโดว์

2.5.2.11.3 การขอคอนเท็กซ์สำหรับพื้นที่นอกเหนือการแสดงผล

ถ้าหากต้องการวาดในส่วนนอกเหนือจากพื้นที่แสดงผล สามารถทำได้ด้วยการขอคอนเท็กซ์ในช่วงเมสเสจ WM_NCPAINT ด้วยฟังก์ชัน GetWindowDC และจบด้วยฟังก์ชัน ReleaseDC ฟังก์ชันคู่นี้จะได้คอนเท็กซ์ของอุปกรณ์สำหรับวาดบนเนื้อที่วินโดว์ทั้งหมด

2.5.2.11.4 การขอคอนเท็กซ์สำหรับอุปกรณ์กราฟิกชนิดอื่น ๆ

สำหรับอุปกรณ์กราฟิกชนิดอื่นๆ ที่ไม่ใช่วินโดว์ สามารถขอคอนเท็กซ์สำหรับวาดลงไปได้ในลักษณะเดียวกับวินโดว์ โดยใช้ฟังก์ชัน CreateDC และจบด้วย DeleteDC โดยระบบพารามิเตอร์ต่าง ๆ ที่เกี่ยวข้องกับอุปกรณ์นั้น ๆ เช่น ถ้าเป็นเครื่องพิมพ์ จะต้องระบุชื่อไดรเวอร์ ชื่อเครื่องพิมพ์ และพอร์ตที่ต่ออยู่เป็นต้น

2.5.2.12 การขอคอนเท็กซ์แบบหน่วยความจำ

นอกเหนือจากอุปกรณ์แล้ว ยังสามารถเขียนกราฟิกลงบนหน่วยความจำแทนอุปกรณ์ได้ด้วย การขอคอนเท็กซ์สำหรับหน่วยความจำนั้นใช้ฟังก์ชัน CreateCompatibleDC จบด้วย DeleteDC ส่วนจะให้หน่วยความจำเข้ากันได้กับอุปกรณ์ชนิดใด ให้ใส่คอนเท็กซ์ของอุปกรณ์นั้นลงในอาร์กิวเมนต์ (argument) ของฟังก์ชันนี้

2.5.2.13 ลักษณะของงานในคอนเท็กซ์อุปกรณ์

ลักษณะของงานที่กำหนดไว้ในคอนเท็กซ์ของอุปกรณ์มีอยู่ 3 ส่วนคือ วัตถุที่ใช้ในการวาด, ลักษณะของการวาด และระบบพิกัดจุด

2.5.2.13.1 วัตถุที่ใช้วาด

วัตถุที่ใช้วาดประกอบด้วย ปากกา (pen), แปรง (brush), แบบอักษร (font) และรูปภาพ (bitmap) วัตถุสามชนิดแรกนั้นเมื่อมีการขอคอนเท็กซ์อุปกรณ์ วินโดวส์จะให้มาอย่างละ 1 ได้แก่เป็นค่าดีฟอลต์ (default) ได้แก่ แปรงสีขาว (WHITE_BRUSH), ปากกาสีดำ (BLACK_PEN), และแบบอักษรของระบบ (SYSTEM_FONT) ซึ่งสามารถนำไปใช้ได้ทันที เช่น นำไปวาดรูปสี่เหลี่ยมด้วยฟังก์ชัน Rectangle ซึ่งจะเกิดรูปสี่เหลี่ยมที่เกิดจากการใช้ปากกา และแปรงสีผสมกัน ขอบนอกเป็นเส้นสีดำ และภายในระบายด้วยสีขาว, การลากเส้นตรงด้วยฟังก์ชัน Line จะมีสีดำเนื่องจากใช้ปากกาสีดำไปวาด หรือเมื่อทำการพิมพ์อักษรด้วยฟังก์ชัน TextOut จะเกิดอักษรสีดำที่มีรูปแบบเหมือนกับฟอนต์ของระบบ เป็นต้น

วัตถุที่ใช้วาดทั้ง 3 ชนิดนี้ สามารถเปลี่ยนได้ด้วยการใช้ฟังก์ชัน SelectObject ซึ่งจะนำวัตถุไปแทนกับวัตถุที่ใช้อยู่ในคอนเท็กซ์อุปกรณ์ปัจจุบัน พร้อมกับคืนค่าแฮนเดิลของวัตถุเดิมมาให้ วัตถุใหม่ที่เลือกไปแทนนี้อาจจะได้มาจากคลังเก็บวัตถุ (stock) ที่วินโดวส์เก็บไว้สำหรับวัตถุที่มีการเรียกใช้บ่อยๆ ส่วนอีกวิธีหนึ่งคือสร้างขึ้นใหม่

การสร้างวัตถุขึ้นมาใหม่นั้น จะใช้หน่วยความจำของระบบไปส่วนหนึ่ง ดังนั้นเวลาเลิกใช้จึงต้องขจัดวัตถุหรือลบออกด้วยการเรียกใช้ฟังก์ชัน DeleteObject ถ้าไม่ทำเช่นนี้แล้วจะทำให้วัตถุนี้ค้างอยู่ จนกว่าจะออกจากวินโดวส์ ซึ่งมีความสำคัญอย่างยิ่งในการที่แอพพลิเคชันย้อนกลับสร้างวัตถุใหม่พอกพูนตลอดเวลา เช่น การสร้างวัตถุในรอบการทำงานของเมสเสจ WM_PAINT ซึ่งทำให้ทรัพยากรหน่วยความจำลดลงเรื่อยๆ จนไม่เพียงพอ. โนการใช้งานได้

2.5.2.13.2 ลักษณะของการวาด

การวาดรูปตามปกติที่ได้มาจากการขอคอนเท็กซ์ของอุปกรณ์คือ จะทำการลบฉากหลังสีพื้นตามขนาดของรูปที่จะวาด จากนั้นจะทำการวาดรูปทับลงไปด้วยคุณสมบัติของวัตถุที่ใช้วาดชนิดต่างๆ

ที่กล่าวมาเป็นลักษณะของการวาดที่ควบคุมด้วยสีของพื้นหลัง (Background color) และโหมดพื้นหลัง (Background mode) นอกจากนี้แล้วยังมีการควบคุมลักษณะของการวาดอีก 3 ตัว คือ โหมดของการวาดตัวนี้ควบคุมการระบายสีของปากกาให้มีรูปแบบที่หลากหลาย เช่น การวาดกลับสี, การผสมของสีพื้น เป็นต้น โหมดการลงสีรูปหลายเหลี่ยม มีไว้สำหรับควบคุมการวาดด้วยฟังก์ชันที่ขึ้นต้นด้วย Poly และสุดท้ายคือ บริเวณตัดออก (Clipping region) เป็นตัวกำหนดอาณาเขตที่ให้วาดได้ ผลของการวาดจะไม่กระทบกับส่วนที่ไม่ได้กำหนด

2.5.2.13.3 ระบบพิกัดจุด

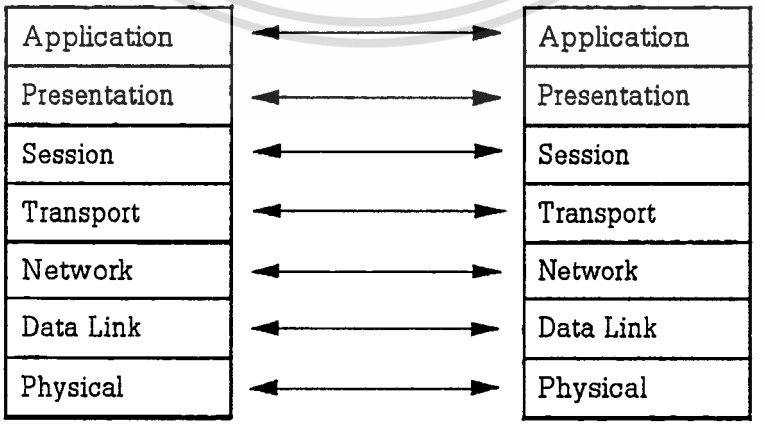
ระบบพิกัดจุดเป็นองค์ประกอบที่สำคัญอย่างหนึ่งที่เกี่ยวข้องกับการวาด นอกเหนือจากวัตถุและฟังก์ชันที่ใช้ในการวาดลงไปในคอนเท็กซ์อุปกรณ์แล้ว ระบบพิกัดจุดจะเป็นตัวควบคุมขนาดทิศทางสำหรับการนำผลลัพธ์ออกมาปรากฏบนอุปกรณ์ สามารถทำการย่อขยาย การกลับทิศทางบนล่าง โดยไม่ต้องเปลี่ยนกรรมวิธีการวาด

สิ่งที่เกี่ยวข้องกับขนาดคือการแปลงจากหน่วยในจินตนาการ หรือหน่วยลอจิค มาเป็นหน่วยในความเป็นจริง เช่นเป็นพิกเซล, นิ้ว, มิลลิเมตร หรือตามที่ผู้ใช้กำหนด และการกลับทิศทางนั้นจะเกี่ยวข้องกับการกำหนดจุดกำเนิด และคู่พิกัดที่ใช้

2.6 การติดต่อสื่อสาร

2.6.1 ระดับชั้น OSI

ระดับชั้นต่างๆ ตามแบบจำลอง OSI มีทั้งหมด 7 ชั้นได้แก่



รูปที่ 2.12 ระดับชั้นตามแบบจำลอง OSI

2.6.1.1 ระดับชั้นกายภาพ (Physical Layer)

เป็นชั้นที่เกี่ยวกับการติดต่อสื่อสารในระดับเบื้องต้น เช่น ใช้ระดับแรงดันไฟฟ้าเท่าไรแทนบิต 0 และ 1 ช่วงเวลาในการส่งบิตที่ติดต่อกัน ควรอยู่ห่างกันน้อยที่สุดเท่าไร การส่งสัญญาณในช่องสัญญาณควรส่งแบบใด การเริ่มต้นในการติดต่อและสิ้นสุดการติดต่อควรทำอย่างไร จำนวนเข็มและหน้าที่ของเข็มในการเชื่อมต่อ ซึ่งจากคุณสมบัติการติดต่อสื่อสารที่กล่าวมาจะเกี่ยวข้องกับคุณสมบัติทางไฟฟ้าและคุณสมบัติทางกลเป็นหลัก

2.6.1.2 ระดับชั้นเชื่อมต่อข้อมูล (Data Link Layer)

หน้าที่หลักของชั้นนี้คือการตรวจที่ผิดและแก้ไข (Error Detection and Error Correction) เมื่อระดับชั้นนี้ได้รับข้อมูล ที่ส่งผ่านมาจากระดับเครือข่ายที่อยู่เหนือระดับชั้นนี้แล้ว จะทำการแบ่งข้อมูลออกมาเป็นเฟรมๆ โดยที่แต่ละเฟรม ประกอบด้วยจำนวนข้อมูลคงที่จำนวนหนึ่ง เมื่อด้านรับได้รับข้อมูลแล้ว จะส่งการตอบสนอง (acknowledgement) กลับมายังด้านส่ง เนื่องจากการส่งข้อมูลในลักษณะส่งไปทีละบิต โดยไม่สนใจว่าบิตดังกล่าวจะประกอบกันขึ้นเป็นเฟรมหรือไม่ จึงเป็นหน้าที่ของระดับชั้นนี้ ที่จะทำการแทรกบิตพิเศษไว้ต้นและท้ายของเฟรม เพื่อใช้ในการตรวจสอบจุดเริ่มต้นและจุดสิ้นสุดของเฟรม

2.6.1.3 ระดับชั้นเครือข่าย (Network Layer)

มีหน้าที่หลักควบคุมการทำงานของเครือข่ายย่อย (Sub Net) ซึ่งจุดใหญ่ในการออกแบบคือ การที่จะกำหนดว่าแพ็กเกต (Packet) ใดจะใช้เส้นทางใดในการส่งจากด้านส่งไปยังด้านรับ โดยการจัดหาเส้นทางนี้อาจถูกกำหนดออกมาแบบตายตัวในลักษณะทางสถิติ และระหว่าง IMP (Interface Message Processors) ตัวหนึ่งกับ IMP อีกตัวหนึ่ง โดยก่อนจะถึงปลายทางจะต้องผ่าน IMP ตัวใดบ้างหรืออาจจะทำเป็นลักษณะไดนามิก (Dynamic) ซึ่งจะทำให้การติดต่อระหว่าง IMP คู่หนึ่งแพ็กเกตของข้อมูลจะถูกส่งไปในเส้นทางที่แตกต่างกัน เพื่อลดปัญหาในเรื่องภาระของเส้นทางในระบบเครือข่าย สำหรับเครือข่ายแบบบอร์ดคาสท์ (Board Cast) นั้น ปัญหาเรื่องการจัดเส้นทางจะไม่มี เนื่องจากเส้นทางในการส่งผ่านนั้นมีเพียงเส้นทางเดียว ระบบเครือข่ายแบบนี้อาจไม่มีระดับชั้นเครือข่ายเลยก็ได้

2.6.1.4 ระดับชั้นส่งต่อ (Transport Layer)

มีหน้าที่หลักคือ การรับข้อมูลจากระดับชั้นการติดต่อ (Session Layer) จากนั้นจะทำการแยกข้อมูลออกเป็นหน่วยต่างๆ ต่อไปยังระดับชั้นเครือข่าย ซึ่งจะต้องทำการอันจะเป็นที่แน่ใจว่าหน่วยต่างๆ เหล่านี้ต่อลงไปยังจุดหมายปลายทางอย่างถูกต้องตามลำดับ ในสภาวะการปกติระดับชั้นนี้จะทำหน้าที่คอยสร้างเส้นทางในการติดต่อสื่อสารตามที่ระดับชั้นการติดต่อร้องขอมาเมื่อต้องการที่จะส่งข้อมูล และเมื่อระดับชั้นนี้ต้องการให้การไหลของข้อมูลเป็นไปได้อย่างรวดเร็วอาจจะสร้างเส้นทางขึ้นมาหลายๆ เส้นทาง โดยแต่ละหน่วยแยกกันไปคนละเส้นทาง และจะไปเส้นทางไหนขึ้นกับระดับชั้นเครือข่ายที่คอยจัดการ แต่ถ้าการสร้างเส้นทางขึ้นมาหลายๆ เส้นทางนั้นเป็นไปได้ยาก

หรือไม่คุ้มค่า ระดับชั้นเชื่อมต่อข้อมูลก็อาจจะทำการสร้างเส้นทางขึ้นมาเพียงเส้นทางเดียว และทำการส่งข้อมูลโนลักษณะของมัลติเพลกซ์ เช่น TDM (Time Division Multiplex), FDM (Frequency Devision Multiplex) เป็นต้น

2.6.1.5 ระดับชั้นการติดต่อ (Session Layer)

ระดับชั้นนี้ทำหน้าที่ในการสร้างการติดต่อ ระหว่างผู้ใช้บริการที่อยู่คนละเครื่องอุปกรณ์กัน ให้สามารถติดต่อสื่อสารกันได้ ความหมายของการติดต่อในที่นี้ก็คือ การที่ยอมให้มีการส่งข้อมูลเป็นลำดับตามมา ตัวอย่างเช่น ในระบบโทรศัพท์ หน้าที่ในการบริการต่อผู้เรียกโดยส่งสัญญาณไปยังผู้รับ ผู้รับรับโทรศัพท์ขึ้นมาและทำให้การสนทนาเริ่มต้นขึ้น หน้าที่ดังกล่าวนี้เป็นของระดับชั้นนี้ แต่หน้าที่ในการที่จะส่งผ่านสัญญาณแต่ละคำพูดจากฝ่ายหนึ่งไปยังอีกฝ่ายหนึ่งนั้นเป็นหน้าที่ของระดับชั้นส่งต่อ

บริการอีกอย่างหนึ่งที่มีความเกี่ยวเนื่องกับระดับชั้นนี้ก็คือ การที่จะไม่อนุญาตให้เครื่องอุปกรณ์ทั้งสองด้านที่ติดต่อสื่อสารกันอยู่นั้น ทำสิ่งที่เหมือนกันในเวลาเดียวกันได้ นอกจากนี้หน้าที่อีกอันหนึ่งคือ การทำซิงค์โครไนซ์ (Synchronize) เครื่องอุปกรณ์ทั้งสองตัวเข้าด้วยกัน เพื่อให้การติดต่อรับส่งข้อมูลนั้นเป็นไปอย่างคล้องจองกัน

2.6.1.6 ระดับชั้นการนำเสนอ (Presentation Layer)

มีหน้าที่เกี่ยวกับ รูปแบบต่างๆ ของข่าวสารที่ถูกส่งจากเครื่องอุปกรณ์ออกไป เช่น การเข้ารหัสข้อมูลให้อยู่ในรูปที่สามารถเข้าใจกันทั้งผู้รับและผู้ส่ง ตัวอย่างเช่น เมื่อระดับชั้นนี้ได้รับข้อความจากระดับชั้นแอพพลิเคชัน (Application Layer) ซึ่งเป็นตัวอักษรแล้ว ก็จะต้องมาทำการเข้ารหัสอักขระแต่ละตัวให้อยู่ในรูปแบบของ ASCII หรือ EBCDIC เป็นต้น จุดประสงค์ใหญ่ของหน้าที่นี้ก็คือเพื่อที่จะทำให้เครื่องอุปกรณ์ต่างยี่ห้อ และใช้รหัสที่ต่างกันสามารถส่งผ่านข้อมูลกันได้ นอกเหนือจากนี้แล้วยังมีหน้าที่อย่างอื่น เช่น การบีบอัดข้อมูลให้มีขนาดเล็กลง เพื่อจำนวนข้อมูลที่จะส่ง

2.6.1.7 ระดับชั้นแอพพลิเคชัน (Application Layer)

ระดับชั้นนี้มีโปรโตคอล (Protocol) มากมายหลายแบบ เนื่องจากระดับชั้นนี้เป็นส่วนที่ติดต่อกับผู้ใช้บริการ สมมติว่าในเครือข่ายมีเทอร์มินัล (Terminal) มากมายหลายแบบ และมีซอฟต์แวร์อยู่ตัวหนึ่งที่คอยทำหน้าที่เป็นอิดิเตอร์ (Editor) ที่เทอร์มินัล ในลักษณะเช่นนี้ก็จะเกิดปัญหาขึ้นทางด้านการแสดงผลต่างๆ ที่เทอร์มินัล เนื่องจากเป็นเทอร์มินัลคนละชนิดกัน วิธีหนึ่งที่จะแก้ปัญหานี้คือ การนิยามสิ่งที่เรียกว่า เน็ตเวิร์กเวอร์ชวลเทอร์มินัล (Network Virtual Terminal) ขึ้นมา โดยให้อิดิเตอร์ทำการติดต่อกับเทอร์มินัลแบบนี้ แทนที่จะเป็นตัวเทอร์มินัลจริง จากนั้นจึงเป็นหน้าที่ของซอฟท์แวร์ที่เทอร์มินัล ที่จะต้องทำการจำลองรูปแบบต่างๆ จากเทอร์มินัลเสมือน (Virtual Terminal) ไปยังเทอร์มินัลจริงให้เหมาะสม ตัวอย่างเช่น ถ้าอิดิเตอร์ทำการเคลื่อนเคอร์เซอร์ให้ไปอยู่ที่มุมบนซ้ายของจอของเทอร์มินัลเสมือนแล้ว เทอร์มินัลจริงก็จะเลื่อนเคอร์เซอร์ไปที่มุมบนซ้ายของจอด้วย เทอร์มินัลซอฟต์แวร์ (Terminal Software) เป็นซอฟท์แวร์เฉพาะของเทอร์มินัลแต่ละแบบ และซอฟท์แวร์ที่ว่านี้ก็จะทำหน้าที่ภายใต้ระดับชั้นแอพพลิเคชัน

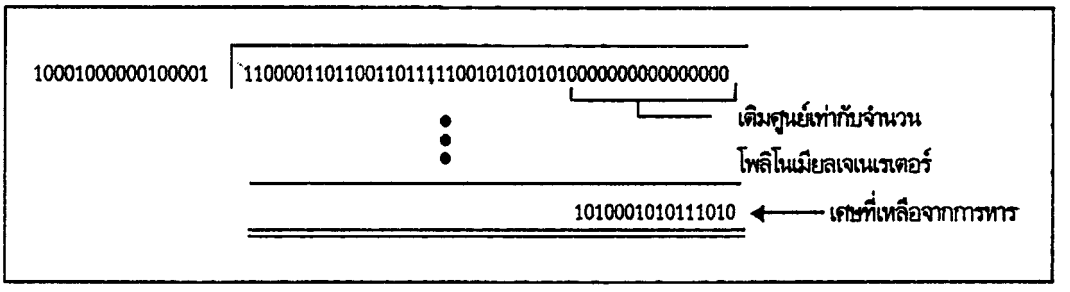
2.6.2 ซีอาร์ซี (CRC, Cyclic Redundancy Check)

ในการส่งข้อมูลจากที่หนึ่งไปยังอีกที่หนึ่ง หรือจากระบบหนึ่งไปยังอีกระบบหนึ่งนั้น ย่อมต้องมีความผิดพลาดของข้อมูลเกิดขึ้นได้เสมอ ดังนั้นเราจำเป็นที่จะต้องมีการป้องกันความผิดพลาดที่จะเกิดขึ้น โดยการตรวจสอบความผิดพลาดของข้อมูล ที่ส่งมาจากด้านต้นทางในแต่ละเฟรมซึ่งมีหลายวิธีด้วยกัน แต่วิธีที่นิยมใช้กันอย่างแพร่หลาย และเป็นวิธีที่มีประสิทธิภาพมากในการตรวจสอบความถูกต้องมากคือ "วิธีไซคลิก รีดันแดนซี เช็ค" หรือเรียกย่อๆ ว่า "ซีอาร์ซี" ซึ่งเป็นการเข้ารหัสข้อมูลทั้งหมดที่ต้องส่งในรูปของรหัส 16 บิต หรือ 32 บิต ขึ้นอยู่กับขนาดของ FCS ฟิลด์ในเฟรมของข้อมูลที่จะส่ง และที่ด้านปลายทางจะทำการตรวจสอบข้อมูลที่รับได้จากด้านต้นทางโดยอาศัยข้อมูลใน FCS ฟิลด์ดังกล่าว วิธีในเข้ารหัสข้อมูลแบบซีอาร์ซีมีอยู่ 2 แบบด้วยกันคือ

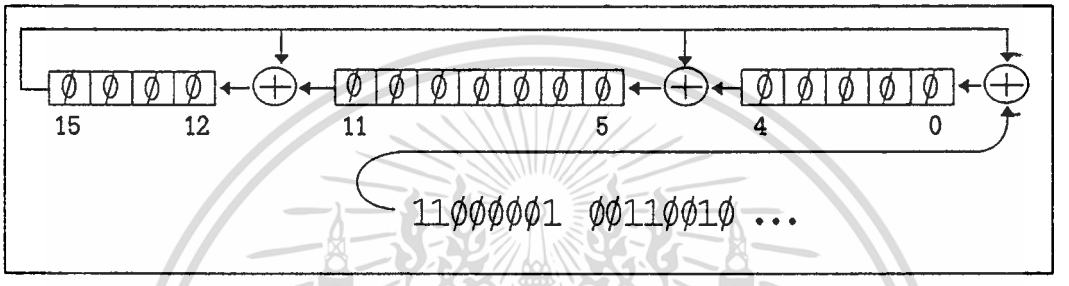
1) วิธีการตั้งหารแบบโมดูโล-2 [Modulo-2]

การตรวจสอบความผิดพลาดโดยวิธีการตั้งหารแบบโมดูโล-2 เป็นวิธีที่นิยมใช้เนื่องจากใช้หน่วยความจำน้อยมากโดยใช้เพียงชิฟท์รีจิสเตอร์เพียงไม่กี่ตัว ซึ่งหาได้โดยทำตามขั้นตอนต่อไปนี้

1) เพิ่มบิตที่เป็น 0 เท่ากับจำนวนบิตของ FCS ฟิลด์ที่ต้องการท้ายข้อมูลที่จะทำการเข้ารหัส
2) ทำการหารข้อมูลที่ได้จากข้อ 1 แบบโมดูโล-2 (ซึ่งในทางดิจิตอลสามารถใช้คุณสมบัติของเอ็กคลูซิพออร์ (Exclusive-OR) ได้) โดยโพลิโนเมียลเจเนอเรเตอร์ (Polynomail Generator) ซึ่งปกติแล้วโพลิโนเมียลเจเนอเรเตอร์จะมีจำนวนบิต มากกว่าจำนวนบิตของ FCS ฟิลด์อยู่ 1 บิตโดยที่บิตเริ่มต้นและบิตสุดท้ายของโพลิโนเมียลเจเนอเรเตอร์จะต้องเป็น 1 เสมอ
3) เศษที่ได้จากการหารเมื่อใช้จำนวนบิตของข้อมูลที่ได้จากข้อ 1 หมดแล้ว ค่าดังกล่าว คือ ข้อมูลใน FCS ฟิลด์ที่ได้จากการเข้ารหัสแบบซีอาร์ซี
4) เมื่อด้านปลายทางได้รับข้อมูลที่ส่งจากด้านต้นทางแล้ว จะทำการถอดรหัสโดยการหารแบบโมดูโล-2 เช่นเดียวกับด้านต้นทาง แต่บทที่เพิ่มต่อท้ายจะเป็นข้อมูลใน FCS ฟิลด์ที่ส่งมาแทนที่จะเป็น 0 เหมือนการเข้ารหัสของด้านต้นทาง และเศษที่ได้จากการหาร ถ้าเป็น 0 แสดงว่าข้อมูลดังกล่าวที่ส่งมาไม่ผิดพลาด แต่ถ้าไม่เท่ากับ 0 แสดงว่าข้อมูลดังกล่าวที่ส่งมาผิดพลาด ในทางปฏิบัติแล้วโพลิโนเมียลเจเนอเรเตอร์นิยมใช้ขนาด 16 บิต ซึ่งตามมาตรฐานของ CCITT โดยใช้ $X^{16} + X^{12} + X^{5} + 1$ และที่นิยมใช้อีกตัวหนึ่งคือ $X^{16} + X^{15} + X^{2} + 1$ ซึ่งโอกาสที่จะหาข้อมูลผิดพลาดได้ถูกต้องมีถึง 99.99 เปอร์เซ็นต์



รูปที่ 2.13 วิธีการตั้งหารแบบโมดูโล-2



รูปที่ 2.14 ฮาร์ดแวร์จำลองการหารแบบโมดูโล-2 โดยใช้ชิฟรีจิสเตอร์ (Shift Register) และเอ็กคลูซีฟออร์เกต (Exclusive-OR Gate)

```
#define CRC_POLY    0x1021
unsigned int CRC_16_Byte ( crc, msg, msglen )
unsigned int crc    /* starting value for CRC */
char *msg           /* pointer to message block */
int msglen          /* number of bytes in block */
{
        int j;
        while ( msglen-- > 0 ) {
                crc = ( crc ^ *msg++) & 0xff;
                for ( j = 0 ; j < 8 ; j++ ) {
                        if (crc & 0x0001)
                                crc = ( crc >> 1 ) ^ CRC_POLY;
                        else
                                crc = crc >> 1;
                }
        }
        return ( crc );
}
```

รูปที่ 2.15 ฟังก์ชันหาค่าซีอาร์ซีทีละไบต์ (ภาษาซี)

2) วิธีเทียบจากตาราง (Look-up Table)

จะเห็นว่าวิธีแรกต้องเสียเวลาในการตั้งหารข้อมูลที่เข้ามา หรือการเลื่อนข้อมูลทีละบิต จึงได้มีการปรับปรุงโดยใช้วิธีการเทียบจากตารางแทน กล่าวคือ จะนำค่าการเข้ารหัสแบบซีอาร์ซีสำหรับแต่ละค่าของข้อมูลมาเก็บไว้เป็นตาราง และใช้ข้อมูลที่จะหาค่าเข้ารหัสแบบซีอาร์ซีเป็นตัวชี้ค่าในตารางแทน ซึ่งทำให้ใช้เวลาน้อยลง แต่เปลืองหน่วยความจำมากกว่าวิธีแรก สมมติให้ใช้โพลิโนเมียลเจเนอเรเตอร์ คือ $X^{16} + X^{12} + X^{5} + 1$ และ ข้อมูลที่เข้ามาคือ 0x02

0000 0000 0000 0000 ข้อมูลการเข้ารหัสแบบซีอาร์ซีเริ่มต้น

0000 0010 XOR กับข้อมูลที่นำมาเข้ารหัส 0x02

0000 0000 0000 0010

ตารางซีอาร์ซี 16 บิท ของโพลิโนเมียล 0x1021

0x0000	0x1021	0x2042	
0x8108	0x9129	0xA14A	

0000 0010 0100 0010 ค่ าซีอาร์ซีที่ได้จากตาราง

0000 0000 0000 0000 XOR กับค่าซีอาร์ซีเดิ มคือ 0

0000 0010 0100 0010 ค่าซีอาร์ซีใหม่

```
unsigned int crctable[256] =  /* use ccitt  1021 hex */
{
      0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
      0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
      0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
      0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
      0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
      0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
      0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
      0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
      0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
      0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
      0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
      0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
      0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
      0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
      0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
      0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
      0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
      0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
      0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
      0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
      0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
      0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
      0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
      0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
      0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
      0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
      0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
      0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
      0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
      0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
      0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
      0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};
```

รูปที่ 2.16 ฟังก์ชันในการหาค่าเข้ารหัสแบบซีอาร์ซี

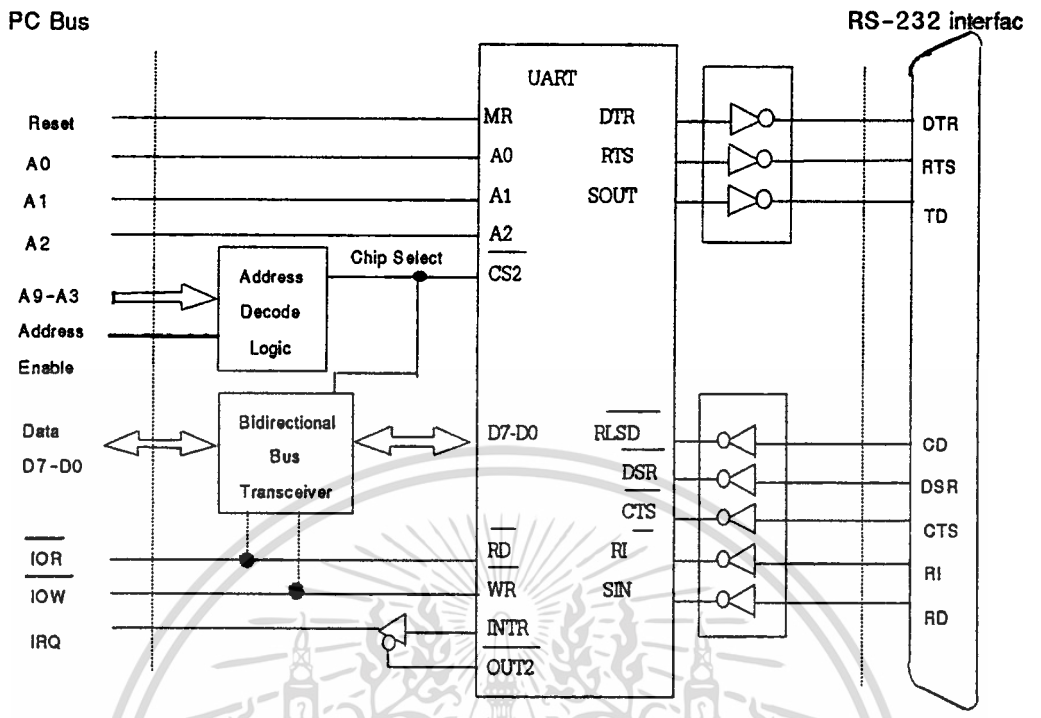
```
unsigned int update_crc(unsigned int crc,int ch)
{
   return (crc>>8)^crctable[(crc^ch) & 0xff];
}
```

รูปที่ 2.17 ฟังก์ชันในการหาค่าเข้ารหัสแบบซีอาร์ซี (ต่อ)

2.6.3 พอร์ตสื่อสารแบบอนุกรม

ใน PC โดยทั่วไปจะมีอะแดปเตอร์สื่อสารแบบอะซิงโครนัส (Asychronous) เป็นการ์ดอะแดปเตอร์แบบอนุกรมและขนานในการ์ดอันเดียวกัน สำหรับอะแดปเตอร์แบบอนุกรมจะแบ่งได้เป็น 3 ประเภทดังนี้

- ประเภทแรกเป็นอะแดปเตอร์ที่ใช้ชิพ (Chip) เบอร์ 8250 หรือ 16450 UART เป็นอะแดปเตอร์ที่พื้นฐานที่สุดสำหรับการสื่อสารโดยทั่วไป
- ประเภทที่สองคล้ายๆ ประเภทแรกแต่จะใช้ชิพเบอร์ 16550 และ FIFO (First In First Out) บัฟเฟอร์ ซึ่งมีความประสิทธิภาพมากกว่า ใช้สำหรับการสื่อสารโมเด็มความเร็วสูงและสภาวะแวดล้อมการทำงานแบบหลายๆ งาน (Multitasking Environments) แต่จะมีราคาแพงกว่าประเภทแรก
- ประเภทที่สามประกอบด้วยอะแดปเตอร์แบบพิเศษคือ ตัวควบคุมพอร์ตอนุกรม (Serial Port Controller) และ เฮส์เอนฮานซ์ดีเรียลอินเตอร์เฟซ (Hayes Enhanced Serial Interface, HESI) สำหรับประเภทที่สามนี้จะสนับสนุน การถ่ายเทข้อมูลแบบ DMA (Direct Memory Access) และมีโฟลว์คอนโทรล (flow control) ของตัวเอง HESI เป็นโคโพรเซสเซอร์ (coprocessor) ทางการสื่อสารที่สมบูรณ์แบบ สามารถใช้ไมโครโพเซสเซอร์ของตนเองในการจัดการสื่อสารโดยไม่ขึ้นกับประเภทของคอมพิวเตอร์



รูปที่ 2.18 แสดงบล็อกไดอะแกรมพอร์ตอนุกรม

พอร์ตอนุกรม ประกอบไปด้วยโครงสร้างหลายอย่างด้วยกัน คือ

2.6.3.1 ไอโอแอดเดรส (I/O Address)

ในคอมพิวเตอร์ จะมีไอโอแอดเดรสสำหรับใช้อ้างถึงอะแดปเตอร์แบบอนุกรมอยู่ 4 ช่วง ดังตารางที่ 2.3 ไอโอแอดเดรส ดังกล่าวเชื่อมต่อกับ UART (Universal Asyncronous Reciver Transmitter) ซึ่งทำหน้าที่ในการติดต่อสื่อสารแบบอะซิงโครนัส และอินเตอร์เฟสแบบ RS-232 เมื่อเราติดตั้งพอร์ตอนุกรมอันแรก (COM1) จะมีฐานแอดเดรสอยู่ที่ 03F8H โดยเราสามารถอ้างถึงรีจิสเตอร์ใน UART จากฐานของแอสเดรสนี้ เช่น รีจิสเตอร์ตัวที่ 4 ของ COM1 จะมีแอดเดรสตรงกับ PC I/O แอดเดรสที่ 03F8H + 4 = 03FCH

ตารางที่ 2.3 แสดงความสัมพันธ์ระหว่างเบอร์พอร์ตอนุกรม และไอโอแอดเดรส

พอร์ตอนุกรมหมายเลข	ฐานไอโอแอดเดรส	IRQ มาตรฐาน.
COM1	03F8H-03FFH	4
COM2	02F8H-02FFH	3
COM3	03E8H-03EFH	4
COM4	02E8H-02EFH	3

หมายเหตุ COM1, COM2, COM3 และ COM4 เป็นชื่อของอุปกรณ์ (Device) ที่ตั้งโดยระบบปฏิบัติการ MS-DOS สำหรับพอร์ตอนุกรมอันที่ 1 ถึงอันที่ 4 ตามลำดับ ที่ติดตั้งอยู่ในระบบ เพราะโปรแกรมส่วนใหญ่จะติดต่อกับฮาร์ดแวร์โดยตรง ดังนั้นชื่อเหล่านี้จึงเป็นที่รู้จักโดยทั่วไป เมื่ออ้างถึงการสื่อสารแบบอะซิงโครนัส

2.6.3.2 การเชื่อมต่อกับ PC บัส (Bus Interface)

การอ่าน และเขียนข้อมูลผ่านรีจิสเตอร์ของ UART ไอโอแอดเดรสของรีจิสเตอร์นั้น จะวางอยู่บน PC แอดเดรสบัส PC จะสโตรป (strobe) เส้นสัญญาณ IOR (I/O read) หรือ IOW (I/O write) เมื่อส่วนวงจรถอดรหัสแอสเดรส (Address Decoding Logic) ที่อยู่บนการ์ดอนุกรมตรวจสอบพบว่าเป็นแอดเดรสของตัวมันเอง มันจะทำการกระตุ้นสัญญาณเลือกชิป (chip select) ของ UART (CS2) การอ้างถึงรีจิสเตอร์ภายในของ UART สามารถทำได้โดยการถอดรหัส (Decode Address) จากเส้นแอดเดรส A_2-A_0ซึ่งเป็น 3 บิตต่ำของ I/O แอดเดรส

สัญญาณเลือกชิปจะทำให้สัญญาณข้อมูลของ UART เชื่อมต่อกับบัสข้อมูลของ PC โดยการควบคุมแบบรับส่งสองทิศทาง (Bidirectional Bus Transceiver) ทิศทางการไหลของข้อมูลที่ผ่านตัวรับส่ง (Tranceiver) จะได้จากการตรวจสอบสัญญาณ IOR และ IOW ของ PC ซึ่งสัญญาณเหล่านี้จะต่อกับสัญญาณ RD และ WR ของ UART

2.6.3.3 การเชื่อมต่ออินเทอร์รัพต์ (Interrupt Interface)

สำหรับแต่ละพอร์ตอนุกรมจะมีสัญญาณอินเทอร์รัพต์โดยต่อสัญญาณ INTR เข้ากับบัสของ PC โดยได้กำหนดให้ IRQ 4 ใช้ร่วมกันระหว่าง COM1 และ COM3 ส่วน IRQ 3 ใช้ร่วมกันระหว่าง COM2 และ COM4 หมายความว่าเราไม่สามารถที่จะใช้ COM1 และ COM3 โดยใช้แบบอินเทอร์รัพต์ได้ในเวลาเดียวกัน

จากบล็อกไดอะแกรมโครงสร้างพอร์ตอนุกรมข้างต้น จะเห็นว่าสัญญาณ INTR ไม่ได้ต่อโดยตรงกับสัญญาณ IRQ ที่อยู่บนบัสของ PC แต่จะมีลอจิกเกต (Logic Gate) คั่นกลางซึ่งควบคุมการเปิดปิดของเกตนี้โดยใช้สัญญาณ OUT2 ของ UART ดังนั้นในการที่จะทำให้เกิดการอินเทอร์รัพต์ขึ้นจึงต้องมีการโปรแกรมให้เซตบิต OUT2 ที่อยู่ในรีจิสเตอร์ควบคุมโมเด็ม (Modem Control Register) เป็น 1 เสียก่อน

2.6.3.4 อินเตอร์เฟส RS-232

ขา DTR, RTS และ SOUT ของ UART จะต่อกับอินเตอร์เฟสแบบ RS-232 โดยผ่านตัวขับสัญญาณ (Line Drivers) สำหรับตัวขับสัญญาณนี้จะแปลงสัญญาณ 0V และ +5V ของ UART เป็น -12V และ +12V ตามมาตรฐานของ RS-232 เช่นเดียวกันกับขาสัญญาณอินพุท CD, DSR, CTS, RI และ RD ของ RS-232 ก็จะต่อกับขาของ UART ที่สอดคล้องกัน ในการออกแบบตัวขับสัญญาณของ RS-232 นั้นจะใช้ตัวขับสัญญาณที่มีลอจิกตรงข้ามคือมันจะทำการกลับค่าลอจิกอินพุทที่มีมาจาก UART

11 อินเทอร์รัพต์ของสถานะสายทางด้านรับ ความสำคัญสูงสุด

10 อินเทอร์รัพต์ของการมีข้อมูลมา ความสำคัญอันดับสอง

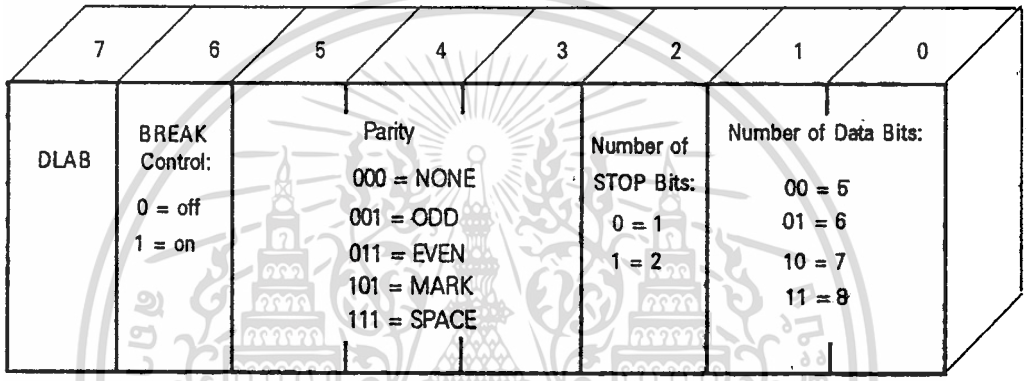
01 อินเทอร์รัพต์การว่างของรีจิสเตอร์ Transmitter Holing ความสำคัญอันดับสาม

00 อินเทอร์รัพต์ของสถานะของโมเด็ม ความสำคัญอันดับสี่

บิต 0 บอกการค้างของอินเตอร์รัพต์

0 มีอินเทอร์รัพต์ค้างอยู่ ค่าที่อยู่ในรีจิสเตอร์สามารถใช้ในการชี้ประเภทของอินเทอร์รัพต์ที่เกิดขึ้น

1 ไม่มีอินเทอร์รัพต์ค้างอยู่



พอร์ต 03FBH

ขณะอ่าน/เขียน รีจิสเตอร์ควบคุมสาย (Line Control Register)

บิต 7 เป็นแฟล็ก

1 เป็นบิต Divisor Latch Access (DLAB)

0 เป็นการอ้างถึง Receiver Buffer, Transmitter Holding หรือ รีจิสเตอร์การเปิดการอินเทอร์รัพต์ (Interrupt Enable Register)

บิต 6 เป็นการตั้งค่าเปิดเบรค

1 เซตการเปิดเบรค(Break Enable)

บิต 5-4 ประเภทของพาริตี้

00 พาริตี้คี่ (Odd Parity)

01 พาริตี้คู่ (Even Parity)

10 พาริตี้มาร์ค (Mark Parity)

11 พาริตี้ว่าง (Space Parity)

บิต 3 กำหนดพาริตี

1 ให้มีบิตพาริตี

บิต 2 กำหนดบิตหยุด

0 มี 1 บิตหยุด (Stop Bit)

1 ไม่มีบิตหยุด

บิต 1-0 ความยาวของเวิร์ด (Word Length)

00 มีความยาว 5 บิต

01 มีความยาว 6 บิต

10 มีความยาว 7 บิต

11 มีความยาว 8 บิต

7	6	5	4	3	2	1	0
Alway 0	Alway 0	Alway 0	Local Loopback Test	General Purpose Output 2	General Puepose Output 1	RTS	DTR

พอร์ต 03FCH

ขณะอ่าน/เขียน เป็นรีจิสเตอร์ควบคุมโมเด็ม

บิต 7-5 = 000 สงวนไว้

บิต 4 = 1 โหมดลูปแบ็ค (Loopback Mode) สำหรับการตรวจสอบพอร์ตอนุกรมโดยใช้เอาท์พุทของรีจิสเตอร์ Transmitter Shift ป้อนลูปแบ็คกลับไปยังอินพุทของรีจิสเตอร์ Receiver Shift ในโหมดนี้ข้อมูลที่ส่งออกไปจะได้รับในทันที ดังนั้นหน่วยประมวลผลกลาง สามารถที่จะยืนยันถึงข้อมูลที่ส่ง และรับจากพอร์ตอนุกรมว่าถูกต้องหรือไม่

บิต 3 = 1 เอาท์พุทของ Auxiliary User-designated 2

บิต 2 = 1 เอาท์พุทของ Auxiliary User-designated 1

บิต 1 = 1 บังคับให้ Request-To-Send แอคทีฟ

บิต 0 = 1 บังคับให้ Data-Terminal-Ready แอคทีฟ

7	6	5	4	3	2	1	0
Alway 0	Transmitter Empty (TXE)	Transmitter Buffer Empty (TBE)	BREAK Detect	Frameing Error	Parity Error	Overrun Error (RxRDY)	Data Ready

พอร์ต 03FDH

ขณะอ่าน เป็นรีจิสเตอร์สถานะของสาย

บิต 7=0 สงวนไว้

บิต 6=1 รีจิสเตอร์ nd holding ว่าง

บิต 5=1 รีจิสเตอร์ transmitter holding register ว่างพร้อมที่จะรับตัวอักขระใหม่จะส่งต่อไป

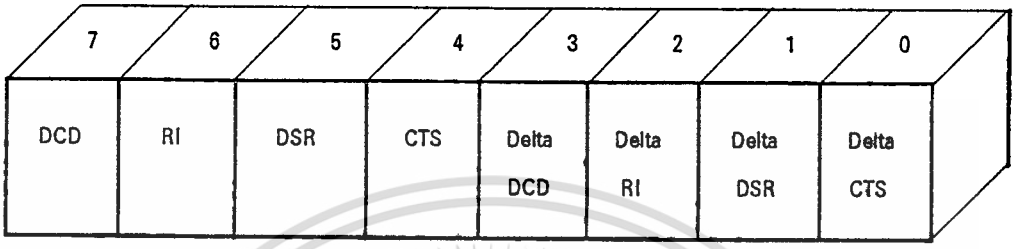
บิต 4=1 เป็นการหยุดอินเทอร์รัพต์ (break interrupt)

บิต 3=1 เกิดความผิดพลาดทางเฟรม (framing error)

บิต 2=1 เกิดความผิดพลาดทางพาริตี้ (parity error)

บิต 1=1 เกิดความผิดพลาดทาง overrun

บิต 0=1 มีข้อมูลมาพร้อมแล้วที่รีจิสเตอร์ receiver buffer



พอร์ต 03FEH

ขณะอ่าน เป็นรีจิสเตอร์ควบคุมโมเด็ม

บิต 7 = 1 Data Carrier Detect (DCD)

บิต 6 = 1 Ring Indicator (RI)

บิต 5 = 1 Data Set Ready (DSR)

บิต 4 = 1 Clear To Send (CTS)

บิต 3 = 1 Delta Data Carrier Detect (DDCD)

บิต 2 = 1 Trailing Edge Ring Indicator (TERI)

บิต 1 = 1 Delta Data Set Ready (DDSR)

บิต 0 = 1 Delta Clear To Send (DCTS)

บิต 0 - 3 จะถูกรีเซตเมื่อ CPU อ่านค่าในรีจิสเตอร์สถานะของโมเด็ม

บิต 4 เป็น RTS ของรีจิสเตอร์สถานะของโมเด็มขณะที่มีการทดสอบลูปแบ็ค

บิต 5 เป็น DTR ของรีจิสเตอร์สถานะของโมเด็มขณะที่มีการทดสอบลูปแบ็ค

บิต 6 เป็น OUT1 ของรีจิสเตอร์สถานะของโมเด็มขณะที่มีการทดสอบลูปแบ็ค

บิต 7 เป็น OUT2 ของรีจิสเตอร์สถานะของโมเด็มขณะที่มีการทดสอบลูปแบ็ค

พอร์ต 03FFH

ขณะอ่าน/เขียน เป็นรีจิสเตอร์ Scratch ไม่มีหน้าที่อะไรพิเศษที่เกี่ยวกับ UART และสามารถใช้เป็นที่เก็บข้อมูลชั่วคราวขนาด 1 ไบต์

2.6.4 โมเด็ม

โมเด็ม เป็นอุปกรณ์ในการแปลงสัญญาณดิจิตอล ซึ่งมักเป็นสัญญาณจากคอมพิวเตอร์ เป็นสัญญาณอะนาล็อก เพื่อส่งไปตามสายโทรศัพท์ และแปลงกลับเป็นสัญญาณดิจิตอล เมื่อข้อมูลไปถึงปลายทาง

2.6.4.1 การติดต่อและควบคุมโมเด็ม

โมเด็มในยุคแรกการใช้งานต้องอาศัยคนช่วยเกือบทุกอย่าง เนื่องจากโมเด็มทำหน้าที่แค่การรับส่งสัญญาณและแปลงสัญญาณที่ได้รับมาให้กับคอมพิวเตอร์เท่านั้น ส่วนการหมุนโทรศัพท์ และการติดต่อกับอีกฝั่งหนึ่งเพื่อส่งข้อมูล ไปจนถึงการรับโทรศัพท์สำหรับโมเด็มด้านรับนั้น ต้องให้คนเป็นผู้ทำสิ่งเหล่านี้

ต่อมา บริษัทเฮย์สไมโครคอมพิวเตอร์โปรดักส์ (Hayes Microcomputer Products Inc) เป็นผู้คิดริเริ่มการสั่งงานโมเด็มโดยใช้คำสั่งแทน และได้รับความนิยมอย่างมากจนถือเป็นมาตรฐานอันหนึ่ง มาตรฐานคำสั่งนี้เรียกว่าเฮย์สคอมมานด์เซต (Hayes Command Set) เป็นคำสั่งที่ช่วยให้ผู้ใช้สามารถกำหนดการทำงานต่างๆ ของโมเด็มได้ โดยใช้ซอฟต์แวร์สั่งจากคอมพิวเตอร์ไปยังโมเด็มได้โดยตรง โมเด็มที่เราใช้กับเครื่องคอมพิวเตอร์ เกือบทั้งหมดจะรับคำสั่งตามมาตรฐานของเฮย์สนี้ ซึ่งคำสั่งต่างๆ จะกล่าวถึงต่อไป

เมื่อคำสั่งของโมเด็ม เป็นสิ่งจำเป็นสำหรับการใช้งานโมเด็มสมัยใหม่ ทาง CCITT (International Telephone and Telegraph Consultative Committee) ก็ได้มีการกำหนดมาตรฐานของคำสั่งโมเด็มขึ้น เรียกว่ามาตรฐานคำสั่งแบบ V.25 bis แต่เนื่องจากว่าโมเด็มที่ใช้กับเครื่องคอมพิวเตอร์และซอฟต์แวร์ สั่งงานโมเด็มเกือบทั้งหมดใช้คำสั่งมาตรฐานของเฮย์ส ดังนั้นมาตรฐานคำสั่งโมเด็มแบบ V.25 bis จึงไม่ค่อยมีใช้กันเท่าไร เรียกได้ว่าเครื่องคอมพิวเตอร์ติดต่อกับโมเด็มโดยใช้คำสั่งแบบเฮย์สคอมมานด์เกือบทั้งหมด ส่วนคำสั่งแบบ V.25 bis มีใช้บ้างเหมือนกัน แต่เป็นในระบบชุมสายตามมาตรฐานของ CCITT เท่านั้น โมเด็มบางแบบสามารถรับคำสั่ง ได้ทั้งแบบเฮย์สคอมมานด์ และแบบ V.25 bis ของ CCITT แต่โมเด็มราคาถูกที่ใช้กับคอมพิวเตอร์ส่วนมาก จะรับคำสั่งได้แต่เฮย์สคอมมานด์เท่านั้น ผู้ผลิตโมเด็มบริษัทอื่นๆ ได้ใช้มาตรฐานคำสั่งตามอย่างบริษัทเฮย์ส ซึ่งเป็นต้นฉบับ โดยรับคำสั่งจากคอมพิวเตอร์ ตามแบบคำสั่งของเฮย์สทุกประการ เรียกว่าเป็นโมเด็มที่เข้ากันได้กับแบบของเฮย์ (Hayes Compatible)

การติดต่อและควบคุมโมเด็ม สามารถกระทำได้โดยการส่งคำสั่งที่ขึ้นต้นด้วย AT เสมอจึงมีอีกชื่อหนึ่งเรียกว่าเอทีคอมมานด์ (AT Command) เมื่อจบคำสั่งให้ปิดท้ายด้วยคาร์ริเอจรีเทิร์น (Carriage Return) โมเด็มจะรับคำสั่งนั้นไปทำงานทันที และตอบสนองกลับมา เราสามารถแบ่งคำสั่งได้เป็น 4 กลุ่ม ดังนี้

- กลุ่มคำสั่งที่เกี่ยวกับการติดต่อกับผู้ใช้ (User Interface)
- กลุ่มคำสั่งเบื้องต้นในการหมุนและรับโทรศัพท์ (Primary Dial/Answer)
- กลุ่มคำส่งเพิ่มเติมในการหมุนโทรศัพท์ (Dial Modifier)
- กลุ่มคำสั่งเบ็ดเตล็ด (Miscellaneous)

2.6.4.1.1 กลุ่มคำสั่งที่เกี่ยวกับการติดต่อกับผู้ใช้ (User Interface)

เป็นคำสั่งที่ใช้ในการติดต่อระหว่างโมเด็มและผู้ใช้แต่ไม่มีผลกับการปฏิบัติการทางไฟฟ้าของโมเด็ม

- ATB เป็นคำสั่งเลือกการทำงานของโมเด็ม ว่าจะใช้มาตรฐานผสมสญญาณตาม CCITT หรือ มาตรฐานของสหรัฐอเมริกา (Bell Standard) โดยใช้คำสั่ง ATB0 สำหรับมาตรฐาน CCITT และ ATB1 สำหรับ มาตรฐานสหรัฐอเมริกา ปกติเราจะใช้มาตรฐาน CCITT เท่านั้น และโมเด็มมักจะตั้งค่า ATB0 มาจากโรงงานอยู่แล้ว

- ATE สั่งให้โมเด็มส่งข้อความกลับ (echo) ไปให้เครื่องคอมพิวเตอร์ โดยใช้คำสั่ง ATE0 สำหรับไม่ให้โมเด็มส่งข้อความกลับ และ ATE1 เพื่อให้โมเด็มส่งข้อความกลับไปให้เครื่องคอมพิวเตอร์ ในขณะที่โมเด็ม รับคำสั่งอยู่ในโหมดคำสั่ง เมื่อเราใช้โมเด็มรับส่งข้อมูลแบบการติดต่อแบบสมบูรณ์ (full duplex) จะต้องเลือกใช้ ATE1 เพื่อให้ข้อความที่เราพิมพ์ส่งไปยังโมเด็มส่งข้อความกลับมาให้เราเห็นบนจอภาพ และถ้าเป็นการติดต่อแบบ กึ่งสมบูรณ์ (Half Duplex) ก็เลือกใช้ ATE0 เพราะเครื่องคอมพิวเตอร์จะพิมพ์ข้อความนั้นบนจอภาพอยู่แล้ว คำสั่งนี้จะมีผลเฉพาะก่อนการเชื่อมต่อกับปลายทางเท่านั้น

- ATL ใช้สำหรับเป็นตัวปรับความดังของลำโพงภายในโมเด็ม ซึ่งปรับความดังได้ 3 ระดับนี้ คือ ATL0 และ ATL1 เป็นคำสั่งปรับความดังให้น้อยที่สุด ATL2 จะเป็นระดับความดังปานกลาง และ ATL3 สำหรับปรับความดังของลำโพงมากที่สุด แต่โมเด็มบางแบบจะใช้ปุ่มหมุนปรับความดังของลำโพงแทน

-ATV เป็นคำสั่งบอกให้โมเด็มแสดงผลลัพธ์ของคำสั่งต่างๆ เป็นรหัสตัวเลข (digit code) หรือเป็นตัวอักษรภาษาอังกฤษ ATV0 จะสั่งให้โมเด็มแสดงผลลัพธ์ของคำสั่งกลับมาเป็นตัวเลข และ ATV1 จะสั่งให้โมเด็มแสดงผลลัพธ์ของคำสั่งกลับมาเป็นตัวอักษร เช่น เมื่อเราสั่งให้โมเด็มหมุนโทรศัพท์ด้วยคำสั่ง ATDT 2345678 <CR> โมเด็มจะตอบกลับมาว่า OK ในแบบตัวอักษร หรือตอบกลับมาเป็นเลข 0 ในแบบตัวเลข และถ้าโมเด็มต่อกับปลายทางได้ก็จะตอบว่า "CONNECT 1200" ในกรณีที่ใช้ความเร็ว 1200 บิตต่อวินาที หรือตอบเป็นรหัสตัวเลข 5 แทน ปกติเราจะใช้โมเด็มตอบผลลัพธ์เป็นตัวอักษร เพื่อให้เข้าใจง่ายว่าขณะนี้โมเด็ม มีสภาพเป็นอย่างไร ซอฟต์แวร์ควบคุมโมเด็มทั่วไป ก็มักจะให้โมเด็มตอบรับเป็นตัวอักษรเช่นกัน โมเด็มที่เป็นแบบเข้ากันได้กับแบบของเฮย์สจึงต้องแสดงผลลัพธ์ของคำสั่งต่างๆ ให้เหมือนกับที่บริษัทเฮย์สใช้ทุกตัวอักษร มิฉะนั้นซอฟต์แวร์ควบคุมโมเด็มจะสั่งงานโมเด็มผิดพลาดได้ หรือสั่งงานโมเด็มไม่ได้เลย เนื่องจากโมเด็มตอบรหัสกลับไป ไม่ตรงตามที่บริษัทเฮย์สกำหนด

ตารางที่ 2.4 แสดงการตอบสนองของสมาร์ทโมเด็ม

โค้ดตัวเลข	โค้ดคำ	คำอธิบาย
0	OK	คำสั่งที่ไม่ใช่คำสั่งในการหมุนโทรศัพท์ เมื่อทำคำสั่งนั้นได้สำเร็จ
1	CONNECT	เมื่อได้รับสัญญาณตอบรับ
2	RING	เมื่อมีสัญญาณเรียก
3	NO CARRIER	เมื่อไม่มีสัญญาณตอบรับ

4	ERROR	เมื่อมีความผิดพลาดในคำสั่ง หรือ ไม่มีคำสั่งดังกล่าว หรือ บัพเฟอร์คำสั่งเกิน 40 ตัวอักษร
5	CONNECT 1200	เมื่อได้รับสัญญาณตอบรับโดยสามารถส่งข้อมูลที่ความเร็ว 1200 บิตต่อวินาที

- ATM ใช้สำหรับควบคุมการทำงานของลำโพงภายในโมเด็ม เช่น ATM0 จะสั่งไม่ให้ลำโพงทำงาน คือ เงียบตลอดเวลา ATM1 จะให้ลำโพงมีเสียง เฉพาะตอนที่ยังต่อกับปลายทางไม่ได้เท่านั้นเพื่อให้เราฟังเสียง การหมุนโทรศัพท์และสัญญาณตอบรับได้ว่ามีอะไรผิดปกติหรือเปล่า ATM2 จะสั่งให้ลำโพงมีเสียงตลอดเวลา

2.6.4.1.2 กลุ่มคำสั่งเบื้องต้นในการหมุนและรับโทรศัพท์ (Primary Dial/Answer)

- ATA เป็นคำสั่งให้โมเด็มตอบรับสัญญาณโทรศัพท์ที่เรียกเข้ามา เมื่อโมเด็มทำคำสั่งนี้การติดต่อระหว่างปลายทางทั้งสองข้างจะเริ่มขึ้น และโมเด็มจะอยู่ในช่วงรับส่งข้อมูล ไม่รับคำสั่งจากเครื่องคอมพิวเตอร์ จนกว่าจะเลิกการติดต่อกับปลายทางเสียก่อน

- ATD ใช้สั่งให้โมเด็มทำการหมุนโทรศัพท์อัตโนมัติ (auto dialing) แทนการที่ใช้คนหมุนเอง คำสั่งนี้มักตามด้วย ชนิดของโทรศัพท์ที่ต่ออยู่ด้วย เช่น ATDT จะเป็นคำสั่งให้โมเด็มหมุนโทรศัพท์แบบ กดปุ่ม และ ATDP จะเป็นคำสั่งสำหรับหมุนโทรศัพท์แบบมือหมุน

- ATH เป็นคำสั่งให้โมเด็มวางสายโทรศัพท์ หรือ ปลดตัวเองออกจากสายโทรศัพท์ และวางหูนั่นเอง ส่วนมากคำสั่งนี้จะใช้เมื่อเลิกติดต่อกับปลายทาง โดยสั่ง ATH เพื่อวางหูหลังจากส่งข้อมูลจบแล้ว ก่อนใช้คำสั่งนี้ เราต้องสั่งให้โมเด็มกลับมาอยู่ในสภาวะรับคำสั่งเสียก่อน ซึ่งทำได้โดยส่งเครื่องหมาย + สามตัวไปให้โมเด็ม แต่ละตัวห่างกันประมาณครึ่งวินาที โมเด็มจะถือว่าจบการส่งข้อมูล และรอรับคำสั่งจากเครื่องคอมพิวเตอร์ต่อไป

2.6.4.1.3 กลุ่มคำส่งเพิ่มเติมในการหมุนโทรศัพท์ (Dial Modifier)

เป็นคำสั่งที่ใช้ในคำสั่ง ATD เราเรียกอีกอย่างว่า ส่วนขยายการหมุนโทรศัพท์ (Dialing adverbs)

- P คำสั่งสำหรับให้หมุนโทรศัพท์แบบมือหมุน
- T คำสั่งสำหรับให้หมุนโทรศัพท์แบบกดปุ่ม
- , เครื่องหมายจุลภาค คำสั่งให้หยุดรอสักพัก
- ; คำสั่งให้กลับมาที่โหมดคำสั่งหลังจากที่หมุนโทรศัพท์แล้ว

2.6.4.1.4 กลุ่มคำสั่งเบ็ดเตล็ด (Miscellaneous)

- ATO เป็นคำสั่งให้โมเด็มอยู่ในโหมดออนไลน์ (online) คือเปลี่ยนแปลงจากรับคำสั่งจากเครื่องคอมพิวเตอร์มาเป็นการรับส่งข้อมูลผ่านทางสาย

-ATS ใช้สำหรับเปลี่ยนแปลงค่าของรีจิสเตอร์เอส (S-register) ซึ่งเป็นที่เก็บพารามิเตอร์ในการทำงานต่างๆ เฮย์ส เช่น S0 ใช้เก็บจำนวนครั้งของสัญญาณเรียกเข้า ก่อนที่โมเด็มจะทำการรับโทรศัพท์ ถ้าเราต้องการ ให้

โมเด็มรับโทรศัพท์ที่เรียกเข้ามาโดยมีสัญญาณเรียกดัง 4 ครั้ง ก็สั่งว่า ATS0 = 4 เป็นต้น รีจิสตอร์เอส ของเฮย์ส แสดงในตารางประกอบว่าตัวไหนมีหน้าที่อะไร นอกจากนี้คำสั่ง ATS ยังใช้เรียกดูค่าของรีจิสเตอร์เอสได้อีกด้วย ว่าค่าที่เก็บอยู่ในปัจจุบันเป็นเท่าไร โดยใช้คำสั่ง เช่น ATS0? โมเด็มจะนำค่าของ S0 ที่เก็บอยู่ในขณะนั้นแสดงให้เราดู

- ATX เป็นคำสั่งโมเด็มแสดงผลลัพธ์ในการหมุนโทรศัพท์แบบต่างๆ เช่น ATX0 และ ATX1 จะเป็น คำสั่งให้โมเด็มไม่ตรวจสอบเสียงเรียกหมุน (dial tone) ก่อนหมุนโทรศัพท์ และไม่รับรู้สัญญาณที่ตอบกลับ มาว่าปลายทางไม่ว่าง (busy signal) ATX2 จะให้โมเด็มตรวจสอบสายก่อนว่าสายโทรศัพท์มีเสียงเรียกหมุน ก่อนทำการหมุน แต่ไม่รับรู้สัญญาณปลายทางไม่ว่างเหมือนเดิม ถ้าหากโมเด็มไม่ได้รับเสียงเรียกหมุนจากชุมสายใน 5 วินาที โมเด็มจะตอบกลับไปยังคอมพิวเตอร์ว่า "NO DIALTONE" ATX3 จะให้โมเด็มรับรู้สัญญาณปลายทาง ไม่ว่างได้ และ ATX4 ซึ่งเป็นคำสั่งที่เราใช้เป็นส่วนมากจะให้โมเด็มตรวจสอบเสียงเรียกหมุนก่อนที่ จะทำการหมุนโทรศัพท์ และรับรู้สัญญาณปลายทางไม่ว่างได้ด้วย คือเท่ากับ ATX2 รวมกับ ATX3 นั่นเอง

- ATZ เป็นคำสั่งให้รีเซ็ตโมเด็มกลับมาอยู่ในสภาวะที่ผู้ผลิตกำหนด (default configuration) เมื่อโมเด็มได้รับคำสั่งนี้มันจะยกเลิกคำสั่งเปลี่ยนแปลงค่าต่างๆ ที่ได้รับมาทั้งหมด และดึงค่าพารามิเตอร์ที่ผู้ผลิตกำหนดเอาไว้จากโรงงาน ซึ่งเก็บอยู่ในรอมของโมเด็มมาใช้ คำสั่งนี้มีประโยชน์มากในกรณีที่ต้องการให้ค่าต่างๆ กลับคืนมาเหมือนเดิม

บทที่ 3
การคำนวณและการสร้าง

3.1 การออกแบบโปรโตคอลระดับการติดต่อสื่อสาร

การติดต่อระหว่างโฮส กับเทอร์มินอล สามารถติดต่อกันได้ 2 วิธี คือ การติดต่อโดยใช้ไฟล์ และการติดต่อโดยใช้โมเด็ม

3.1.1 การติดต่อโดยใช้ไฟล์

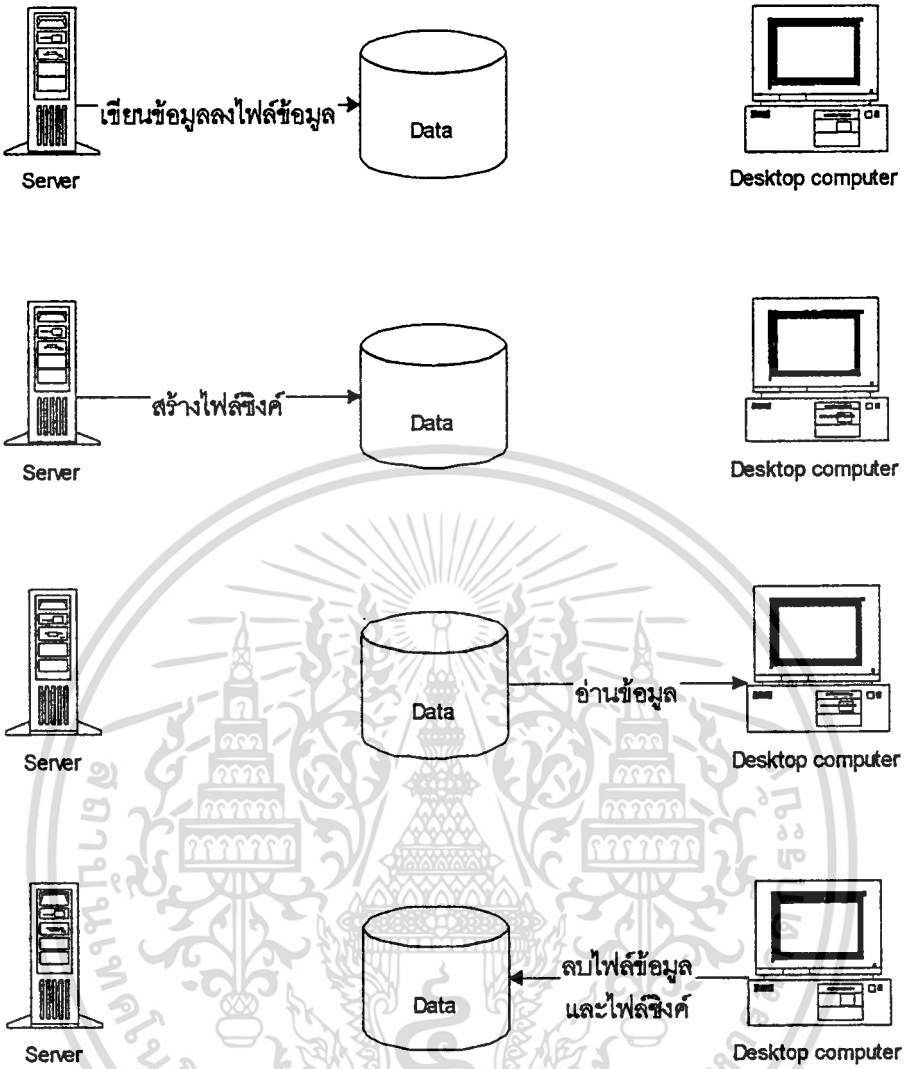
การติดต่อโดยใช้ไฟล์ เพื่อให้สามารถใช้โปรแกรมบนระบบเน็ตเวิร์ก เช่น ระบบแลน (LAN) ได้ โดยโฮสและเทอร์มินัล จะเขียน, อ่าน และลบไฟล์ในไดเรกทอรีเดียวกัน

3.1.1.1 การติดต่อ

การติดต่อแบบนี้ จะใช้ไฟล์ทั้งหมด 4 ไฟล์ คือ ไฟล์โฮส, ไฟล์เทอร์มินัล, ไฟล์ซิงค์ของโฮส และไฟล์ซิงค์ของเทอร์มินัล โดยการส่งข้อมูลจากโฮสไปยังเทอร์มินัล จะใช้ไฟล์โฮส และไฟล์ซิงค์ของโฮส และการส่งข้อมูลจากเทอร์มินัลไปยังโฮส ก็ใช้อีกสองไฟล์ที่เหลือ

การส่งข้อมูลทั้งสองแบบ จะเริ่มจากฝ่ายที่ต้องการส่งข้อมูล เขียนไฟล์ข้อมูลก่อน ซึ่งก็คือไฟล์โฮส สำหรับการส่งข้อมูลจากโฮส และไฟล์เทอร์มินัล สำหรับการส่งข้อมูลจากเทอร์มินัล เมื่อเขียนข้อมูลสู่ไฟล์ข้อมูลเสร็จแล้ว ก็สร้างไฟล์ซิงค์ (ไฟล์ซิงค์ของโฮสสำหรับการส่งข้อมูลจากโฮส และไฟล์ซิงค์ของเทอร์มินัลสำหรับการส่งข้อมูลจากเทอร์มินัล) ก็เสร็จการส่งข้อมูลของฝ่ายส่ง

สำหรับฝ่ายรับข้อมูล ก็จะรอไฟล์ซิงค์ ซึ่งฝ่ายส่งจะสร้างเมื่อเขียนข้อมูลเสร็จเรียบร้อยแล้ว เมื่อฝ่ายรับพบไฟล์ซิงค์ ก็จะอ่านข้อมูลจากไฟล์ข้อมูลขึ้นมา จากนั้นก็จะลบไฟล์ข้อมูล และไฟล์ซิงค์ เพื่อเตรียมพร้อมสำหรับการส่งข้อมูลครั้งต่อไป



รูปที่ 3.1 การติดต่อโดยใช้ไฟล์

3.1.1.2 การเริ่มต้นการติดต่อ

โฮสส่ง

4 ไบต์ เลขรหัส GRIPS_ID

2 ไบต์ GRIPS_WAITLOGIN

2 ไบต์ หมายเลขเวอร์ชันของโฮส

2 ไบต์ ความยาวชื่อไฟล์เทอร์มินัล

2 ไบต์ ความยาวชื่อไฟล์ซิงค์ของโฮส

2 ไบต์ ความยาวชื่อไฟล์ซิงค์ของเทอร์มินัล

n ไบต์ ชื่อไฟล์เทอร์มินัล

n ไบต์ ชื่อไฟล์ซิงค์ของโฮส

n ไบต์ ชื่อไฟล์ซิงค์ของเทอร์มินัล

เทอร์มินัลส่ง

4 ไบต์ เลขรหัส

2 ไบต์ GRIP_LOGIN

3.1.1.3 การจบการติดต่อ

การจบการติดต่อ สามารถทำได้ทันที หลังจากที่มีการส่งข้อมูลชุดสุดท้าย โดยปกติแล้ว ถ้าฝ่ายรับข้อมูลครั้งสุดท้าย ไม่ส่งข้อมูลมาอีก ก็ถือเป็นการจบการติดต่อ

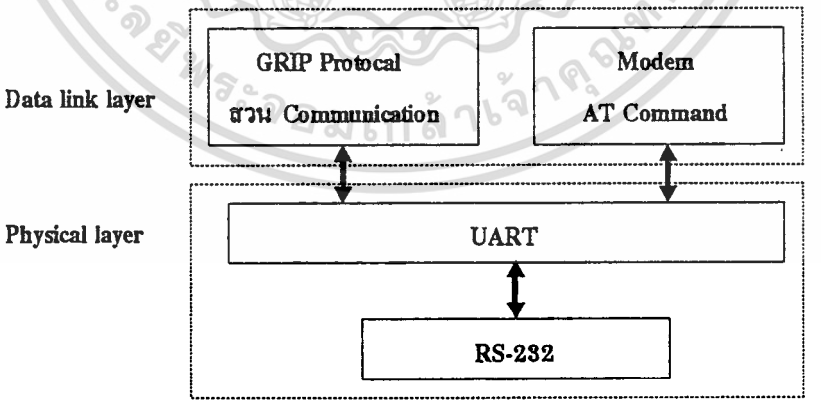
3.1.1.4 การแก้ไขข้อผิดพลาด

ข้อผิดพลาดที่อาจเกิดขึ้นได้ ในระหว่างการรับ/ส่งข้อมูล ผ่านทางไฟล์ ก็คือ ไฟล์โดนโปรแกรมอื่นลบ ซึ่งอาจเกิดขึ้นได้ในระบบแลน สำหรับข้อผิดพลาดประเภทข้อมูลที่เขียนลงไฟล์ผิดพลาด หรืออ่านข้อมูลขึ้นมาผิดพลาด มีโอกาสเกิดขึ้นได้น้อยมาก เพราะระบบแลนจะดูแลจุดนี้ให้

อย่างไรก็ตาม ในการรับ/ส่งข้อมูลผ่านทางไฟล์ ยังไม่มีการแก้ไขข้อผิดพลาดใดๆ ทั้งสิ้น เพราะโอกาสเกิดข้อผิดพลาดดังได้กล่าวมาแล้ว จะมีน้อยมาก

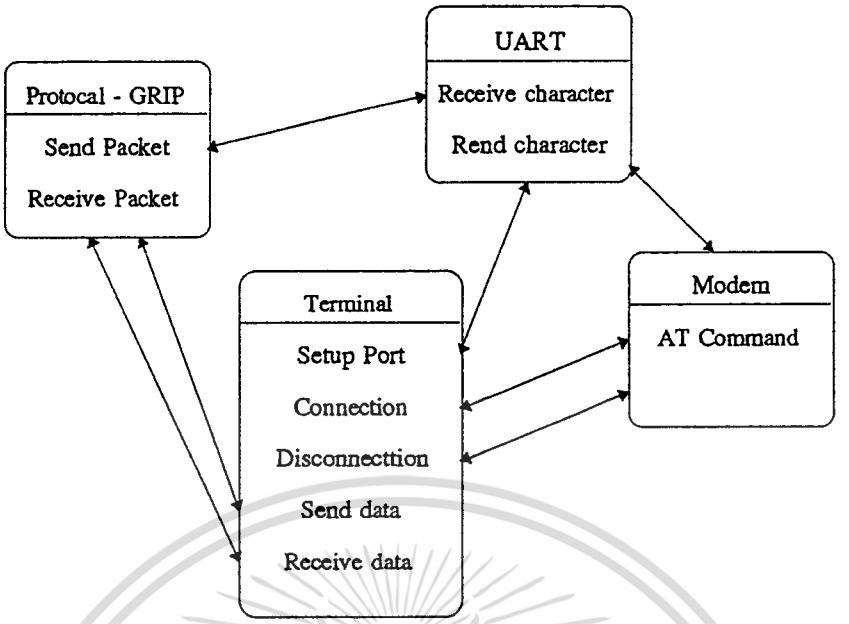
3.1.2 ส่วนประกอบของส่วนติดต่อสื่อสาร

ในส่วนนี้ทั้งหมดทำหน้าที่เกี่ยวกับการรับส่งข้อมูลผ่านพอร์ตอนุกรม และการสื่อสารโดยใช้โมเด็ม รวมทั้งการกำหนดโปรโตคอลที่ใช้ในการติดต่อสื่อสารด้วย สามารถแบ่งออกเป็นส่วนๆ ได้ดังต่อไปนี้



รูปที่ 3.2 ส่วนประกอบของ Communication Part

ถ้าแบ่งตามการทำงานจะได้ดังรูปต่อไป



รูปที่ 3.3 การทำงานและความสัมพันธ์ของส่วนประกอบต่างๆ ในส่วนติดต่อสื่อสาร

3.1.2.1 การออกแบบในแต่ละส่วน

เป็นการออกแบบในส่วนต่างๆ ของโปรแกรม คือ ส่วนที่ใช้ในการติดต่อโมเด็ม, ส่วนที่ใช้ในการติอต่อกับพอร์ตอนุกรม และส่วนเทอร์มินัล

3.1.2.1.1 ส่วนที่ใช้ในการติดต่อโมเด็ม

ทำหน้าที่เกี่ยวกับการส่งคำสั่งที่ใช้กับโมเด็ม เช่น การกำหนดค่าเริ่มต้น, การหมุนโทรศัพท์, การยกเลิกการติดต่อ

3.1.2.1.1.1 การกำหนดค่าเริ่มต้นให้โมเด็มก่อนใช้งาน

ก่อนการใช้โมเด็มทุกครั้งควรที่จะกำหนดค่าเริ่มต้นก่อนการใช้งาน โมเด็มที่ผลิตจากผู้ผลิตต่างๆ จะมีการกำหนดค่าเริ่มต้นไม่เหมือนกัน ดังนั้นจะต้องศึกษาจากคู่มือเสียก่อน แต่มีวิธีง่ายที่สุดในการกำหนดค่าเริ่มต้นคือ ใช้ค่าที่ทางผู้ผลิตได้กำหนดไว้มาจากโรงงานแล้ว โดยการใช้คำสั่ง ATZ อีกทั้งคำสั่งนี้ยังสามารถใช้ได้ในกรณีที่มีการเปลี่ยนแปลงค่าภายในโมเด็มไปจากเดิม หรือ ไมทราบสถานะของโมเด็ม สามารถใช้คำสั่งนี้ในการกำหนดให้ค่าของโมเด็มกลับมาเหมือนเมื่อตอนเปิดเครื่องได้ หลังจากส่งคำสั่งไปแล้วก็รอให้โมเด็มตอบรับว่า "OK" ในโมเด็มรุ่นใหม่ๆ จะสามารถรับและส่งแฟกซ์ได้ด้วย ซึ่งจะมีปัญหาในการส่งข้อมูล การรับและส่งแฟกซ์จะมีการควบคุมการไหลของข้อมูลโดยการใช้ฮาร์ดแวร์ (Hardware Flow Control) เราจำเป็นต้องกำหนดให้โมเด็มไม่ใช้การควบคุมการไหลของข้อมูลโดยการใช้ฮาร์ดแวร์นี้ โดยดูจากคู่มือของโมเด็มแต่ละชนิดว่าจะต้องใช้คำสั่งใดในการยกเลิก

3.1.2.1.1.2 การหมุนโทรศัพท์

ในการหมุนโทรศัพท์จะใช้คำสั่ง ATD และตามด้วย T หรือ P ขึ้นอยู่กับว่าโทรศัพท์เป็นระบบโทน หรือระบบพัลส์ หลังจากนั้นตามด้วยหมายเลขโทรศัพท์ ถ้าในกรณีที่ต้องผ่าน PABX เราจะใช้เครื่องหมายจุลภาคคั่นระหว่างหมายเลขโทรศัพท์ภายในกับศัพท์ภายนอก เพื่อบอกให้โมเด็มรู้ว่าหลังจากที่หมุนหมายเลขโทรศัพท์ภายในแล้วให้หยุดรอสักครู่หนึ่งแล้วจึงค่อยหมุนหมายเลขโทรศัพท์ภายนอก หลังจากนั้นรอตอบรับจากโมเด็ม ถ้าโมเด็มตอบรับว่า "CONNECT" แปลว่าตอนนี้เราสามารถติดต่อกับด้านปลายทางได้แล้ว ถ้าโมเด็มตอบอย่างอื่นแปลว่ายังไม่สามารถติดต่อกับด้านปลายทางได้ จะต้องตรวจสอบว่าเกิดขึ้นจากอะไร เช่น สายไม่ว่าง เป็นต้น หลังจากที่สามารถติดต่อกับด้านปลายทางได้แล้ว จะใช้การติดต่อโดยใช้โปรโตคอลแทน

3.1.2.1.1.3 การยกเลิกการติดต่อ

หลังจากที่ได้รับคำสั่งจากส่วนบนให้ยกเลิกการติดต่อ สามารถทำได้สองวิธีด้วยกัน วิธีแรกใช้คำสั่งในการยกเลิกการติดต่อ อีกวิธีหนึ่งคือใช้ฮาร์ดแวร

3.1.2.1.1.3.1 การใช้คำสั่งในการยกเลิกการติดต่อ

เนื่องจากตอนนี้โมเด็มอยู่ในโหมดของการส่งข้อมูล จะต้องให้โมเด็มกลับมาอยู่ในโหมดคำสั่งเสียก่อน จึงจะสามารถใช้คำสั่งในการยกเลิกการติดต่อได้ โดยการส่งเครื่องบวกติดต่อกันสามตัวแล้วรอสักครู่ โมเด็มจะเปลี่ยนสถานะจากโหมดการส่งข้อมูลมาเป็นโหมดคำสั่ง ด้วยการตอบรับว่า "OK" หลังจากนั้นให้ส่งคำสั่งยกเลิกการติดต่อโดยใช้คำสั่ง ATH0 เป็นการบอกให้โมเด็มวางหูโทรศัพท์ โมเด็มจะตอบรับว่า "OK" ถ้าสามารถยกเลิกการติดต่อได้สำเร็จ การยกเลิกการติดต่อโดยใช้คำสั่งนี้ในบางครั้งก็ไม่ได้ผล ต้องใช้ฮาร์ดแวร์แทน

3.1.2.1.1.3.2 การใช้ฮาร์ดแวร์ในการยกเลิกการติดต่อ

ใน UART จะมีพอร์ตที่เกี่ยวกับการควบคุมโมเด็ม (Modem Control Register) ซึ่งจะมีบิตหนึ่งเป็นตัวบอกว่าตอนนี้ทางด้านเทอร์มินัลพร้อมที่จะติดต่อ หรือ บิต DTR ถ้ากำหนดค่าให้เป็น 0 จะทำให้โมเด็มวางหูโทรศัพท์ (ในตอนเริ่มต้นจะต้องกำหนดค่าบิต DTR นี้ให้เป็น 1 จึงจะสามารถติดต่อกับโมเด็มได้)

3.1.2.1.2 ส่วนที่ใช้ในการติอต่อกับพอร์ตอนุกรม

ทำหน้าที่ในการรับและส่งตัวอักษรไปยังพอร์ตอนุกรมเพื่อส่งต่อไปยังโมเด็ม การรับและส่งมีด้วยกันสองวิธี คือ แบบอินเทอร์รัพต์และ แบบโพลลิง (Polling)

3.1.2.1.2.1 การติดต่อพอร์ตอนุกรมแบบโพลลิง

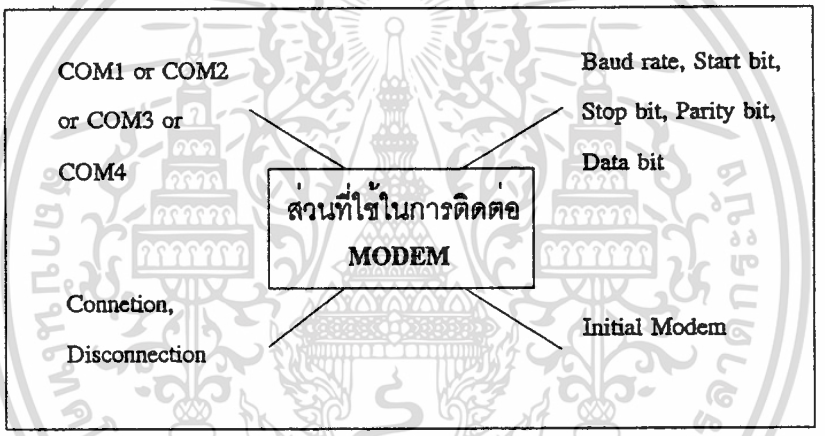
การติดต่อพอร์ตอนุกรมแบบโพลลิงคือ การที่จะต้องค่อยไปตรวจสอบพอร์ตอนุกรมว่ามีข้อมูลเข้ามาหรือไม่ เมื่อโปรแกรมทำงานอย่างอื่นอยู่ ถ้ากลับมาอ่านข้อมูลจากพอร์ตอนุกรมไม่ทัน ก็จะทำให้ข้อมูลก่อนหน้านั้นถูกทับด้วยข้อมูลใหม่ ดังนั้นจึงต้องระวังว่า จะต้องกลับมาอ่านข้อมูลจากพอร์ตอนุกรมให้ทัน

3.1.2.1.2.2 การติดต่อพอร์ตอนุกรมแบบอินเทอร์รัพต์

ในการรับส่งด้วยความเร็วสูง วิธีแบบโพลลิงไม่สามารถใช้ได้ เพราะไม่สามารถอ่านข้อมูลได้ทัน วิธีแบบอินเทอร์รัพต์จะได้เปรียบมากกว่า เพราะไม่ต้องกังวลว่าจะรับข้อมูลได้ไม่ครบ สามารถทำงานในส่วนอื่นได้โดยไม่จำกัดเวลาในการทำงาน เมื่อมีข้อมูลเข้ามาจะเกิดอินเทอร์รัพต์ขึ้น และจะนำข้อมูลไปเก็บไว้ในบัฟเฟอร์ โดยบัฟเฟอร์ที่ใช้จะเป็นแบบวงแหวน เพื่อการประหยัดหน่วยความจำ เมื่อต้องการใช้ข้อมูลจะอ่านจากบัฟเฟอร์แทนที่จะอ่านจากพอร์ตอนุกรมโดยตรง

3.1.2.1.3 ส่วนเทอร์มินัล

ทำหน้าที่เกี่ยวกับการติดต่อ, การยกเลิกการติดต่อ, การติดตั้งฮาร์ดแวร์ เช่น ขนาดความเร็วบิต, จำนวนบิตเริ่ม, จำนวนบิตหยุด, ชนิดของพาริตี้, ขนาดของข้อมูล เป็นต้น



รูปที่ 3.4 ฟังก์ชันในส่วนที่ใช้ในการติดต่อโมเด็ม

3.1.2.2 Graphical Remote Information Protocal (GRIP) ส่วนติดต่อสื่อสาร

เมื่อสามารถติดต่อสื่อการกันได้แล้วในการส่งข้อมูลระหว่างโฮส์และเทอร์มมินัลจะต้องอาศัยโปรโตคอลช่วยในการส่งข้อมูล โปรโตคอลที่ออกแบบได้แนวความคดมาจาก Zmodem ซึ่งเป็นโปรโตคอลแบบการติดต่อแบบกึ่งสมบูรณ์ มีรูปร่างหน้าตาดังต่อไปนี้

GPAD	GPAD	GDLE	GTYPE	OFFSET1	OFFSET2	CRC16

รูปที่ 3.5 รายละเอียดแต่ละฟิลด์ในส่วนหัวของแพ็คเกจเฟรม

DATA	CRC16

รูปที่ 3.6 Data Packet ของ GRIP

GPAD	ตัวอักษรที่เป็นตัวบอกจุดเริ่มต้นของแพ็คเกจ
GDLE	ตัวอักษรดาต้าลิงค์เอสเคบ (Data link escape)
GTYPE	ชนิดของแพ็คเกจ
OFFSET1-2	ขึ้นอยู่กับชนิดของแพ็คเกจ
CRC16	ส่วนตรวจสอบความผิดพลาดในการส่ง (Error Detection)

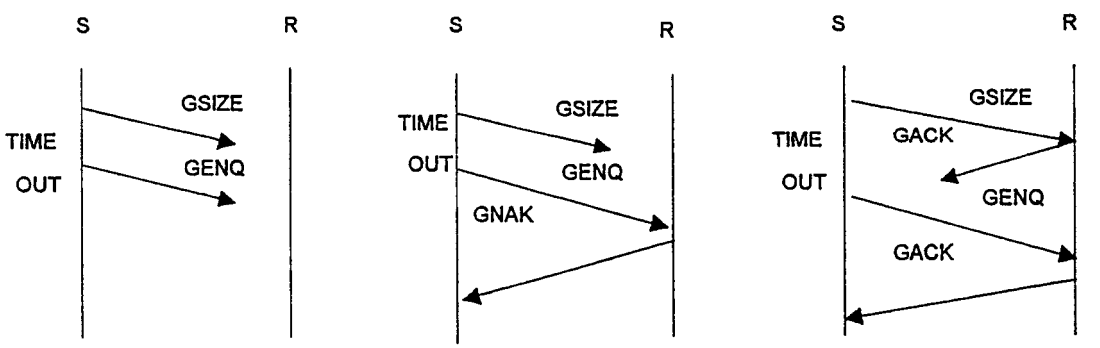
3.1.2.3 ชนิดของแพ็คเกจ

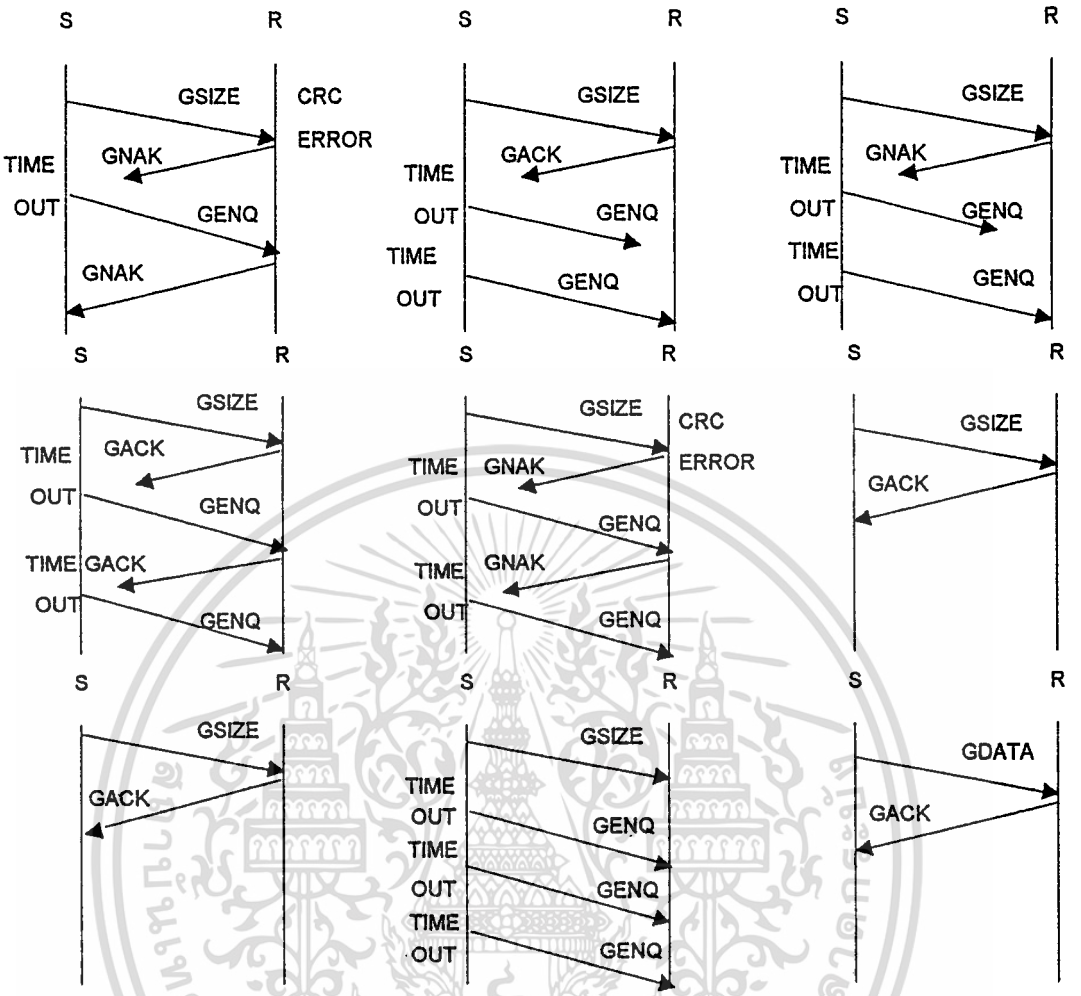
GSIZE	บอกขนาดของข้อมูลที่จะส่งทั้งหมด
GDATA	บอกขนาดของข้อมูลที่จะส่งต่อหนึ่งแพ็คเกจ
GACK	ตอบรับเมื่อได้รับข้อมูลถูกต้อง
GNAK	ตอบรับเมื่อได้รับข้อมูลไม่ถูกต้อง
GENQ	ร้องขอการตอบรับครั้งสุดท้าย
GFIN	ขอจบการติดต่อ

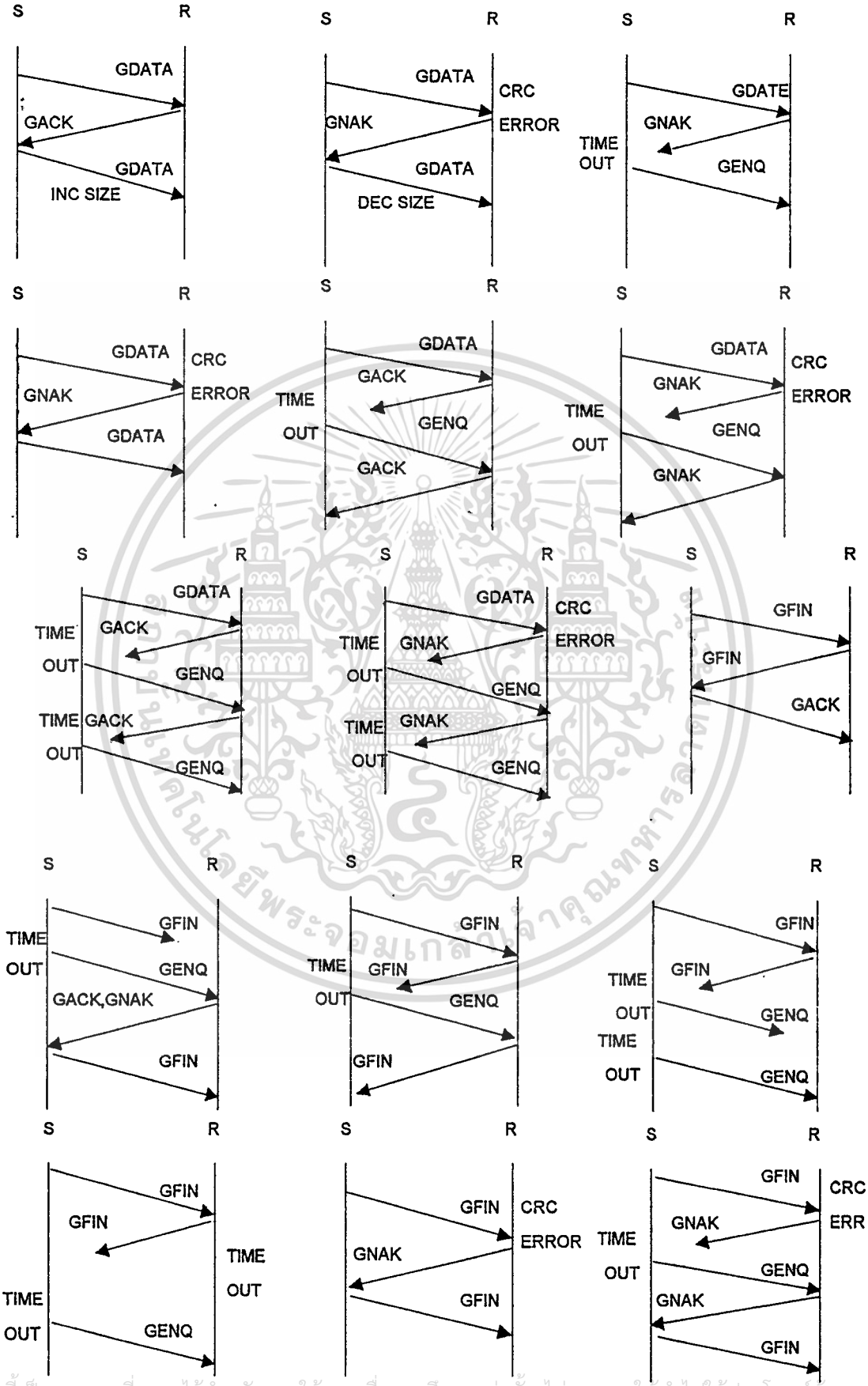
3.1.2.4 ข้อดีของโปรโตคอล GRIP

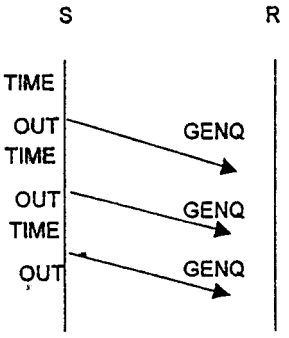
- สามารถตรวจสอบการผิดพลาดที่เกิดขึ้นในการส่งข้อมูลได้เกือบ 100 เปอร์เซ็นต์ เพราะใช้ซีอาร์ซีในการตรวจสอบ และใช้วิธีการส่งซ้ำในการเรียกคืนข้อมูลที่ผิดพลาด
- มีการลดและเพิ่มขนาดของแพ็คเกจ ในกรณีที่เกิดการผิดพลาดในการส่ง ทำให้การส่งข้อมูลผิดพลาดลดลง ถ้าขนาดของข้อมูลที่จะส่งมีขนาดเล็ก หรือ ลดเวลาในการส่งเมื่อสภาพการส่งดี ไม่เกิดการผิดพลาดในการส่ง
- สามารถลดผลกระทบจากสัญญาณรบกวนได้บางส่วน เช่น มีสัญญาณรบกวนในช่วงของการรอรับการตอบรับจากอีกด้านหนึ่ง
- สามารถตรวจสอบการชนกันของข้อมูลที่เกิดจากการส่งทั้งสองด้าน

3.1.2.5 การควบคุมการไหล (Flow control) ของแพ็คเกจ









3.2 การออกแบบโปรโตคอลระดับแอพพลิเคชัน

โปรโตคอลระดับแอพพลิเคชัน เป็นส่วนที่ใช้ติดต่อกันระหว่างโปรแกรมโฮส และโปรแกรมเทอร์มินัล โดยจะไม่ขึ้นกับว่า โปรโตคอลระดับการติดต่อสื่อสารจะเป็นแบบใด

3.2.1 โครงสร้างของโปรโตคอล

โปรโตคอลระดับแอพพลิเคชัน แบ่งเป็น 2 ประเภท คือ ใช้สำหรับควบคุมการทำงานของโปรแกรม เช่น การล็อกอิน (log in), การส่งการตอบสนองของผู้ใช้ เป็นต้น และใช้สำหรับส่งข้อมูล

3.2.2 โปรโตคอลสำหรับการควบคุม

โปรโตคอลสำหรับการควบคุมการทำงานของโปรแกรม จะมีใช้ 3 อย่าง คือ การล็อกอิน, การส่งการตอบสนองของผู้ใช้ไปยังโฮส และการล็อกเอาต์ (log out)

3.2.2.1 การล็อกอิน

โฮสส่ง

2 ไบต์ GRIPS_WAITLOGIN	
2 ไบต์ wVer	บอกรุ่นของเทอร์มินัลที่ต้องการ
2 ไบต์ wLen	ความยาวของชื่อแอพพลิเคชัน
n ไบต์ cServInfo[wLen]	ชื่อแอพพลิเคชัน

เทอร์มินัลส่ง

2 ไบต์ GRIPS_LOGIN	
2 ไบต์ wBitMapVer	รุ่นของบิตแมพที่มีอยู่
2 ไบต์ wNLen	ความยาวของชื่อผู้ใช้
2 ไบต์ wPLen	ความยาวของรหัสผ่าน
n ไบต์ cName[wNLen]	ชื่อผู้ใช้

n ไบต์ cPwd[wPLen] รหัสผ่าน

ถ้าโฮสต้องการเพิ่มเติมบิตแมพ

โฮสส่ง

2 ไบต์ GRIPS_UPDATEBITMAP

เทอร์มินัลส่ง

2 ไบต์ GRIPS_ACKUPDATE

โฮสส่ง

2 ไบต์ GRIPS_SENDBITMAP

2 ไบต์ wNLen ความยาวชื่อไฟล์

2 ไบต์ wFLen ความยาวไฟล์

n ไบต์ cName[wNLen] ชื่อไฟล์

n ไบต์ cBitMap บิตแมพ

หรือ

2 ไบต์ GRIPS_REPLACEBITMAP

2 ไบต์ wNLen ความยาวชื่อไฟล์

2 ไบต์ wFLen ความยาวไฟล์

n ไบต์ cName[wNLen] ชื่อไฟล์

n ไบต์ cBitMap บิตแมพ

หรือ

2 ไบต์ GRIPS_DELBITMAP

2 ไบต์ wNLen ความยาวชื่อไฟล์

n ไบต์ cName[wNLen] ชื่อไฟล์

เทอร์มินัลส่ง

2 ไบต์ GRIPS_ACKBITMAP

ถ้าโฮสต้องการส่งไฟล์บิตแมพอีกก็จะส่ง GRIPS_SENDBITMAP มาอีก

ถ้าโฮสส่งไฟล์บิตแมพเสร็จแล้ว หรือโฮสไม่ต้องการเพิ่มเติมบิตแมพ

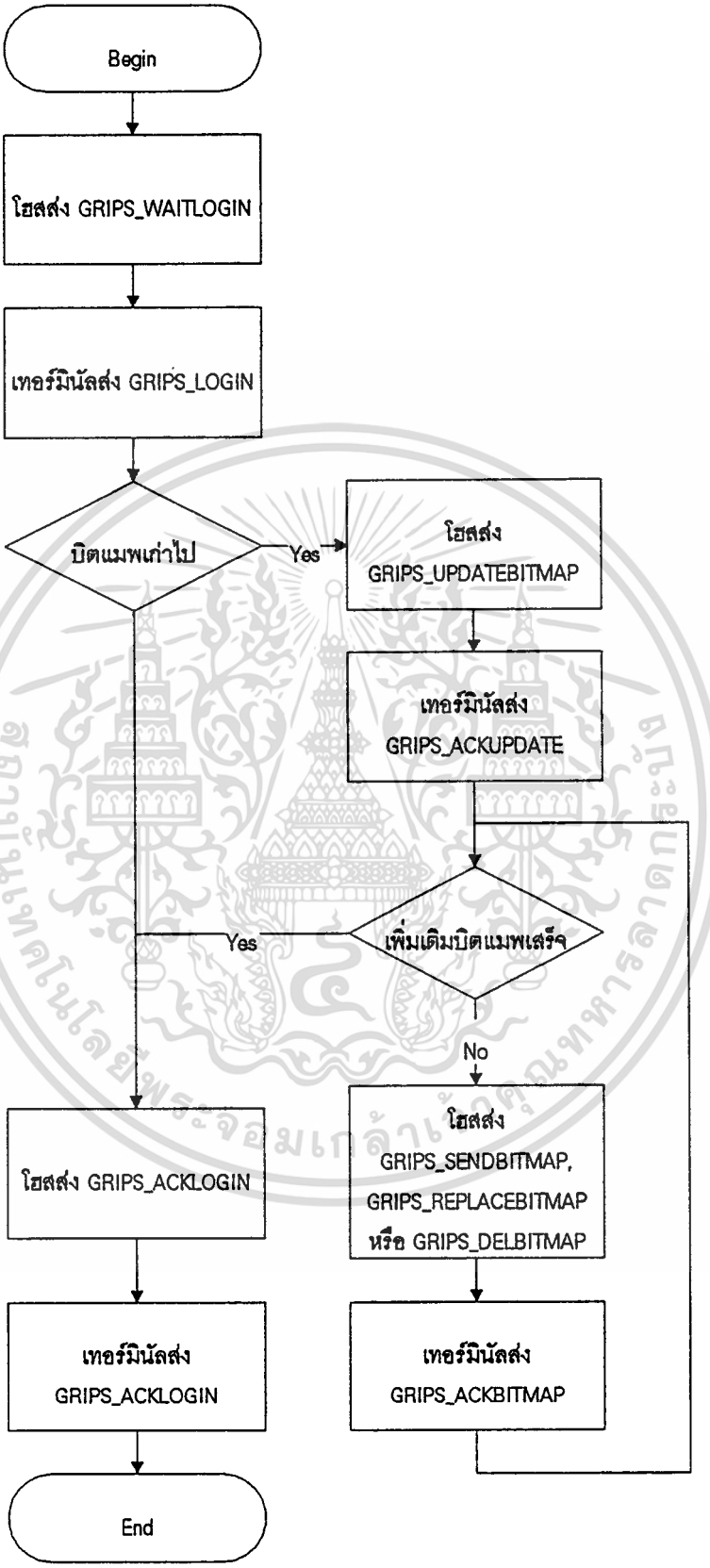
โฮสส่ง

2 ไบต์ GRIPS_ACKLOGIN

เทอร์มินัลส่ง

2 ไบต์ GRIPS_ACKLOGIN

โฮสส่งหน้าจอแรก



รูปที่ 3.7 การล็อกอิน

3.2.2.2 การตอบสนองของผู้ใช้ไปยังโฮส

เทอร์มินัลส่ง

2 ไบต์ GRIPS_RETURN

2 ไบต์ wLen ความยาวของชื่อไฟล์

n ไบต์ cName[wLen] ชื่อไฟล์ที่ต้องการ

โฮสส่ง

ไฟล์ที่เทอร์มินัลต้องการ

3.2.2.3 การล็อกเอาต์

การล็อกเอาต์ จะมี 2 วิธี คือ โฮสขอล็อกเอาต์ ในกรณีที่จบโปรแกรม และเทอร์มินัลของล็อกเอาต์ ในกรณีที่ผู้ใช้ต้องการเลิกการทำงาน

3.2.2.3.1 การล็อกเอาต์โดยโฮส

โฮสส่ง

2 ไบต์ GRIPS_EXIT

เทอร์มินัลส่ง

2 ไบต์ GRIPS_EXIT_ACK

3.2.2.3.2 การล็อกเอาต์โดยเทอร์มินัล

เทอร์มินัลส่ง

2 ไบต์ GRIPS_EXIT_REQ

โฮสส่ง

2 ไบต์ GRIPS_EXIT

เทอร์มินัลส่ง

2 ไบต์ GRIPS_EXIT_ACK

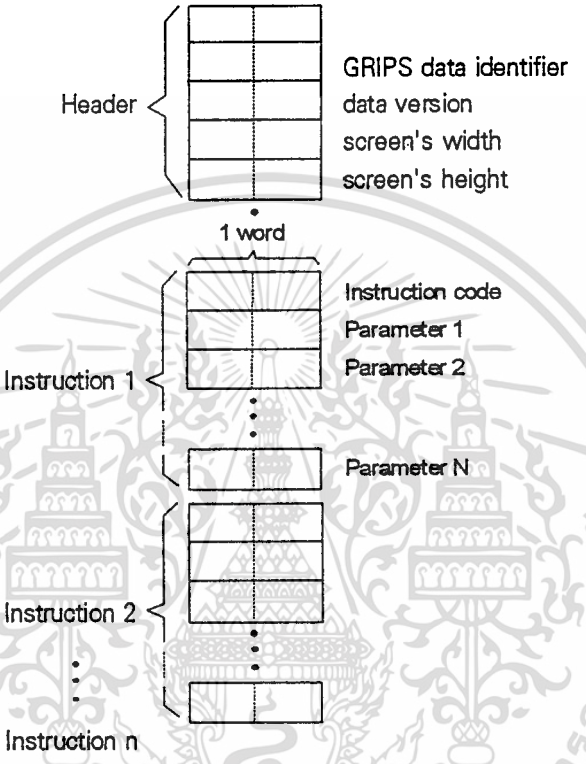
3.2.3 โปรโตคอลสำหรับส่งข้อมูล

โปรโตคอลสำหรับส่งข้อมูล จะประกอบไปด้วย 2 ส่วน คือ ส่วนหัว (header) และส่วนข้อมูล (data) โดยส่วนหัว จะเป็นข้อมูลที่ใช้เพียงครั้งเดียว เช่น ความกว้าง ความยาวของจอภาพ ในขณะส่วนข้อมูล ก็จะเป็นกราฟิกต่างๆ

3.2.3.1 ส่วนหัว

โครงสร้างส่วนหัว จะประกอบไปด้วยฟิลด์ต่างๆ ดังนี้

2 ไบต์ เลขรหัส	เป็นการบอกว่าเป็นข้อมูลของโปรแกรม GRIPS
2 ไบต์ wVer	บอกรุ่นของข้อมูล
2 ไบต์ wWidth	ความกว้างของหน้าจอ
2 ไบต์ wHeight	ความสูงของหน้าจอ



รูปที่ 3.8 โครงสร้างของข้อมูล GRIPS

3.2.3.2 ส่วนข้อมูล

ส่วนข้อมูลประกอบไปด้วยคำสั่งของ GRIPS หลายๆ คำสั่งเรียงต่อกัน โดยแต่ละคำสั่งจะมีความยาวเป็นจำนวนคู่เสมอ ส่วนนี้จะใช้สำหรับสร้างแต่ละหน้าจอของโปรแกรม คำสั่งในส่วนนี้จะทำงานตามลำดับเสมอ นั่นคือ กราฟิกที่วาดที่หลังจะทับกราฟิกที่วาดก่อน ดังนั้นการเรียงลำดับของคำสั่งจะมีผลต่อผลลัพธ์ด้วย

3.2.4 คำสั่ง GRIPS

คำสั่งกราฟิกของ GRIPS จะประกอบไปด้วยรหัสคำสั่งยาว 2 ไบต์ และพารามิเตอร์ของแต่ละคำสั่ง ซึ่งจะยาวไม่เท่ากัน แต่จะเป็นจำนวนคู่เสมอ การทำงานของเทอร์มินัล จะเก็บตำแหน่ง, สีวาด, สีระบาย, ลักษณะการระบาย และรูปแบบ, ขนาด และลักษณะของตัวอักษร เอาไว้ เพราะคำสั่งส่วนใหญ่ จะมีการใช้ค่าเหล่านี้ด้วย เช่น คำสั่งลากเส้นจากจุดปัจจุบันไปยังจุดที่กำหนด เป็นต้น

3.2.4.1 GRIPS_TEXTOUT

แสดงผลข้อความที่ตำแหน่งที่กำหนด ตามรูปแบบ, ลักษณะ, ขนาด และสีที่กำหนดไว้

รูปแบบคำสั่ง

2 ไบต์ GRIPS_TEXTOUT	
2 ไบต์ wX	ตำแหน่งในแนวแกนนอน
2 ไบต์ wY	ตำแหน่งในแนวแกนตั้ง
2 ไบต์ wLen	ความยาวของข้อความ
n ไบต์ cStr[wLen]	ข้อความ ถ้ามีความยาวเป็นคี่ ให้เพิ่มอักขระ NULL ต่อท้าย

การทำงาน

แสดงข้อความที่จุดที่กำหนด โดยใช้รูปแบบ, ขนาดตัวอักษร และลักษณะตัวอักษรปัจจุบัน (ที่กำหนดครั้งสุดท้าย ถ้าไม่กำหนดจะใช้รูปแบบ ไทมส์นิวโรมัน (Times New Roman) ขนาด 10 ปอยท์ ลักษณะตัวอักษรปกติ) ข้อความที่มีความยาวเป็นจำนวนคี่ จะต้องเติมอักขระ NULL ต่อท้าย 1 ตัว เพื่อให้ความยาวเป็นคู่ โดยข้อมูลตัวนี้จะไม่นำไปใช้งานใดๆ

หมายเหตุ

ตำแหน่งที่กำหนด จะเป็นตำแหน่งมุมบนซ้ายของข้อความ

คำสั่งที่เกี่ยวข้อง

GRIPS_SETFONTSIZE

GRIPS_SETFONTSTYLE

GRIPS_SETFONTTYPE

3.2.4.2 GRIPS_RECTANGLE

วาดรูปสี่เหลี่ยมมุมฉากด้วยตำแหน่ง และขนาดที่กำหนด

รูปแบบคำสั่ง

2 ไบต์ GRIPS_RECTANGLE	
2 ไบต์ wX	ตำแหน่งในแนวแแกนนอนของมุมบนซ้าย
2 ไบต์ wY	ตำแหน่งในแนวแแกนตั้งของมุมบนซ้าย
2 ไบต์ wSX	ขนาดในแนวแกนนอน
2 ไบต์ wSY	ขนาดในแนวแกนตั้ง

การทำงาน

วาดรูปสี่เหลี่ยมมุมฉาก โดยด้านของสี่เหลี่ยมจะอยู่ในแนวแกนตั้ง และแกนนอน สำหรับจุดที่กำหนด จะเป็นจุดมุมบนซ้าย และจะวาดออกไปทางด้านขวา และด้านล่าง เท่ากับขนาดที่กำหนด ในการวาดจะใช้สี และการระบายสีภายในตามค่าปัจจุบัน

หมายเหตุ

จุดปัจจุบันของการวาดรูปคือจุด wX+wSX, wY+wSY

คำสั่งที่เกี่ยวข้อง

GRIPS_SETCOLOR

GRIPS_SETFILLCOLOR

GRIPS_SETFILLSTYLE

3.2.4.3 GRIPS_LINETO

ลากเส้นจากจุดปัจจุบันไปจุดที่กำหนด

รูปแบบคำสั่ง

2 ไบต์ GRIPS_LINETO

2 ไบต์ wX ตำแหน่งในแนวแกนนอน

2 ไบต์ wY ตำแหน่งในแนวแกนตั้ง

การทำงาน

ลากเส้นจากจุดปัจจุบันไปจุดที่กำหนด โดยใช้สีปัจจุบัน

หมายเหตุ

จุดปัจจุบันของการวาดรูปคือจุด wX, wY

คำสั่งที่เกี่ยวข้อง

GRIPS_MOVETO

GRIPS_SETCOLOR

3.2.4.4 GRIPS_MOVETO

กำหนดจุดปัจจุบันใหม่

รูปแบบคำสั่ง

2 ไบต์ GRIPS_MOVETO

2 ไบต์ wX ตำแหน่งในแนวแกนนอน

2 ไบต์ wY ตำแหน่งในแนวแกนตั้ง

การทำงาน

ย้ายตำแหน่งจุดปัจจุบันไปยังจุดที่กำหนด

คำสั่งที่เกี่ยวข้อง

GRIPS_LINETO

3.2.4.5 GRIPS_SETCOLOR

กำหนดสีที่ใช้วาด

รูปแบบคำสั่ง

2 ไบต์ GRIPS_SETCOLOR

4 ไบต์ dwColor สี

การทำงาน

กำหนดสีปัจจุบัน โดยค่า dwColor จะอยู่ในรูป B*0x10000+G*0x100+R โดยค่า B, G และ R แทนสีน้ำเงิน, เขียว และแดงตามลำดับ และมีค่าตั้งแต่ 0 (ไม่มีสีนั้นๆ เลย) ถึง 255 (สีนั้นๆ สว่างเต็มที่) หรืออาจจะเขียนในรูป 0x00bbggrr โดย bb, gg, rr มีค่าตั้งแต่ 0x00 ถึง 0xff ก็ได้ เช่น 0x00ff00ff จะหมายถึงสีม่วงมาเจนตา (Magenta)

คำสั่งที่เกี่ยวข้อง

GRIPS_SETFILLCOLOR

3.2.4.6 GRIPS_SETFILLCOLOR

กำหนดสีที่ใช้ระบาย

รูปแบบคำสั่ง

2 ไบต์ GRIPS_SETFILLCOLOR

4 ไบต์ dwColor สี

การทำงาน

กำหนดสีที่ใช้ระบาย โดยค่า dwColor เหมือนกับค่า dwColor ในคำสั่ง GRIPS_SETCOLOR

คำสั่งที่เกี่ยวข้อง

GRIPS_SETCOLOR

3.2.4.7 GRIPS_POLYLINE

วาดรูปหลายเหลี่ยมโดยไม่ระบายสี

รูปแบบคำสั่ง

2 ไบต์ GRIPS_POLYLINE

2 ไบต์ wCount จำนวนจุดของรูป

n ไบต์ gpntPoint[wCount] อะเรย์ของคู่ลำดับแทนจุดมุมของรูปหลายเหลี่ยม

โดยที่คู่ลำดับใน gpntPoint[wCount] มีรูปแบบดังนี้

2 ไบต์ wX ตำแหน่งในแนวแกน X

2 ไบต์ wY ตำแหน่งในแนวแกน Y

การทำงาน

วาดรูปหลายเหลี่ยมโดยไม่ระบายสีภายในรูป โดยการวาดรูปหลายเหลี่ยม จะลากเส้นตรงจากจุดที่ 1 ไปจุดที่ 2 จากจุดที่ 2 ไปจุดที่ 3 ไปเรื่อยๆ จนถึงจุดสุดท้าย โดยไม่ทำการปิดรูปให้ ถ้าต้องการปิดรูป จะต้องกำหนดให้จุดสุดท้ายทับจุดแรก สำหรับสีที่ใช้วาดจะใช้สีปัจจุบัน

สำหรับคู่ลำดับแทนจุดมุมของรูปหลายเหลี่ยมนั้น เพื่อความสะดวกสามารถเขียนได้เป็น

((wX1, wY1), (wX2, wY2), (wX3, wY3), ...)

หมายเหตุ

ตำแหน่งปัจจุบันคือจุดสุดท้ายในอาเรย์

คำสั่งที่เกี่ยวข้อง

GRIPS_POLYGON

GRIPS_SETCOLOR

3.2.4.8 GRIPS_POLYGON

วาดรูปหลายเหลี่ยมโดยระบายสีข้างในด้วย

รูปแบบคำสั่ง

2 ไบต์ GRIPS_POLYGON

2 ไบต์ wCount จำนวนจุดของรูป

n ไบต์ gpntPoint[wCount] อะเรย์ของคู่ลำดับแทนจุดมุมของรูปหลายเหลี่ยม

โดยคู่ลำดับใน gpntPoint[wCount] มีรูปแบบเหมือนกันกับในคำสั่ง GRIPS_POLYLINE

การทำงาน

วาดรูปหลายเหลี่ยมโดยระบายสีภายในรูป โดยใช้วิธีวาดเหมือนกับคำสั่ง GRIPS_POLYLINE สำหรับสีที่ใช้วาดจะใช้สีปัจจุบัน และสีที่ใช้ระบายภายในก็จะใช้สี และวิธีระบายปัจจุบัน

หมายเหตุ

ตำแหน่งปัจจุบันคือ จุดสุดท้ายในอะเรย์

คำสั่งที่เกี่ยวข้อง

GRIPS_POLYLINE

GRIPS_SETCOLOR

GRIPS_SETFILLCOLOR

GRIPS_SETFILLSTYLE

3.2.4.9 GRIPS_SETFONTTYPE

กำหนดรูปแบบตัวอักษร

รูปแบบคำสั่ง

2 ไบต์ GRIPS_SETFONTTYPE

2 ไบต์ wFontNum หมายเลขรูปแบบตัวอักษร

การทำงาน

กำหนดรูปแบบตัวอักษรที่ใช้พิมพ์ โดยหมายเลขรูปแบบตัวอักษรดูได้จากตารางที่ 3.1

ตารางที่ 3.1 หมายเลขรูปแบบตัวอักษร

หมายเลข	รูปแบบตัวอักษร
0	system font
1	Angsana
2	Bowalia
3	Cordia
4	Dellinia
5	Eucrosia
6	Flecia
7	Iris
8	Jasmine
9	Kodchiang
10	Lily
11	Times New Roman
12	Helvetica Arial
13	Courier

คำสั่งที่เกี่ยวข้อง

GRIPS_SETFONTSIZE

GRIPS_SETFONTSTYLE

GRIPS_TEXTOUT

3.2.4.10 GRIPS_SETFONTSIZE

กำหนดขนาดของตัวอักษร

รูปแบบคำสั่ง

2 ไบต์ GRIPS_SETFONTSIZE 0x000a

2 ไบต์ wWidth ความกว้างของตัวอักษร มีหน่วยเป็นจุด

2 ไบต์ wHigh ความสูงของตัวอักษร มีหน่วยเป็นจุด

การทำงาน

กำหนดขนาดของตัวอักษร

คำสั่งที่เกี่ยวข้อง

GRIPS_SETFONTSTYLE

GRIPS_SETFONTTYPE

GRIPS_TEXTOUT

3.2.4.11 GRIPS_SETFONTSTYLE

กำหนดลักษณะของตัวอักษร

รูปแบบคำสั่ง

2 ไบต์ GRIPS_SETFONTSTYLE

2 ไบต์ wStyle ลักษณะของตัวอักษร

การทำงาน

กำหนดรูปแบบของตัวอักษร โดยค่าของลักษณะตัวอักษรดูได้จากตารางที่ 3.2 โดยค่าลักษณะสามารถนำมาบวกกันได้เพื่อกำหนดลักษณะหลายๆ แบบ และถ้ากำหนดค่าเป็น 0 ก็หมายถึงเป็นตัวอักษรธรรมดา

ตารางที่ 3.2 คำลักษณะของตัวอักษร

ค่า	ลักษณะ
1	ตัวหนา
2	ตัวเอียง
4	ตัวขีดเส้นใต้

คำสั่งที่เกี่ยวข้อง

GRIPS_SETFONTTYPE

GRIPS_SETFONTTYPE

GRIPS_TEXTOUT

3.2.4.12 GRIPS_ELLIPSE

วาดรูปวงรีที่มีจุดศูนย์กลาง, รัศมีในแนวแกนตั้ง และรัศมีในแนวแกนนอนที่กำหนด โดยใช้สีปัจจุบัน

รูปแบบคำสั่ง

2 ไบต์ GRIPS_ELLIPSE

2 ไบต์ wX	จุดศูนย์กลางในแนวแกนนอน
2 ไบต์ wY	จุดศูนย์กลางในแนวแกนตั้ง
2 ไบต์ wXRadial	รัศมีในแนวแกนนอน
2 ไบต์ wYRadial	รัศมีในแนวแกนตั้ง

การทำงาน

วาดรูปวงรีจุดศูนย์กลางอยู่ที่ตำแหน่ง wX, wY โดยมีรัศมีในแนวแกนนอน และแกนตั้ง เป็น wXRadial และ wYRadial ตามลำดับ ในการวาด จะเป็นวงรีแบบไม่ระบายสี และใช้สีปัจจุบัน

คำสั่งที่เกี่ยวข้อง

GRIPS_SETCOLOR

3.2.4.13 GRIPS_SETFILLSTYLE

กำหนดลักษณะการระบายสี

รูปแบบคำสั่ง

2 ไบต์ GRIPS_SETFILLSTYLE

2 ไบต์ wStyle	ลักษณะของการระบายสี

การทำงาน

กำหนดลักษณะของการระบายสี โดยลักษณะของการระบายสีดูได้จากตารางที่ 3.3

ตารางที่ 3.3 ค่าลักษณะของการระบายสี

ค่า	ลักษณะการระบายสี
0	ระบายสีทึบ
1	ระบายสีโปร่ง หรือไม่ระบายสี

คำสั่งที่เกี่ยวข้อง

GRIPS_POLYGON

GRIPS_RECTANGLE

GRIPS_SETFILLCOLOR

3.2.4.14 GRIPS_BITMAP

แสดงรูปบิตแมพที่กำหนด ที่ตำแหน่งที่กำหนด

รูปแบบคำสั่ง

2 ไบต์ GRIPS_BITMAP	
2 ไบต์ wX	ตำแหน่งในแนวแแกนนอนของมุมบนซ้าย
2 ไบต์ wY	ตำแหน่งในแนวแแกนตั้งของมุมบนซ้าย
2 ไบต์ wLen	ความยาวของชื่อไฟล์
n ไบต์ cStr[wLen]	ชื่อไฟล์รูปบิตแมพที่ต้องการนำมาแสดง

การทำงาน

แสดงรูปบิตแมพที่ตำแหน่งที่กำหนด โดยทำการอ่านรูปบิตแมพที่กำหนด มาจากดิสก์ ซึ่งไฟล์บิตแมพเหล่านี้ โปรแกรมโฮสจะส่งมายังเทอร์มินัลตอนผู้ใช้ทำการล็อกอินไปโฮสเป็นครั้งแรก และจะทำการปรับปรุงเพิ่มเติมรูปเท่าที่จำเป็น ทุกครั้งที่ผู้ใช้ทำการล็อกอินไปยังโฮสอีก

3.2.4.15 GRIPS_BUTTON

สร้างปุ่มที่ตำแหน่งที่กำหนด โดยมีขนาด และข้อความตามที่กำหนด โดยเมื่อผู้ใช้กดปุ่ม โปรแกรมเทอร์มินัลจะส่งการตอบสนองไปยังโฮส ตามที่กำหนด

รูปแบบคำสั่ง

2 ไบต์ GRIPS_BUTTON	
2 ไบต์ wX	ตำแหน่งในแนวแแกนนอนของมุมบนซ้าย
2 ไบต์ wY	ตำแหน่งในแนวแแกนตั้งของมุมบนซ้าย
2 ไบต์ wXWidth	ขนาดในแนวแกนนอน
2 ไบต์ wYHeight	ขนาดในแนวแกนตั้ง
2 ไบต์ wLen1	ความยาวของข้อความตอบสนองสำหรับส่งไปโฮสเมื่อผู้ใช้กดปุ่มนี้
2 ไบต์ wLen2	ความยาวของข้อความ
n ไบต์ cStr[wLen1]	ข้อความตอบสนอง
n ไบต์ cStr[wLen2]	ข้อความที่ต้องการพิมพ์บนปุ่ม

การทำงาน

สร้างปุ่มโดยมีขนาด และมีข้อความที่ตำแหน่งที่กำหนด และเมื่อผู้ใช้กดปุ่มนี้ โปรแกรมเทอร์มินัลก็จะส่งข้อความตอบสนองไปยังโฮส โดยปุ่มที่สร้าง จะมีลักษณะขึ้นอยู่กับระบบปฏิบัติการที่ใช้

คำสั่งที่เกี่ยวข้อง

GRIPS_ACTIVEAREA

3.2.4.16 GRIPS_ACTIVEAREA

สร้างพื้นที่รับการกด เมื่อเลื่อนเมาส์เข้ามาในพื้นที่ ตัวชี้เมาส์จะเปลี่ยนรูปร่างเป็นรูปมือชี้ เมื่อผู้ใช้กดปุ่มเมาส์ในบริเวณนี้ โปรแกรมเทอร์มินัลจะส่งการตอบสนองไปยังโฮสตามที่กำหนด

รูปแบบคำสั่ง

2 ไบต์ GRIPS_ACTIVEAREA	
2 ไบต์ wX	ตำแหน่งในแนวแกนนอนของมุมบนซ้าย
2 ไบต์ wY	ตำแหน่งในแนวแกนตั้งของมุมบนซ้าย
2 ไบต์ wXWidth	ขนาดในแนวแกนนอน
2 ไบต์ wYHeight	ขนาดในแนวแกนตั้ง
2 ไบต์ wLen1	ความยาวของข้อความตอบสนองสำหรับส่งไปเมื่อผู้ใช้กดพื้นที่นี้
n ไบต์ cStr[wLen1]	ข้อความตอบสนอง

การทำงาน

คำสั่งนี้ จะไม่ปรากฏรูปใดๆ บนจอ แต่เมื่อผู้ใช้เลื่อนเมาส์เข้าไปในบริเวณนี้ ตัวชี้เมาส์จะเปลี่ยนรูปร่างเป็นรูปมือชี้ และเมื่อผู้ใช้กดปุ่มเมาส์ในบริเวณนี้ โปรแกรมเทอร์มินัลจะส่งการตอบสนองไปยังโฮสตามที่กำหนด

คำสั่งที่เกี่ยวข้อง

GRIPS_BUTTON

3.3 โปรแกรมส่วนเทอร์มินัลบนวินโดวส์

โปรแกรมด้านเทอร์มินัลนี้มีเป็นส่วนหนึ่งของระบบ ซึ่งมีหน้าที่การเป็นตัวกลางในการติดต่อระหว่างผู้ต้องการใช้ข้อมูลสารสนเทศกับแหล่งของข้อมูลสารสนเทศที่ศูนย์ให้บริการ โดยรับคำร้องขอเพื่อการขอข้อมูลต่างๆ จากผู้ใช้ แล้วส่งให้ไปยังศูนย์ให้บริการ เมื่อได้รับการตอบกลับพร้อมกับข้อมูลจากศูนย์ให้บริการแล้ว จึงนำข้อมูลสารสนเทศจากศูนย์ที่ให้บริการมาแสดงทางด้านผู้ใช้ที่ขอข้อมูลสารสนเทศ รวมทั้งดำเนินการตามการการตอบสนองจากผู้ใช้ที่มีต่อข้อมูลสารสนเทศนี้จนกว่าจะจบสิ้นการติดต่อสื่อสารระหว่างผู้ขอใช้ข้อมูล และผู้ให้บริการข้อมูล

3.3.1 กล่าวทั่วไป

สำหรับส่วนโปรแกรมที่ทำหน้าที่เป็นตัวกลางระหว่างผู้ใช้และศูนย์ให้บริการข้อมูลที่ทำงานในระบบวินโดวส์นี้ มีเป้าหมายสำคัญในเรื่องของการใช้ส่วนติดต่อกับผู้ใช้และฟังก์ชันที่เกี่ยวข้องของระบบวินโดวส์ให้เป็นประโยชน์ในการทำให้ผู้ใช้สามารถใช้งานระบบติดต่อกับผู้ใช้ที่เป็นมาตรฐาน เช่นเดียวกับโปรแกรมในสภาพแวดล้อมแบบกราฟิกของวินโดวส์ทั่วไป และสามารถใช้ความสามารถรวมถึงทรัพยากรต่างๆ ของระบบคอมพิวเตอร์ให้เป็นประโยชน์ได้มีประสิทธิภาพอย่างเต็มที่ เนื่องจากระบบวินโดวส์นั้น สามารถดึงเอาความสามารถของหน่วยประมวลผลของคอมพิวเตอร์มาใช้ได้มากกว่าระบบปฏิบัติการดอส

3.3.2 การพัฒนาโปรแกรม

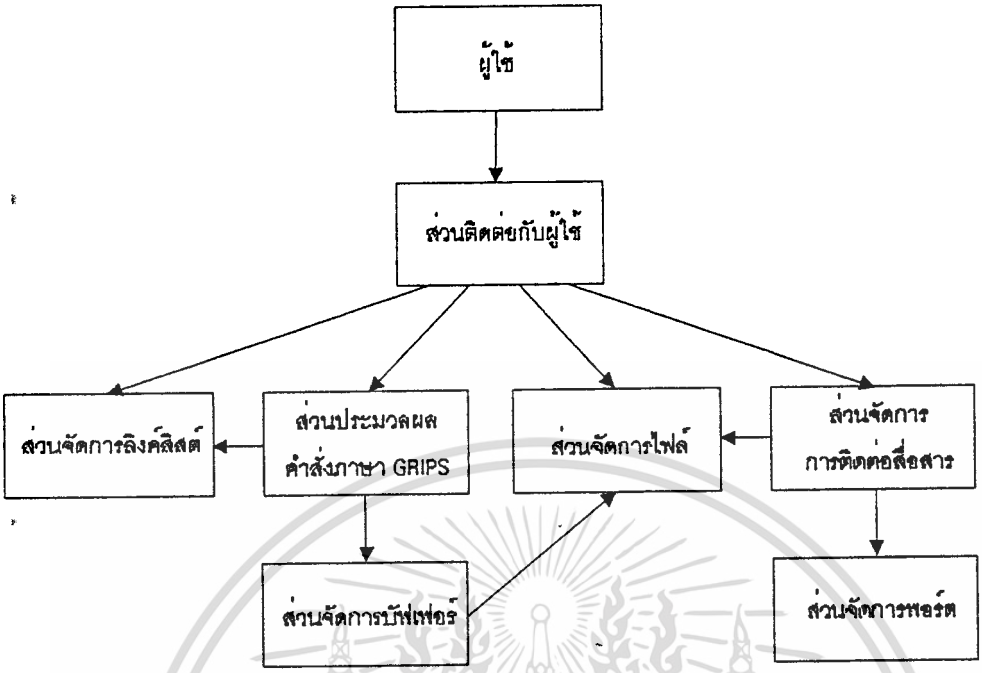
สำหรับด้านเครื่องมือที่ใช้ในการพัฒนาในระบบวินโดวส์นั้น ในโปรแกรมนี้ได้ใช้ภาษาการทำโปรแกรมคือภาษาซีเป็นส่วนหลัก แต่ในการคอมไพล์นั้นเลือกให้เป็นแบบซีพลัสพลัสโดยอัตโนมัติ ด้วยการตั้งนามสกุลของไฟล์เป็น .CPP เพราะมีบางส่วนที่จำเป็นต้องใช้ความสามารถของภาษาซีพลัสพลัส แต่ไม่ใช่เป็นการพัฒนาโปรแกรมแบบเชิงวัตถุแต่อย่างใด ซึ่งเป็นข้อดีสำหรับการพัฒนาโปรแกรมในระบบวินโดวส์ ที่ใช้วิธีการเขียนโปรแกรมแบบภาษาซีธรรมดา เพราะทำให้โปรแกรมสามารถในไปคอมไพล์โดยคอมไพเลอร์ตัวอื่นๆ ได้ โดยเพียงแต่ทำการไขเมคไฟล์ให้เหมาะสมกับแต่ละคอมไพเลอร์เท่านั้น

ส่วนของซอร์สไฟล์ของโปรแกรมนั้น ได้แยกส่วนที่เป็นซอร์สโปรแกรมของแอพพลิเคชัน ซึ่งประกอบไปด้วยส่วนที่เกี่ยวข้องโดยตรงกับแอพพลิเคชัน ไม่ว่าจะเป็นข้อมูลเฉพาะ, ฟังก์ชันที่ใช้เฉพาะซึ่งมีการใช้ตัวแปรร่วม (global variable), ฟังก์ชันหลัก, ฟังก์ชันประจำวินโดว์ และค่าคงที่ที่ใช้เฉพาะแอพพลิเคชันเอาไว้ออกเป็นส่วนต่างหากจากอีกส่วนที่สามารถใช้โดยโปรแกรมอื่นๆ ได้ โดยส่วนนี้จะประกอบไปด้วยรูปแบบของโครงสร้างข้อมูล พร้อมกับฟังก์ชันที่จัดการโครงสร้างข้อมูล, ฟังก์ชันที่เป็นอิสระจากแอพพลิเคชันและไม่ค่อยแก้ไขบ่อยนัก เพื่อทำให้เวลาที่มีการแก้ไขแอพพลิเคชันจะได้ไม่ต้องทำการคอมไพล์ใหม่หมดทุกส่วน

ในขณะที่มีการพัฒนานั้น เพื่อป้องกันความสับสนในซอร์สทั้งหมดของโปรแกรม จึงได้มีการกำหนดรูปแบบของหัวไฟล์ในทุกทุกโปรแกรม โดยระบุวันที่และรายละเอียดที่เกี่ยวข้องเข้าไปด้วย รวมทั้งมีการควบคุมหมายเลขของแอพพลิเคชันโดยระบุวันที่ เดือน และปีเข้าไปกับเลขเวอร์ชันด้วย โดยเก็บไว้ในตารางสตริง (string table) ที่อยู่ใน รีซอร์สไฟล์

3.3.3 ส่วนประกอบต่างๆ ของโปรแกรม

สำหรับส่วนประกอบต่างๆ ของโปรแกรม นั้นสามารถแยกเป็นส่วนต่างๆ ตามรูปที่ 3.9 โดยแสดงถึงส่วนประกอบแต่ละส่วนของโปรแกรมรวมทั้งสภาพแวดล้อม และความสัมพันธ์ระหว่างแต่ละส่วนประกอบซึ่งแสดงโดยการใช้เส้นเชื่อมระหว่างกัน การติดต่อระหว่างส่วนต่างๆ ที่ออกแบบไว้ ในทางโปรแกรมแทนได้ด้วยทั้งการติดต่อด้วยการส่งเมสเสจ และการเรียกใช้ฟังก์ชันที่เป็นส่วนหนึ่งของส่วนประกอบนั้น



รูปที่ 3.9 แสดงส่วนประกอบต่างๆ ของโปรแกรม และความสัมพันธ์

3.3.3.1 ส่วนติดต่อกับผู้ใช้

เป็นส่วนที่ทำหน้าที่รับคำสั่งและแสดงข้อมูลแก่ผู้ใช้ ซึ่งปรากฏในลักษณะของหน้าจอและคอนโทรลต่างๆ ของโปรแกรม ซึ่งผู้ใช้สามารถเห็นได้โดยตรง การออกแบบส่วนนี้จึงใช้หลักการเหมือนกับโปรแกรมในระบบกราฟิกทั่วไป โดยใช้รูปแบบของการติดต่อกับผู้ใช้ทั้งที่มีอยู่แล้ว โดยระบบวินโดวส์จะจัดการให้ เช่น ระบบเมนู, กรอบข้อความ และบางส่วนที่ต้องจัดการเอง แต่โดยรวมแล้วจะเน้นการควบคุมโดยใช้เมาส์เป็นหลัก ซึ่งทำให้ผู้ใช้สามารถใช้คำสั่งต่างๆ ของโปรแกรมได้สะดวกกว่า ส่วนคำสั่งบางประเภทนั้นสามารถรับอินพุทจากแป้นพิมพ์ได้แต่ผ่านการประมวลผลก่อนเพื่อสามารถทำให้มีผลลัพธ์เหมือนกับการใช้เมาส์

3.3.3.2 ส่วนประมวลผลคำสั่ง GRIPS

ส่วนนี้เป็นส่วนที่สำคัญของโปรแกรมเทอร์มินัลมากที่สุดส่วนหนึ่ง โดยทำหน้าที่แปลความหมายของภาษา GRIPS (ตามข้อกำหนดที่ระบุไว้ ในโปรโตคอลระดับแอพพลิเคชัน) ที่เก็บไว้ ให้เป็นผลลัพธ์สำหรับแสดงออกทางหน้าจอแสดงผล และเพื่อให้ได้ประสิทธิภาพสูงสุด จึงได้ออกแบบส่วนนี้ให้มีการทำงานคล้ายกับเป็นอินเตอร์พรีเตอร์ (interpreter) ซึ่งจะตีความคำสั่งในภาษา GRIPS ที่ละคำสั่งแล้วทำงานตามคำสั่งนั้นทันที

3.3.3.3 ส่วนจัดการไฟล์

ส่วนจัดการไฟล์ที่มีอยู่ในโปรแกรมนี้ จะเกี่ยวข้องกับการจัดการไฟล์โดยทั่วไปเช่น การเปิดไฟล์ การตรวจสอบความผิดพลาดต่างๆ ที่เกิดขึ้นจากไฟล์ การเก็บข้อมูลที่เกี่ยวข้องกับไฟล์ที่สำคัญไว้เพื่อใช้ในการติดต่อสื่อสาร

รวมทั้งส่วนที่เกี่ยวข้องกับไฟล์เก็บค่าเริ่มต้นของแอพพลิเคชัน (initial file) ซึ่งใช้ความสามารถของระบบวินโดวส์ที่สนับสนุนในเรื่องนี้อยู่แล้ว

ทั้งหมดของส่วนจัดการไฟล์ที่ใช้ในโปรแกรมนั้น ใช้การจัดการไฟล์แบบโดยใช้ไฟล์แฮนเดิล (file handle) ซึ่งต่างจากปกติในดอสทั่วไปที่ใช้การจัดการไฟล์แบบสตรีม เนื่องจากฟังก์ชันเกี่ยวกับไฟล์ส่วนใหญ่ของวินโดวส์นั้นใช้ไฟล์แฮนเดิล ไม่ได้ใช้โครงสร้างข้อมูลไฟล์ ในกรณีการจัดการแบบสตรีม

3.3.3.4 ส่วนจัดการการติดต่อสื่อสาร

ส่วนจัดการการติดต่อสื่อสารนี้จะทำหน้าที่ในการติดต่อระหว่างผู้ใช้ซึ่งเป็นผู้ขอข้อมูล กับศูนย์บริการข้อมูล ทั้งในแบบผ่านการติดต่อแบบท้องถิ่น และการติดต่อผ่านการสื่อสารทางโทรศัพท์ ซึ่งประกอบไปด้วยการเริ่มต้นการติดต่อ การตอบสนองต่อการติดต่อ การรับส่งข้อมูลระหว่างกันด้วยวิธีการต่างๆ และการเลิกการติดต่อสื่อสาร รวมถึงส่วนอื่นๆ ที่เกี่ยวข้องกับอุปกรณ์โมเด็มที่ใช้ในการสื่อสาร การรับส่งคำสั่ง การจัดการกับข้อมูลในการติดต่อ ซึ่งเป็นไปตามข้อกำหนดในโปรโตคอลระดับล่าง การจัดการหมายเลขโทรศัพท์ เป็นต้น

3.3.4 โครงสร้างข้อมูล

โครงสร้างข้อมูลที่ใช้ในโปรแกรมนี้ มีเป้าหมายในการออกแบบเพื่อให้สามารถใช้ได้อย่างทั่วไป ดังนั้นจึงได้ทำการออกแบบแยกตามโครงสร้างข้อมูลที่ใช้สองแบบ คือ ส่วนที่เป็นบัฟเฟอร์ (buffer) และส่วนที่เป็นลิงค์ลิสต์ (linked list)

3.3.4.1 บัฟเฟอร์

บัฟเฟอร์เป็นโครงสร้างข้อมูลที่ใช้เก็บข้อมูลต่างๆ ที่มีการเปลี่ยนแปลงบ่อย จึงต้องมีการออกแบบทั้งโครงสร้างและฟังก์ชันให้อำนวยความสะดวกในการเรียกใช้โดยส่วนอื่นๆ ได้ง่าย

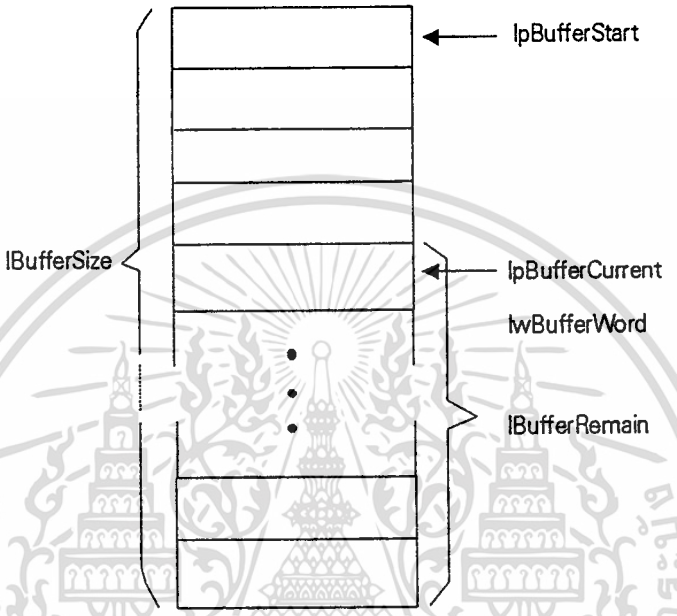
3.3.4.1.1 การใช้งาน

ส่วนบัฟเฟอร์นี้จะถูกเรียกใช้โดยส่วนประมวลผลคำสั่งภาษา GRIPS เป็นส่วนใหญ่ แต่เป็นการอ่านเป็นอย่างเดียว ส่วนการเขียนข้อมูลลงบัฟเฟอร์นี้จะเป็นหน้าที่ของส่วนจัดการไฟล์ ซึ่งจะอ่านไฟล์เข้ามาเก็บในบัฟเฟอร์นี้ นอกจากนั้นยังมีการเรียกใช้จากส่วนติดต่อกับผู้ใช้บางฟังก์ชัน ที่ต้องการเข้าถึงข้อมูลในบัฟเฟอร์เพื่อใช้ในการแสดงผลด้วย

3.3.4.1.2 โครงสร้าง

โครงสร้างข้อมูลของบัฟเฟอร์นี้ มีความสัมพันธ์โดยตรงกับวิธีการจัดการหน่วยความจำของวินโดวส์จึงต้องพิจารณาตัวแปรต่างๆ ที่จำเป็นในการใช้ เมื่อมีการเรียกใช้ฟังก์ชันของระบบที่เกี่ยวข้องกับการจัดการหน่วยความจำด้วย จึงได้รูปแบบของโครงสร้างข้อมูลบัฟเฟอร์ตามรูปที่ 3.10

```
typedef struct tagBUFFER {
  HGLOBAL hgBuffer;
  LPSTR   lpBufferCurrent;
  LPSTR   lpBufferStart;
  LPWORD  lpwBufferWord;
  LONG    lBufferRemain;
  LONG    lBufferSize;
} BUFFER;
```



รูปที่ 3.10 โครงสร้างข้อมูลของบัฟเฟอร์

สำหรับรายละเอียดของตัวแปรต่างๆ ในโครงสร้างนี้คือ

hgBuffer เป็นแฮนเดิลแบบ HGLOBAL ของหน่วยความจำที่จองโดยบัฟเฟอร์ ซึ่งเป็นหน่วยความจำรวมของระบบวินโดวส์ (รวมทั้งหน่วยความจำเสมือนด้วย) ทำให้ขนาดของหน่วยความจำบัฟเฟอร์นี้สามารถมีขนาดใหญ่กว่าขนาดของเซกเมนต์ (segment) หรือ 64 กิโลไบต์ได้

lpBufferCurrent เป็นพอยเตอร์ที่ชี้ไปยังตำแหน่งของหน่วยความจำของบัฟเฟอร์ที่ใช้อยู่ในปัจจุบัน ซึ่งเป็นตัวแบบ LPSTR ซึ่งก็คือ char far * นั่นเอง โดยปกติพอยเตอร์นี้จะถูกควบคุมโดยฟังก์ชันที่ใช้ในการจัดการบัฟเฟอร์ต่างๆ จึงมีค่าที่เปลี่ยนแปลงไปอยู่เสมอ ซึ่งการใช้จะใช้ในกรณีที่อ่านข้อมูลข้อความที่เก็บเป็นสตริงที่ตัวอักษรเรียงต่อกันไป

lpBufferStart เป็นพอยเตอร์ประเภทเดียวกันกับ lpBufferCurrent แต่ชี้ไปที่จุดเริ่มต้นของบัฟเฟอร์ ซึ่งเก็บจุดเริ่มต้นของหน่วยความจำบัฟเฟอร์ไว้ ดังนั้นในการจองและใช้บัฟเฟอร์ครั้งหนึ่งๆ ค่าพอยเตอร์นี้จะไม่เปลี่ยนแปลงอีก จนกว่าจะมีการคืนหน่วยความจำที่จองไว้ในบัฟเฟอร์นี้ แล้วจองใหม่อีกครั้ง

lpwBufferWord พอยเตอร์แบบเวิร์ด LPWORD หรือ unsigned int far * ที่ชี้ในตำแหน่งปัจจุบันเช่นเดียวกับพอยเตอร์ lpBufferCurrent เพราะบัฟเฟอร์ที่ออกแบบนี้จัดการแบบที่จัดเรียงแบบเวิร์ด (word align) ส่วนการใช้นั้ ใช้ในการอ่านข้อมูลที่เป็นจำนวนที่มีขนาดเป็นเวิร์ด และดับเบิลเวิร์ด (double word)

lBufferRemain ใช้ในการเกบขนาดของบัฟเฟอร์ที่เหลืออยู่ เมื่อพิจารณาจากพอยเตอร์ที่ชี้ตำแหน่งปัจจุบัน ซึ่งค่านี้จะถูกปรับทุกครั้งที่มีการอ่านบัฟเฟอร์หรือ lpBufferCurrent เปลี่ยนไป

lBufferSize เก็บขนาดของหน่วยความจำบัฟเฟอร์ทั้งหมด ของบัฟเฟอร์

3.3.4.1.3 ฟังก์ชันในการจัดการบัฟเฟอร์

ฟังก์ชันในการจัดการบัฟเฟอร์สามารถแบ่งตามประเภทได้ดังนี้

3.3.4.1.3.1 การจองและเลิกใช้บัฟเฟอร์

ฟังก์ชันในกลุ่มนี้ คือ

- BufferAlloc ใช้ในการจองหน่วยความจำในบัฟเฟอร์
- BufferFree ใช้ในการเลิกใช้หน่วยความจำในบัฟเฟอร์

การจองบัฟเฟอร์สำหรับโครงสร้างข้อมูลบัฟเฟอร์นั้น เริ่มต้นจะทำการตรวจสอบแฮนเดิลของหน่วยความจำที่เก็บในโครงสร้างข้อมูลบัฟเฟอร์ก่อน ถ้ามีผ่านการจองมาแล้ว (ไม่เป็น NULL) จะทำการเลิกใช้ก่อนโดยการเรียกใช้ฟังก์ชัน GlobalFree สำหรับพารามิเตอร์ที่เป็นแฮนเดิลนี้ก่อน แล้วจึงจองใหม่อีกครั้งหนึ่งด้วยการใช้ฟังก์ชัน GlobalAlloc ถ้าจองไม่สำเร็จจะรีเทินกลับพร้อมกับค่า FALSE

ส่วนถ้าการจองสำเร็จ ขั้นต่อไปจะทำการกำหนดค่าตำแหน่งของหน่วยความจำสำหรับแฮนเดิลที่ได้จากการจองแก่ตัวแปรที่ใช้ในการชี้หน่วยความจำ lpBufferCurrent โดยใช้ฟังก์ชัน GlobalLock และเรียกใช้ฟังก์ชัน GlobalUnlock อีกครั้งเมื่อทำการกำหนดค่าเสร็จ หลังจากนั้นจึงทำการเก็บค่าข้อมูลต่างที่เกี่ยวข้องลงไปในโครงสร้างบัฟเฟอร์จนครบ

สำหรับการเลิกใช้บัฟเฟอร์นั้นจะใช้ฟังก์ชัน GlobalUnlock เพื่อปล่อยหน่วยความจำนั้นก่อน แล้วจึงเรียกใช้ฟังก์ชัน GlobalFree เพื่อเลิกใช้หน่วยความจำนี้

3.3.4.1.3.2 การเขียนข้อมูลลงบัฟเฟอร์

- BufferFromFile ใช้ในการอ่านข้อมูลจากไฟล์เพื่อเก็บลงในบัฟเฟอร์

เริ่มต้นด้วยการเปิดไฟล์ที่ต้องการอ่านเข้ามาก่อน หลังจากนั้นทำการเรียกใช้ฟังก์ชันที่ทำหน้าที่จองหน่วยความจำในบัฟเฟอร์ แล้วจึงเรียกฟังก์ชัน _hread เพื่อทำการอ่านข้อมูลจากไฟล์แฮนเดิลเข้ามาเก็บไว้ในหน่วยความจำบัฟเฟอร์ตามขนาดของไฟล์ที่อ่านเข้ามา แล้วจึงปรับค่าตัวแปรต่างๆ ในโครงสร้างบัฟเฟอร์ตามขนาดของหน่วยความจำบัฟเฟอร์ใหม่

3.3.4.1.3.3 การปรับบัฟเฟอร์

BufferAdjust ใช้ในการปรับบัฟเฟอร์

BufferStart ปรับบัฟเฟอร์ให้กลับไปในสภาพเริ่มต้น

ฟังก์ชันที่ใช้ในการปรับขนาดจะทำการปรับค่าของพอยเตอร์ที่ชี้ไปยังหน่วยความจำบัฟเฟอร์ทั้ง lpBufferCurrent และ lwBufferWord รวมทั้งขนาดของบัฟเฟอร์ที่เหลืออยู่ ตามพารามิเตอร์ที่เป็นขนาดที่ต้องการจะปรับ

ส่วนในกรณีของการปรับบัฟเฟอร์ไปยังสภาพเริ่มต้นนั้น จะใช้ประโยชน์จากจุดเริ่มต้นของหน่วยความจำบัฟเฟอร์ที่เก็บไว้ในตัวแปร lpBufferStart โดยปรับทั้งพอยเตอร์ lpBufferCurrent และ lwBufferWord ให้เท่ากับตัวแปร lpBufferStart และปรับขนาดที่เหลืออยู่ให้เท่ากับขนาดของบัฟเฟอร์ดังเดิม ที่เก็บไว้ในตัวแปร lBufferSize

3.3.4.1.3.4 การอ่านข้อมูลจากบัฟเฟอร์

- BufferGetWORD อ่านข้อมูลเวิร์ดจากบัฟเฟอร์
- BufferGetDWORD อ่านข้อมูลดับเบิลเวิร์ดจากบัฟเฟอร์
- BufferGetSTR อ่านสตริงจากบัฟเฟอร์

การอ่านข้อมูลจากบัฟเฟอร์นั้นจะใช้ประโยชน์จากพอยเตอร์ที่ชี้หน่วยความจำแบบเวิร์ด ในการให้ข้อมูลที่พอยเตอร์นั้นชี้อยู่ได้ทันที ส่วนในกรณีที่ต้องการข้อมูลแบบดับเบิลเวิร์ดจะต้องอ่านข้อมูลแบบเวิร์ดเข้ามาเก็บไว้ก่อน 2 เวิร์ด แล้วจึงเรียกมาโคร (macro) MAKELONG ให้เป็นข้อมูลแบบดับเบิลเวิร์ดอีกทีหนึ่ง แต่สิ่งสำคัญที่ต้องทำในทั้งสองกรณีเหมือนกันคือ ต้องมีการปรับค่าพอยเตอร์ทั้งหมดที่ชี้หน่วยความจำในบัฟเฟอร์นี้ และขนาดของบัฟเฟอร์ที่เหลืออยู่ด้วย นั่นคือ ในกรณีที่เป็นเวิร์ดจะต้องปรับ lpBufferCurrent เลื่อนไปอีก 2ปรับ lwBufferWord ไปอีก 1 พร้อมกับลดค่าของ lBufferRemain ลงไป 2 ส่วนถ้าเป็นดับเบิลเวิร์ดจะทำเช่นเดียวกัน แต่มีปริมาณในการปรับเป็น 2 เท่าของเวิร์ด

ส่วนการอ่านสตริงจากบัฟเฟอร์นั้นจะเริ่มอ่านจากพอยเตอร์ lpBufferCurrent แล้วเก็บไว้ในสตริงที่รับข้อมูล พร้อมกับปรับเลื่อนพอยเตอร์นี้ไปเรื่อย ๆ จนกระทั่งจบตามความยาวที่กำหนด โดยต้องคำนึงความยาวของสตริงที่มีจำนวนคี่ด้วย เพราะบัฟเฟอร์นี้จะเก็บเรียงตามเวิร์ดเสมอ ดังนั้นจึงต้องมีการปรับบัฟเฟอร์ตามขนาดที่เหมาะสมด้วย นั่นคือถ้าความยาวของสตริงเป็นคี่จะบวกขนาดที่ต้องปรับไปอีก 1 ซึ่งโดยปกติข้อมูลที่ตำแหน่งเสริมนี้จะเป็นศูนย์เสมอ

3.3.4.1.3.5 การขอข้อมูลจากบัฟเฟอร์

- BufferRemain ใช้ในการตรวจสอบขนาดบัฟเฟอร์ที่เหลืออยู่
- BufferCurrent ให้ค่าพอยเตอร์ที่ชี้หน่วยความจำปัจจุบัน
- BufferCurrentWORD ให้ค่าพอยเตอร์ที่ชี้หน่วยความจำปัจจุบันแบบเวิร์ด

ทั้งสามฟังก์ชันนั้นมีการทำงานเหมือนกันคือให้ค่าข้อมูลที่เก็บไว้อยู่แล้วในโครงสร้างข้อมูลบัฟเฟอร์ เพราะไม่ต้องการให้ผู้ใช้เข้าถึงตัวแปรในโครงสร้างข้อมูลบัฟเฟอร์เองโดยตรง

3.3.4.2 ลิงค์ลิสต์

ลิงค์ลิสต์เป็นโครงสร้างที่ใช้ในการเก็บข้อมูลหลายๆ ตัวต่อกันไป โดยมีพอยเตอร์เชื่อมถึงกัน ซึ่งมีลักษณะเป้าหมายสำคัญในการใช้เป็นโครงสร้างข้อมูลอาร์เรย์ (array) แบบไดนามิก ซึ่งจองหน่วยความจำสำหรับแต่ละหน่วยข้อมูล (item) เฉพาะเมื่อมีการจะเพิ่มเข้าไปในลิงค์ลิสต์เท่านั้น

3.3.4.2.1 การใช้งาน

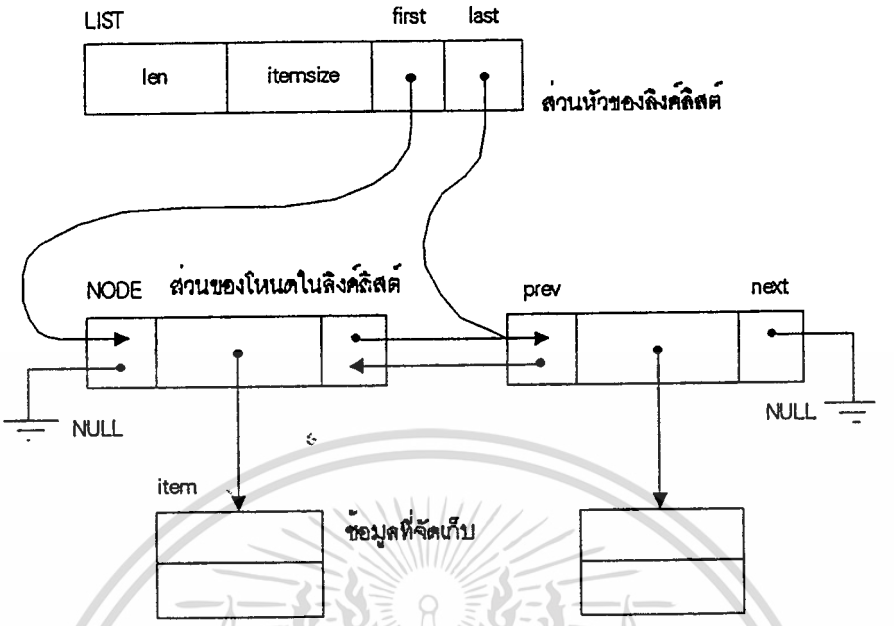
โครงสร้างข้อมูลแบบลิงค์ลิสต์นี้ มีการใช้งานในส่วนประมวลผลคำสั่งภาษา GRIPS โดยใช้เก็บข้อมูลที่เกี่ยวข้องกับรูปบิตแมพที่แสดงในหน้าจอหนึ่ง ๆ และข้อมูลของปุ่มกด ซึ่งเป็นการเรียกใช้ในลักษณะของการจัดเก็บข้อมูลในลิสต์ การลบข้อมูลจากลิสต์ อีกส่วนหนึ่งที่มีการใชงานคือส่วนติดต่อกับผู้ใช้ ซึ่งใช้ข้อมูลปุ่มกดจากที่เก็บไว้ในลิงค์ลิสต์ ในการส่งผลลัพธ์ที่กำกับไว้กับปุ่มนี้ไปใช้ต่อไป

3.3.4.2.2 โครงสร้าง

สำหรับรูปแบบของลิงค์ลิสต์ที่ใช้ในโครงสร้างนี้เป็นดับบลีลิงค์ลิสต์ (doubly linked list) โดยส่วนหัวของลิสต์จะเก็บขนาดของข้อมูลแต่และชิ้น (item) และจำนวนของชิ้นข้อมูลในลิสต์ มีพอยเตอร์ชี้ไปยังโหนด (node) เริ่มต้นและสุดท้ายในลิสต์

ส่วนในแต่ละโหนดนั้นจะเก็บพอยเตอร์ไปยังโหนดตัวถัดไปและตัวก่อนหน้า แต่จะมีบางโหนดที่มีพอยเตอร์ชี้โหนดใกล้เคียงบางตัวเป็น NULL คือ พอยเตอร์ชี้โหนดก่อนหน้าของโหนดแรก และพอยเตอร์ชี้โหนดถัดไปของโหนดสุดท้าย โดยมีการกำหนดโครงสร้างของข้อมูลดังนี้

```
typedef struct tagNODE {
  struct tagNODE *prev;
  void *item;
  struct tagNODE *next;
} NODE;
typedef struct tagLIST {
  WORD len;
  WORD itemsize;
  NODE *first;
  NODE *last;
} LIST;
```



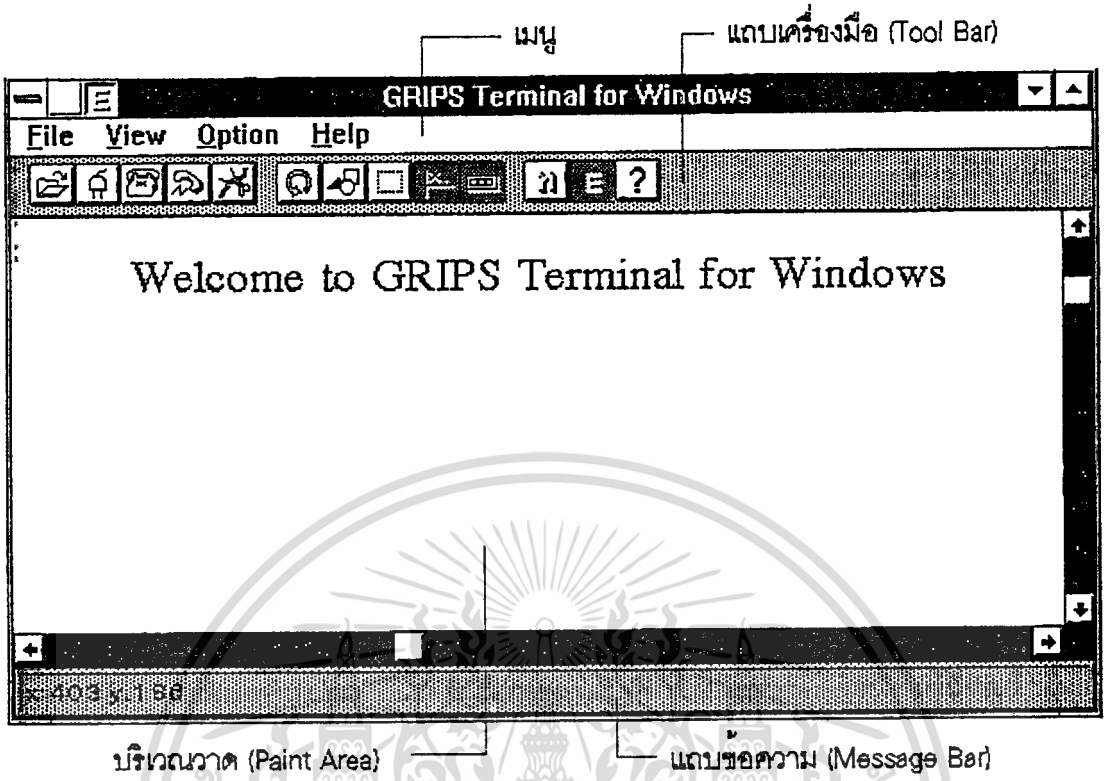
รูปที่ 3.11 โครงสร้างของลิงค์ลิสต์

3.3.4.2.3 ฟังก์ชันสำหรับลิงค์ลิสต์

- List_Init ใช้ในการเริ่มต้นใช้ลิงค์ลิสต์ โดยระบุขนาดของชิ้นข้อมูลในแต่ละโหนด
- List_Ins_First เพิ่มโหนดเข้าไปในลิสต์ที่หัว
- List_Ins_Last เพิ่มโหนดเข้าไปในลิสต์ที่ด้านท้าย
- List_Del_First ดึงโหนดแรกออกจากลิสต์
- List_Del_Last ดึงโหนดสุดท้ายออกจากลิสต์
- List_Item ให้โหนดตามตำแหน่งที่ระบุในลิสต์
- List_Len ใช้หาความยาวของลิสต์

3.3.5 ส่วนติดต่อกับผู้ใช้

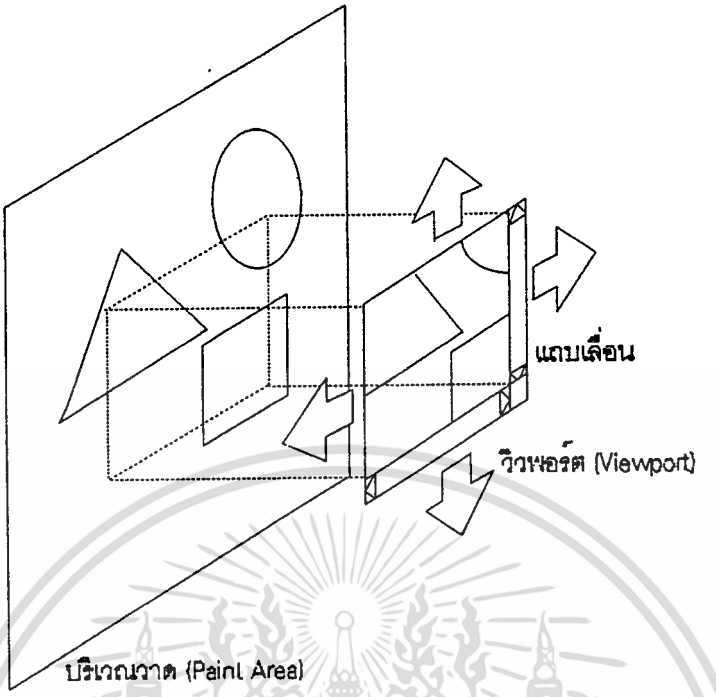
ส่วนติดต่อกับผู้ใช้นี้เป็นส่วนสำคัญที่สุดของโปรแกรม ซึ่งจะเป็นสิ่งที่ปรากฏให้ผู้ใช้เห็น และจัดการการติดต่อกับผู้ใช้ทั้งด้านรับและแสดงผลข้อมูล ถ้าดูจากหน้าจอของโปรแกรมแล้วสามารถแบ่งแยกเป็นส่วนต่างๆ ตามที่แสดงไว้ในรูปที่ 3.12



รูปที่ 3.12 หน้าจอโปรแกรมซึ่งแสดงส่วนประกอบต่างๆ ของส่วนติดต่อกับผู้ใช้

3.3.5.1 บริเวณวาด

เริ่มต้นจากรูปแบบของโปรแกรมที่จะปรากฏแก่ผู้ใช้นั้น มีเป้าหมายสำคัญที่ใช้ในการแสดงข้อมูลสารสนเทศต่างๆ ที่มีรูปแบบเป็นหน้าจอหนึ่งๆ ดังนั้นส่วนที่สำคัญที่สุดของโปรแกรมคือส่วนที่ใช้สำหรับการแสดงข้อมูลสารสนเทศเหล่านี้ จึงได้มีการออกแบบวินโดว์ในลักษณะที่เหมือนกับเป็นแผ่นกระดาษว่างๆ ขนาดใหญ่อยู่แผ่นหนึ่ง เรียกว่าบริเวณสำหรับวาด ซึ่งมีขนาดตามกำหนดของข้อมูลสารสนเทศแต่ละชุด เมื่อข้อมูลเหล่านี้ถูกวาดลงไปยังบริเวณนี้แล้ว จะมีลักษณะเหมือนกับเป็นใบปิดหรือแผ่นโปสเตอร์ขนาดใหญ่ ซึ่งอาจจะแสดงบนหน้าจอไม่ได้ครบทั้งหมด สอดคล้องกับหลักการวิวพอร์ตของวินโดวส์ ที่สามารถแสดงผลภาพที่มีขนาดใหญ่ในวินโดว์ได้ โดยใช้วินโดว์เหมือนกับเป็นช่องเพื่อมองผ่านไปยังพื้นที่วาดอีกทีหนึ่ง ดังรูปที่ 3.13



รูปที่ 3.13 ลักษณะวิวพอร์ตของบริเวณวาด

จากรูปแสดงให้เห็นลักษณะของความสัมพันธ์ระหว่างภาพที่ปรากฏในช่องมองในวินโดว์ กับภาพในบริเวณจริง เมื่อมีการเลื่อนแถบเลื่อนที่วินโดว์เกิดขึ้นนั้น ดูเหมือนว่าวินโดว์จะเป็นฝ่ายเลื่อนไปบนบริเวณวาด แต่ความจริงแล้วบริเวณวาดจะเลื่อนในทิศตรงข้ามกับแถบเลื่อน เช่น ถ้าเลื่อนแถบเลื่อนขึ้นข้างบน นั่นคือบริเวณวาดจะทำการวาดในวิวพอร์ตโดยมีจุดเริ่มต้นที่อ้างอิงกับวิวพอร์ตเลื่อนลงมา ทำให้เห็นบริเวณวาดในวิวพอร์ตลงมา เหมือนกับเลื่อนวิวพอร์ตขึ้นไป

ส่วนบริเวณวาดนี้มีความสัมพันธ์โดยตรงกับเมสเสจ WM_PAINT โดยการกระทำในบริเวณวาดที่เกี่ยวข้องกับการแสดงผลข้อมูลจะถูกทำในช่วงเวลาเมสเสจนี้ทั้งหมด ซึ่งมีความสัมพันธ์โดยตรงกับส่วนประมวลผลคำสั่งภาษา GRIPS ที่จะตีความหมายเป็นคำสั่งในการวาดภาพที่ใช้ฟังก์ชันกราฟิกของ GDI แล้วจึงแสดงผลให้ปรากฏในบริเวณวาดนี้ทุกครั้งที่เกิดเมสเสจ WM_PAINT ขึ้น

สำหรับข้อมูลที่แสดงผลในบริเวณวาดนี้จะถูกเก็บไว้ในบัฟเฟอร์หนึ่ง ซึ่งทำหน้าที่เป็นบัฟเฟอร์ของการแสดงผล โดยมีส่วนประมวลผลคำสั่งภาษา GRIPS คอยจัดการอีกทีหนึ่ง ในการแปลเป็นคำสั่งกราฟิกของ GDI ไปวาดเก็บไว้ในคอนเท็กซ์ของการแสดงผล แล้วระบบวินโดวส์จะจัดการวาดในบริเวณวาดนี้ต่อไป

3.3.5.2 เมนู

เมนูที่ใช้แสดงรายการคำสั่งที่ใช้ในโปรแกรมนี้ เป็นส่วนหนึ่งของการติดต่อกับผู้ใช้มาตรฐานที่วินโดวส์มีมาให้ ดังนั้นการจัดการจึงสามารถปล่อยให้เป็นหน้าที่ของวินโดวส์ได้ หน้าที่ของผู้พัฒนาโปรแกรมจึงเพียงแต่ออกแบบเมนูตามคำสั่งที่ต้องการให้ผู้ใช้สั่งโปรแกรมให้ทำงานตามคำสั่งนั้น โดยระบุเมนูย่อยและเมนูหลักตามมาตรฐานที่มี

การใช้อยู่ทั่วไป และกำหนดค่าคงที่ (define) ของแต่ละรายการในเมนูไม่ให้ซ้ำกัน เพื่อทำให้ทั้งเมนูที่ออกแบบไว้ในรีซอร์สไฟล์ และในโปรแกรมรู้จักรายการในเมนูเดียวกัน

นอกจากนั้นยังเป็นหน้าที่ของโปรแกรมที่จะต้องทำการปรับเปลี่ยนค่าที่แสดงสถานะของเมนู ตามแต่ละเหตุการณ์ที่เกิดขึ้นในโปรแกรม ได้แก่ การคืนสภาพ การหยุดคืนสภาพ การปรับให้ขีดเช็คหรือไม่

สำหรับเมนูที่ใช้โปรแกรมที่ออกแบบนี้เนื่องจากมีความต้องการในการใช้ความสามารถในการแสดงผลภาษาไทยด้วย จึงได้มีการออกแบบรายการเมนูออกเป็นสองชุด คือ เมนูที่เป็นภาษาไทย และภาษาอังกฤษซึ่งจะตอบสนองต่อคำสั่งที่ใช้ในการเปลี่ยนภาษาได้

3.3.5.3 แถบเครื่องมือ

แถบเครื่องมือนี้เป็นเครื่องมือในการติดต่อกับผู้ใช้แบบหนึ่ง ที่นิยมใช้กันอย่างมากในระบบกราฟิกเช่น วินโดวส์ ในการออกแบบจึงได้นำแบบอย่างจากที่มีกันใช้ทั่วไปนั้นมาเป็นต้นแบบ โดยให้มีความรู้สึกเหมือนกันแต่การจัดการนั้นอาจจะต่างกัน

สำหรับแถบเครื่องมือที่ใช้ในโปรแกรมนี้ ได้ออกแบบให้เป็นวินโดว์หนึ่งที่เป็นลูกของวินโดว์หลักของโปรแกรม และปุ่มกดต่างๆ ที่อยู่บนแถบเครื่องมือก็เป็นวินโดว์ลูกของแถบเครื่องมืออีกทีหนึ่ง โดยมีฟังก์ชันประจำวินโดว์ของปุ่มกด ซึ่งคอยจัดการวาดรูปบิตแมพสำหรับสถานะของปุ่มเมื่อมีการกดปุ่มหรือปล่อยปุ่มกด และคอยส่งค่าของปุ่มกด ซึ่งแทนค่าประจำรายการในเมนูเพื่อทำงานตามแต่ละคำสั่งอีกทีหนึ่ง

3.3.5.4 แถบข้อความ

แถบข้อความเป็นเครื่องมือที่แสดงข้อความที่ให้รายละเอียดต่างๆ แก่ผู้ใช้ ได้แก่คำอธิบายของเมนูข้อความแสดงการทำงานในปัจจุบัน และข้อความอื่นๆ รวมทั้งใช้ในการตรวจสอบข้อมูลในโปรแกรมได้ด้วย

การทำงานนั้น แถบข้อความจะเป็นวินโดว์ลูกของโปรแกรมเช่นเดียวกับแถบเครื่องมือ โดยมีตัวแปรร่วมเพื่อใช้ในการเก็บข้อมูลของข้อความที่จะใช้แสดงในแถบข้อความนี้ การใช้งานนั้นจะเรียกฟังก์ชัน MsgBarText เพื่อใช้ในการแสดงข้อความ ซึ่งฟังก์ชันนี้จะนำข้อความที่ระบุนั้นไปเก็บไว้ในตัวแปรที่เป็นข้อความที่จะแสดงผลทางแถบข้อความ แล้วจึงเรียกฟังก์ชันเพื่อส่งเมสเสจ WM_PAINT เพื่อทำการวาดในส่วนของแถบข้อความใหม่โดยใช้ข้อความจะแสดงใหม่ที่เก็บไว้แล้ว ส่วนการเคลียร์ข้อความในแถบข้อความนั้นสามารถทำได้โดยได้เขียนฟังก์ชัน MsgBarClear ซึ่งจะส่งข้อความที่มีขนาดเป็นศูนย์ไปเก็บแล้วสั่งให้วาดใหม่โดยลบพื้น ซึ่งเป็นของเดิมออกด้วย

3.3.6 ส่วนประมวลผลคำสั่ง GRIPS

เมื่อข้อมูลที่จะทำการแสดงผลได้ถูกอ่านเข้ามาเก็บไว้ในโครงสร้างข้อมูลบัฟเฟอร์แล้ว ขั้นต่อไปจะเป็นหน้าที่ของส่วนประมวลผลคำสั่ง GRIPS ซึ่งจะทำการอ่านข้อมูลในบัฟเฟอร์เพื่อนำไปแสดงผล โดยการเรียกใช้ฟังก์ชันที่จัดการบัฟเฟอร์ วนรอบอ่านข้อมูลจากบัฟเฟอร์เข้าทีละเวิร์ดแล้วเทียบค่าของเวิร์ดนี้กับค่าคงที่ ที่กำหนดไว้แทนเพื่อคำสั่งของภาษา GRIPS ที่กำหนดไว้ในโปรโตคอลระดับแอพพลิเคชัน เมื่อตรงกับคำสั่งใดจะดำเนินการอ่านพารามิเตอร์ของคำสั่งนั้นตามโปรโตคอล ซึ่งจะกำหนดไว้ด้วยว่าแต่ละคำสั่งนั้นจะมีพารามิเตอร์กี่ตัว และแต่ละตัวมีการใช้

งานอย่างไร นำพารามิเตอร์นี้ไปผ่านกระบวนการแปลงที่เหมาะสมถ้ามีความจำเป็น เพื่อจะแปลงให้สามารถใช้กับฟังก์ชันกราฟิกของ GDI เพื่อให้เกิดผลลัพธ์ตามคำสั่งของภาษา GRIPS แล้วจึงเรียกใช้คำสั่งนั้นต่อไป

สิ่งสำคัญของส่วนประมวลผลคำสั่ง GRIPS คือเรื่องของการจัดการวัตถุต่างๆ ที่ใช้ในการวาดภาพของฟังก์ชันกราฟิกใน GDI อันได้แก่ ปากกา แปรงสี และฟอนต์ เพราะคำสั่งใน GRIPS นั้นไม่ใช่เป็นการจัดการเปลี่ยนค่าทั้งหมดของแต่ละวัตถุโดยตรง ดังนั้นจึงต้องมีการเก็บโครงสร้างข้อมูลที่เก็บข้อมูลวัตถุในการวาดเหล่านี้ไว้ด้วย เมื่อมีคำสั่งของ GRIPS ที่แก้ไขคุณสมบัติเพียงบางประการของวัตถุ จะได้สามารถเปลี่ยนได้ โดยเปลี่ยนเฉพาะค่าที่ต้องการ แล้วจึงใช้ฟังก์ชันสร้างวัตถุใหม่โดยคุณสมบัติอื่นๆ ยังเหมือนเดิม ซึ่งตามที่ได้ออกแบบนี้จะไม่เก็บแฮนเดิลของวัตถุไว้ คือเมื่อเลือกวัตถุที่สร้างขึ้นใหม่ขึ้นมา ฟังก์ชันจะรีเทินค่าแฮนเดิลของวัตถุเดิมที่ถูกนำไปแทนที่มาให้ด้วย แล้วจึงทำการทำลายวัตถุโดยใช้ค่าแฮนเดิลนี้ทันทีโดยมีลักษณะการใช้ฟังก์ชันดังนี้

```
DeleteObject( SelectObject( DC, CreateObjectIndirect( &Object ) ) );
```

โดย Object นั้นหมายถึง Brush, Pen และ Font โดยมีโครงสร้างข้อมูลแบบ LOGBRUSH, LOGPEN และ LOGFONT ตามลำดับ (LOG คือ Logical)

สำหรับการจัดการคำสั่ง GRIPS ที่เกี่ยวข้องกับการรับข้อมูลกับผู้ใช้ส่งมายังศูนย์บริการ คือคำสั่งที่ใช้ในการจัดการ GRIPS_ACTIVEAREA ซึ่งเป็นบริเวณที่กดแล้วจะส่งการตอบสนอง และ GRIPS_BUTTON ซึ่งคล้ายกันแต่จะมีลักษณะเป็นปุ่มกด การจัดการบริเวณกดนั้นจะพิจารณาคำนวณจากตำแหน่งของตัวชี้เมาส์ กับขอบเขตของบริเวณที่ตอบสนอง ส่วนแบบปุ่มกดนั้นจะจัดเก็บข้อมูลที่เกี่ยวข้องไว้ในโครงสร้างข้อมูลแบบลิงค์ลิสต์ รวมทั้งจะมีการสร้างวินโดว์ลูกของวินโดว์บริเวณวาด ที่มีคลาสเป็นแบบปุ่มกด (button class)

ในการตรวจสอบบริเวณกดนั้น จะดำเนินการใช้ฟังก์ชันประจำวินโดว์ของบริเวณวาด โดยทำการตรวจทุกๆ ครั้งที่เกิดเมสเสจ WM_MOUSEMOVE สำหรับการตรวจสอบตำแหน่งของตัวชี้เมาส์เพื่อใช้ในการเปลี่ยนตัวชี้หรือเคอร์เซอร์ให้เป็นรูปมือชี้ ซึ่งแสดงความแตกต่างระหว่างบริเวณวาดธรรมดากับบริเวณที่กดได้ อีกเมสเสจที่ตรวจสอบคือเมสเสจ WM_LBUTTONDOWN ซึ่งเกิดขึ้นเมื่อมีการกดเมาส์ปุ่มซ้าย ถ้าตรงกับบริเวณกดจึงจะดำเนินการตอบสนองต่อไป

ส่วนปุ่มกดนั้นจะตรวจสอบเมสเสจที่ส่งมาจากแต่ละปุ่มที่เป็นวินโดว์ลูก ในช่วง WM_COMMAND ของฟังก์ชันประจำของวินโดว์แม่ ซึ่งคือวินโดว์ของบริเวณวาด ถ้าตรวจสอบแล้วพบว่าส่งมากจากปุ่มกดใด การตอบสนองจะดึงเอาข้อมูลที่เกี่ยวข้องกับปุ่มนั้น ที่เก็บไว้ในลิงค์ลิสต์มาใช้

นอกจากคำสั่งที่ต้องการการตอบสนองแล้ว คำสั่งที่เกี่ยวข้องกับรูปภาพบิตแมพก็เป็นอีกคำสั่งหนึ่ง ที่ต้องมีการเก็บข้อมูลเกี่ยวกับรูปบิตแมพนั้นไว้ในลิงค์ลิสต์เช่นกัน โดยจะเรียกใช้ในส่วนประมวลผลคำสั่ง GRIPS ซึ่งจะดึงเอาข้อมูลที่จำเป็นในการแสดงรูปบิตแมพนี้ออกมาใช้เพื่อวาดลงในบริเวณวาด

3.3.7 ส่วนจัดการไฟล์

เนื่องจากใช้การควบคุมไฟล์แบบไฟล์แฮนเดิล ดังนั้นการเปิดไฟล์จึงใช้ฟังก์ชัน OpenFile ซึ่งเป็นฟังก์ชันในการจัดไฟล์ของวินโดวส์ ที่สามารถใช้ฟังก์ชันในการอ่านไฟล์แบบดอสที่ใช้ไฟล์แฮนเดิลได้ เช่น read, write, lseek และ _unlink เป็นต้น ซึ่งในการเรียกใช้โดยส่วนจัดการติดต่อสื่อสารแบบไฟล์จะใช้ฟังก์ชันเหล่านี้

ส่วนการอ่านข้อมูลจากไฟล์เพื่อไปเก็บไว้ในหน่วยความจำนน ในส่วนที่เป็นหน่วยความจำแบบรวมนั้นจะใช้ฟังก์ชัน _hread ด้านการปิดไฟล์นั้นใช้ฟังก์ชัน _lclose ซึ่งเป็นฟังก์ชันในการจัดการไฟล์ของวินโดวส์อีกเช่นกัน

นอกจากส่วนที่ใช้ในการดำเนินการกับไฟล์แล้ว ส่วนนี้ยังประกอบไปด้วยฟังก์ชันที่เกี่ยวข้องกับการแยกชื่อไฟล์แบบเต็มออกเป็นส่วนของชื่อไฟล์และพาธ

3.3.8 ส่วนจัดการการติดต่อสื่อสาร

ส่วนจัดการการติดต่อสื่อสารนี้สามารถแยกได้ออกเป็นสองส่วน ตามวิธีที่ใช้ในการติดต่อสื่อสาร คือ ส่วนจัดการการติดต่อสื่อสารผ่านไฟล์ และส่วนจัดการการติดต่อสื่อสารผ่านโมเด็ม สำหรับในแบบแรกนั้นเป็นส่วนที่ไปเรียกใช้ฟังก์ชันที่จัดการไฟล์ เพื่อจัดการไฟล์ทั้งหมดที่ใช้ในการติดต่อระหว่างด้านผู้ขอใช้บริการ และศูนย์ข้อมูลตามข้อกำหนดของโปรโตคอลระดับเชื่อมต่อ และเรียกใช้ฟังก์ชันส่วนจัดการบัฟเฟอร์ เพื่ออ่านข้อมูลจากไฟล์เก็บลงในบัฟเฟอร์ แล้วจึงดำเนินการแปลความหมายของข้อมูลแสดงผล ออกทางบริเวณวาดในโปรแกรม โดยมีส่วนประมวลผลคำสั่งภาษา GRIPS เป็นตัวจัดการ

ส่วนการจัดการติดต่อสื่อสารผ่านโมเด็มนั้น ส่วนหลักคือฟังก์ชันที่ใช้ในการจัดการพอร์ต ประกอบไปด้วยการส่งและรับข้อมูลจากพอร์ต การขอใช้และเลกใช้พอร์ต การเปลี่ยนค่าที่เกี่ยวข้องกับการสื่อสารกับพอร์ตโดยการควบคุมอุปกรณ์โมเด็มนั้นจะเรียกใช้ฟังก์ชันต่างๆ เหล่านี้ในการเริ่มต้นการทำงานของโมเด็ม การหมุนโมเด็ม การตรวจสอบการตอบสนองจากโมเด็ม รวมทั้งการติดต่อสื่อสารอื่นๆ ระหว่างโมเด็มกับพอร์ตซึ่งเป็นคำสั่งในการควบคุมโมเด็ม

นอกจากนี้ ส่วนที่จัดการการสื่อสารแบบโมเด็ม ยังมีความเกี่ยวข้องโดยตรงกับการใช้ และยกเลิกไทเมอร์ (Timer) โดยจะทำการอ่านหรือเขียนข้อมูลไปยังพอร์ต โดยจะใช้ฟังก์ชันกำหนดไทเมอร์ เพื่อให้เกิดเมสเสจ WM_TIMER ไปยังฟังก์ชันประจำวินโดว์ ซึ่งจะมีส่วนที่ตอบสนองต่อเมสเสจนี้ในการอ่านหรือเขียนข้อมูลต่อไป

3.4 โปรแกรมส่วนเทอร์มินัลบนโอเอสทู

เทอร์มินัลบนโอเอสทู (OGTERM, OS/2 GRIPS Terminal) เป็นส่วนแสดงข้อมูล และติดต่อกับผู้ใช้โดยทำงานบนโอเอสทู การทำงาน และวิธีใช้งานส่วนใหญ่ จะเหมือนกับเทอร์มินัลบนวินโดวส์

3.4.1 ภาพรวม

เทอร์มินัลบนโอเอสทู เป็นโปรแกรมที่ทำงานบนพรีเซนเตชันแมเนเจอร์ของโอเอสทู มีหน้าที่ติดต่อไปยังโฮสโดยผ่านทางไฟล์ หรือทางโมเด็ม และติดต่อกับผู้ใช้ หน้าต่างของโปรแกรมจะประกอบไปด้วย 3 ส่วนใหญ่ๆ คือ ทูล

บาร์ (tools bar), ไคลเอนต์แอเรีย (client area) ซึ่งเป็นส่วนที่ใช้แสดงผล และ บรรทัดสถานะ (status line) โดยส่วนไคลเอนต์แอเรีย จะเป็นส่วนที่แสดงรูปกราฟิกที่ส่งมาจากโฮส

โปรแกรมนี้ เขียนขึ้นโดยใช้ภาษาซี ซึ่งได้ใช้คอมไพเลอร์ซีเซตพลัสพลัส (C Set++) ของบริษัทไอบีเอ็ม ซึ่งเป็นคอมไพเลอร์เฉพาะสำหรับการเขียนโปรแกรมภาษาซี หรือซีพลัสพลัส บนโอเอสทู อย่างไรก็ตาม นอกจากซีเซตพลัสพลัสแล้ว ยังมีคอมไพเลอร์อีกหลายโปรแกรมที่สามารถใช้งานได้กับโอเอสทู เช่น บอร์แลนด์ ซีพลัสพลัส (Borland C++), วัตคอมซี (Watcom C) เป็นต้น

ซอร์สไฟล์ของโปรแกรมนี้ ได้แยกออกมาเป็นหลายๆ ไฟล์ด้วยกัน เพื่อความสะดวกในการแก้ไข และความรวดเร็วในการคอมไพล์ เพราะจะทำการคอมไพล์เฉพาะไฟล์ที่มีการแก้ไขเท่านั้น

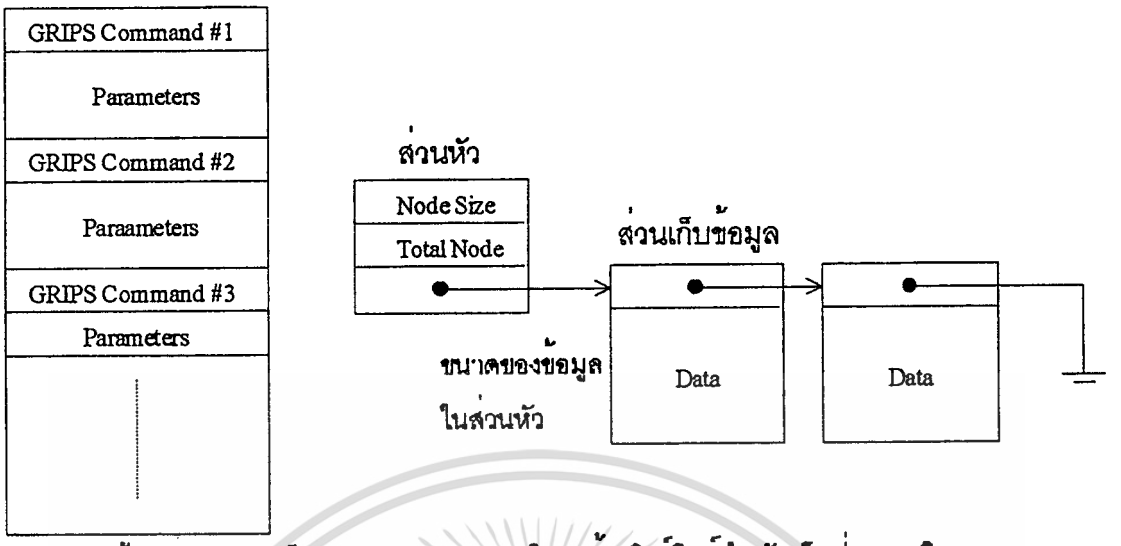
3.4.2 โครงสร้างของโปรแกรม

โปรแกรมเทอร์มินัลบนโอเอสทู ประกอบไปด้วยส่วนต่างๆ หลายส่วนด้วยกัน ซึ่งสามารถแบ่งเป็นกลุ่มๆ ได้ดังนี้

- ส่วนติดต่อกับผู้ใช้ ทำหน้าที่ควบคุมการแสดงผลต่างๆ บนจอภาพ รวมทั้งการรับคำสั่งจากผู้ใช้
- ส่วนจัดการพอร์ตอนุกรม และโมเด็ม มีหน้าที่ควบคุมการใช้งานโมเด็ม ตลอดจนการรับ/ส่งข้อมูลผ่านทางโมเด็ม
- ส่วนจัดการไฟล์ ทำการรับ/ส่งข้อมูลทางไฟล์ อ่านไฟล์
- ส่วนประมวลผลคำสั่ง GRIPS ทำการเก็บข้อมูล และตีความข้อมูล เพื่อนำมาแสดงบนหน้าจอ

3.4.3 การเก็บข้อมูล

การเก็บข้อมูลหน้าจอในโปรแกรม จะเก็บในบัฟเฟอร์ โดยข้อมูลจะเป็นข้อมูลเดียวกับที่ส่งมาจากโฮส นอกจากนี้ โปรแกรมจะแยกข้อมูลส่วน ปุ่ม และบิตแมพ มาเก็บในลิงค์ลิสต์ต่างหากด้วย เพราะต้องเก็บข้อมูลเพิ่มเติมสำหรับการทำงาน ลักษณะการเก็บข้อมูลในโปรแกรมจะเป็นดังรูปที่ 3.14



ก, การเก็บข้อมูลในบัฟเฟอร์ ข, โครงสร้างลิงค์ลิสต์สำหรับเก็บปุ่ม และบิตแมพ

รูปที่ 3.14 การเก็บข้อมูลในโปรแกรม

3.4.4 การติดต่อกับผู้ใช้

ส่วนติดต่อกับผู้ใช้ มีหน้าที่หลักๆ คือวาดหน้าจอ และรับการตอบสนองของผู้ใช้ (หรือการกดปุ่มเมาส์) โดยการวาดหน้าจอ จะมีการย่อ/ขยายวินโดว์, การเลื่อนจอภาพ, การวาดหน้าต่างย่อยในส่วนของปุ่ม รวมถึงการเปลี่ยนรูปร่างของตัวชี้เมาส์ด้วย

การวาดหน้าจอ จะอยู่ในฟังก์ชัน PaintWinProc ซึ่งเป็นวินโดว์โพรซิเจอร์ของหน้าต่างไคลเอนต์แอเรีย โดยการวาดหน้าจอ อยู่ในส่วนที่รับเมสเสจ WM_PAINT ซึ่งเป็นเมสเสจที่บอกให้โปรแกรมทำการวาดหน้าจอใหม่นั่นเอง การทำงานจะเริ่มต้นโดยการกำหนดค่ากราฟิกต่างๆ เช่น สี, รูปแบบตัวอักษร, ฯลฯ จากนั้นก็จะทำการอ่านข้อมูลในบัฟเฟอร์ โดยจะทำการวาดการฟฟิกตามลำดับของคำสั่ง เมื่อวาดกราฟิกจนหมดแล้ว ก็จะทำการคืนทรัพยากรต่างๆ ให้กับระบบ

อย่างไรก็ตาม การประมวลผลตามคำสั่งที่ใช้สร้างปุ่มจะแตกต่างออกไป เพราะปุ่มนั้นจะเป็นวินโดว์ลูกของไคลเอนต์วินโดว์ ซึ่งจะมีวินโดว์โพรซิเจอร์ของตัวเอง ดังนั้น การวาดปุ่มจึงทำโดยอัตโนมัติ

สำหรับการย่อ/ขยายวินโดว์ จะมีเมสเสจ WM_SIZE ไปให้ฟังก์ชัน GripsWindowProc การทำงานก็คือส่งขนาดใหม่ไปให้ทุกๆ วินโดว์ย่อย (ทูลบาร์, ไคลเอนต์แอเรีย และ บรรทัดสถานะ) ซึ่งวินโดว์เหล่านี้ก็จะทำการวาดตัวเองใหม่ ให้เหมาะสมกับขนาดที่เปลี่ยนไป

ส่วนการเลื่อนจอภาพนั้น จะเกิดเมสเสจ WM_HSCROLL หรือ WM_VSCROLL ไปยังฟังก์ชัน PaintWinProc ซึ่งจะมีการคำนวณตำแหน่งในการแสดงผลใหม่ และสั่งให้วาดหน้าจอใหม่ให้ตรงกับขนาดที่เลื่อนไป

ในการเปลี่ยนรูปร่างตัวชี้เมาส์นั้น จะทำการตรวจเมสเสจ WM_MOUSEMOVE ในฟังก์ชัน PaintWinProc ซึ่งจะมีการคำนวณว่าอยู่ในพื้นที่รับการกดหรือไม่ ถ้าใช่ก็จะทำการเปลี่ยนรูปตัวชี้เมาส์ให้เป็นรูปมือชี้ แต่ถ้าไม่อยู่ก็จะเปลี่ยนรูปตัวชี้เมาส์ให้เป็นรูปปกติ

เนื่องจากการอ้างอิงตำแหน่งของกราฟิกในโอเอสทู จะเป็นแบบเวิร์ลด์โคออร์ดิเนต คือ ค่าในแนวแกนนอนเพิ่มขึ้นจากซ้ายไปขวา และค่าในแนวแกนตั้งเพิ่มขึ้นจากล่างขึ้นบน แต่ในคำสั่งของ GRIPS ค่าตำแหน่งจะกำหนดในรูปแบบที่ใช้กันทั่วๆ ไปในทางด้านคอมพิวเตอร์ คือ ค่าในแนวแกนนอนจะเพิ่มขึ้นจากซ้ายไปขวา ซึ่งเหมือนกัน แต่ค่าในแนวแกนตั้งจะเพิ่มขึ้นจากบนลงล่าง ดังนั้น จึงต้องมีการคำนวณตำแหน่งในแนวแกนตั้งทุกครั้ง ที่มีการวาดภาพใหม่

3.4.5 การจัดการพอร์ตอนุกรม และโมเด็ม

ในโอเอสทู มีการเตรียมคำสั่งเกี่ยวกับการจัดการพอร์ตอนุกรม ไว้ให้ใช้อย่างสะดวก การติดต่อกับอุปกรณ์ผ่านทางพอร์ตอนุกรม สามารถทำได้เหมือนกับการอ่าน/เขียนไฟล์บนดิสก์ นอกจากนี้ ยังมีฟังก์ชันที่ใช้สำหรับควบคุมพอร์ตอนุกรมอีกจำนวนหนึ่ง

การติดต่อกับโมเด็ม (หรืออุปกรณ์ใดๆ ที่ติดต่อผ่านทางพอร์ตอนุกรม) จะเริ่มจากการเปิดไฟล์ โดยใช้ฟังก์ชัน DosOpen ซึ่งเป็นคำสั่งเปิดไฟล์ทั่วๆ ไป เพียงแต่ระบุว่าให้ติดต่อกับพอร์ตอนุกรมเท่านั้น จากนั้น ก็สามารถติดต่อกับโมเด็มได้ โดยใช้คำสั่งอ่าน/เขียนไฟล์ธรรมดา คือ DosRead และ DosWrite และเมื่อติดต่อกับพอร์ตเสร็จก็ใช้คำสั่ง DosClose เพื่อปิดไฟล์ สำหรับการควบคุมพอร์ตอนุกรมนั้น จะใช้ฟังก์ชัน DosDevIOCtl ซึ่งเป็นฟังก์ชันที่ใช้ควบคุมอุปกรณ์เชื่อมต่อทั่วๆ ไป รวมทั้งพอร์ตอนุกรมด้วย โปรแกรมสามารถควบคุมการติดต่อกับพอร์ตอนุกรมได้เกือบทุกอย่าง เช่น กำหนดอัตราความเร็วของพอร์ต (bit rate), กำหนดลักษณะของข้อมูลที่จะใช้ติดต่อ, ตรวจว่ามีข้อมูลเข้ามาหรือไม่, ตรวจข้อผิดพลาดของพอร์ต เป็นต้น

สำหรับการใช้งานโมเด็มของโปรแกรม จะทำการเปิดไฟล์ตอนทำการล็อกอินไปยังโฮส และจะทำการส่งคำสั่งหมุนโทรศัพท์ไปยังโมเด็ม หลังจากติดต่อกับโฮสได้แล้ว ก็จะทำการรับ/ส่งข้อมูลโดยใช้ฟังก์ชันที่ได้กล่าวข้างต้น และเมื่อจบการติดต่อ ก็จะสั่งวางหูโทรศัพท์ และปิดไฟล์ที่ติดต่อกับพอร์ต

3.4.6 การจัดการไฟล์

การใช้งานไฟล์ในโปรแกรม มีอยู่หลายที่ด้วยกัน คือ การอ่านค่ากำหนดเบื้องต้น, การเปิดไฟล์, การล็อกอินโดยใช้ไฟล์ และการอ่าน/เขียนไฟล์เก็บรูปบิตแมพ

การใช้งานไฟล์ จะเริ่มจากการเปิดไฟล์โดยใช้คำสั่ง DosOpen ซึ่งจะทำการระบุว่าเป็นการเปิดเพื่ออ่านหรือเปิดเพื่อเขียน รวมทั้งระบุพารามิเตอร์อื่นๆ ด้วย จากนั้นก็ใช้ฟังก์ชัน DosRead และ DosWrite เพื่ออ่าน และเขียนไฟล์ตามลำดับ และเมื่อใช้งานไฟล์เสร็จก็ใช้ฟังก์ชัน DosClose เพื่อปิดไฟล์ นอกจากนี้ ยังมีฟังก์ชันที่ใช้ควบคุมการใช้งานไฟล์อีกด้วย เช่น การหาความยาวของไฟล์ เป็นต้น

3.4.7 การประมวลผลคำสั่ง GRIPS

การประมวลผลคำสั่ง GRIPS ส่วนใหญ่จะกระทำในส่วนของเมสเสจ WM_PAINT ของฟังก์ชัน PaintWinProc ดังได้กล่าวมาแล้ว คือทำการวาดกราฟิกตามลำดับของคำสั่ง อย่างไรก็ตาม ยังมีคำสั่งบางคำสั่งที่

ต้องมีการประมวลผลนอกเหนือจากในฟังก์ชัน PaintWinProc คือ คำสั่งปุ่ม (GRIPS_BUTTON), คำสั่งพื้นที่รับการกด (GRIPS_ACTIVEAREA) และคำสั่งบิตแมพ (GRIPS_BITMAP) โดยคำสั่งปุ่ม และคำสั่งบิตแมพ จะต้องทำงานบางอย่างหลังจากที่นำข้อมูลเข้าสู่บัฟเฟอร์ และโปรแกรมจะต้องทำงานกับคำสั่งปุ่มเมื่อมีการย่อ/ขยายวินโดว์ด้วย ส่วนคำสั่งพื้นที่รับการกด จะต้องทำการตรวจทุกครั้งที่มีการเคลื่อนเมาส์ภายในไคลเอนต์วินโดว์ ซึ่งสามารถทำได้โดยการตรวจเมสเสจ WM_MOUSEMOVE

เมื่อนำข้อมูลเข้าสู่บัฟเฟอร์แล้ว โปรแกรมจะทำการตรวจหาคำสั่งปุ่ม และคำสั่งบิตแมพ โดยถ้าเป็นคำสั่งปุ่ม โปรแกรมจะทำการสร้างวินโดว์ลูก ที่มีลักษณะเป็นปุ่ม พร้อมกับนำข้อมูลควบคุมปุ่มนั้นๆ เก็บในลิงค์ลิสต์สำหรับเก็บข้อมูลปุ่ม ส่วนถ้าเป็นคำสั่งบิตแมพ โปรแกรมจะอ่านไฟล์บิตแมพนั้นเข้าสู่หน่วยความจำ และเก็บข้อมูลควบคุมบิตแมพนั้นในลิงค์ลิสต์สำหรับเก็บข้อมูลบิตแมพ ซึ่งเป็นอีกลิสต์หนึ่ง

เมื่อมีการย่อ/ขยายวินโดว์ จะต้องมีการคำนวณตำแหน่งของแต่ละปุ่มใหม่ เนื่องจากการอ้างอิงตำแหน่งในแนวแกนตั้งของข้อมูล กลับด้านกันกับการอ้างอิงของโอเอสทู ดังได้กล่าวมาแล้ว

สำหรับคำสั่งพื้นที่รับการกด จะต้องมีการตรวจสอบทุกครั้งเมื่อมีการเลื่อนเมาส์ นั่นคือในส่วนเมสเสจ WM_MOUSEMOVE ในฟังก์ชัน PaintWinProc ซึ่งจะมีการคำนวณว่าอยู่ในพื้นที่รับการกดหรือไม่ ถ้าใช่ก็จะทำการเปลี่ยนรูปตัวชี้เมาส์ให้เป็นรูปมือชี้ แต่ถ้าไม่อยู่ก็จะเปลี่ยนรูปตัวชี้เมาส์ให้เป็นรูปปกติ

3.5 โปรแกรมส่วนโฮสบนดอส

โปรแกรมโฮส เป็นโปรแกรมที่ทำการส่งข้อมูลมายังเทอร์มินัล และรับการตอบสนองจากผู้ใช้มาประมวลผล และส่งข้อมูลชุดต่อไปมายังเทอร์มินัลอีก

3.5.1 ภาพรวม

โปรแกรมโฮส เป็นโปรแกรมที่ไม่ต้องติดต่อกับผู้ใช้ ดังนั้นจึงเขียนบนดอสเพื่อความสะดวก และง่ายในการเขียนโปรแกรม โปรแกรมโฮสที่เขียนขึ้นจะเป็นลักษณะทั่วๆ ไป กล่าวคือ สามารถนำไปใช้กับงานใดก็ได้ เพียงแต่เขียนไฟล์ข้อมูลขึ้นมา

3.5.2 การทำงาน

โปรแกรมโฮส จะเริ่มทำงานจากอ่านไฟล์ข้อกำหนดขึ้นมาก่อน ซึ่งในแต่ละแอพพลิเคชันจะไม่เหมือนกัน จากนั้นจะตรวจฐานข้อมูลผู้ใช้ และทำการรอการล็อกอินจากเทอร์มินัล ซึ่งมี 2 แบบ คือ ทางไฟล์ และทางโมเด็ม ดังได้กล่าวมาแล้ว

เมื่อมีผู้ใช้ล็อกอินมา โฮสจะต้องตรวจรหัสผ่านว่าถูกต้องหรือไม่ รวมทั้งตรวจรุ่นของบิตแมพของเทอร์มินัล เพื่อทำการเพิ่มเติมบิตแมพของเทอร์มินัล ถ้าจำเป็น

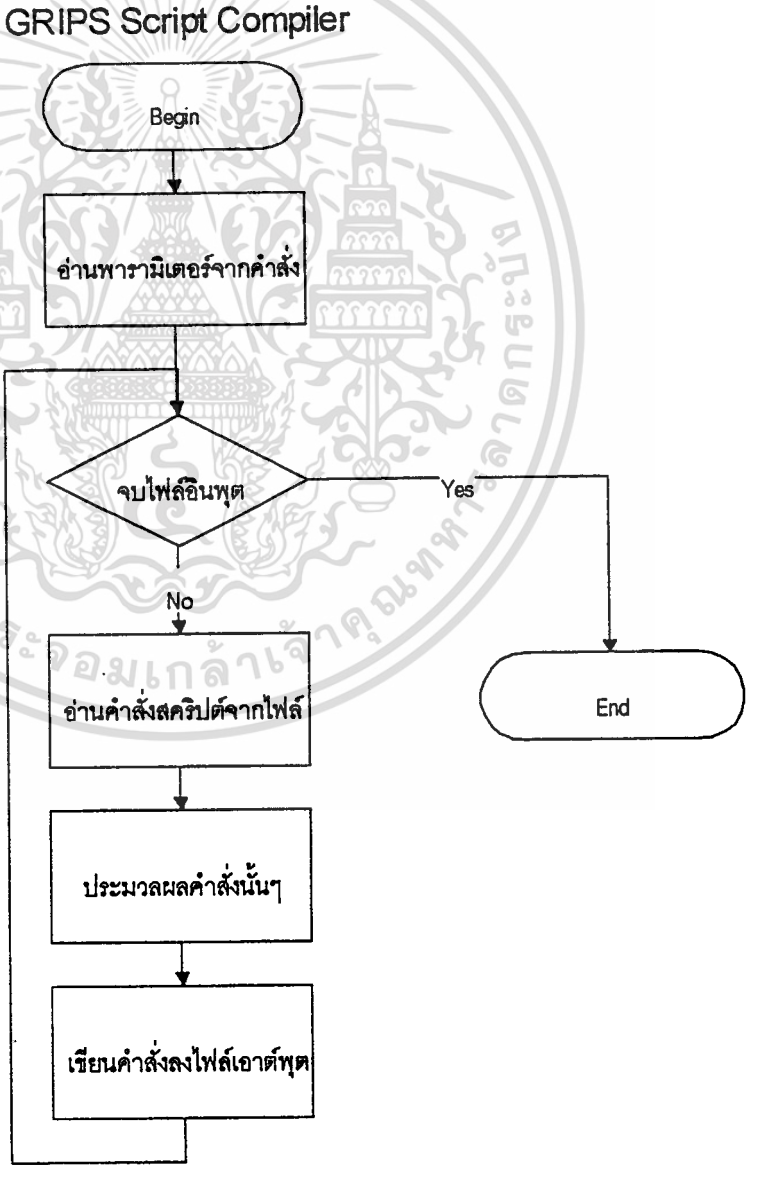
หลังจากการล็อกอินเสร็จเรียบร้อย โฮสก็จะส่งหน้าจอแรกไปยังเทอร์มินัล และรอการตอบสนองจากเทอร์มินัล

3.6 โปรแกรมแปลภาษาสคริปต์ของ GRIPS

โปรแกรมแปลภาษาสคริปต์ของ GRIPS หรือ GSC (GRIPS script compiler) จะใช้สำหรับแปลคำสั่งสคริปต์ ไปเป็นรหัสโปรโตคอลระดับแอพพลิเคชัน

3.6.1 ภาพรวม

ภาษาสคริปต์จะใช้สำหรับการสร้างหน้าจอของแอพพลิเคชันของผู้ใช้ โดยผู้ใช้จะใส่คำสั่งเพื่อวาดรูปต่างๆ บนจอภาพ เช่น เส้นตรง, วงกลม หรือรูปบิตแม็พ เป็นต้น เมื่อผู้ใช้เขียนสคริปต์เสร็จแล้ว จะเก็บอยู่ในไฟล์ที่มีส่วนขยายเป็น .G จากนั้นก็เรียกโปรแกรม GSC เพื่อทำการคอมไพล์สคริปต์ให้เป็นรหัสโปรโตคอลระดับแอพพลิเคชัน ซึ่งจะใช้ส่วนขยายเป็น .GPS



รูปที่ 3.15 การทำงานของโปรแกรม GSC

3.6.2 การทำงาน

การทำงานของโปรแกรม จะเริ่มจากตรวจพารามิเตอร์จากผู้ใช้ ถ้าไม่ถูกต้อง ก็จะอธิบายวิธีการใช้และจบโปรแกรม ถ้าผู้ใช้ป้อนพารามิเตอร์ถูกต้อง ซึ่งก็คือชื่อไฟล์สคริปต์ที่ต้องการนำมาแปล โปรแกรมก็จะอ่านไฟล์สคริปต์นั้นๆ เข้ามา โดยจะอ่านเข้ามาทีละคำสั่ง ซึ่งสามารถแยกเป็นหลายบรรทัดได้ ในกรณีที่ยาวมาก โดยใช้เครื่องหมาย \ ปิดท้ายบรรทัด เพื่อบอกว่าคำสั่งนั้นมีต่อในบรรทัดถัดไป

เมื่ออ่านคำสั่งเข้ามาแล้ว ก็จะทำการแยกส่วนคำสั่ง และพารามิเตอร์ต่างๆ ออกจากกัน แล้วตรวจว่าเป็นคำสั่งที่ถูกต้องหรือไม่ ถ้าถูกต้อง ก็จะเขียนคำสั่งนั้นๆ ลงไฟล์ผลลัพธ์ พร้อมกับพารามิเตอร์ ซึ่งขั้นตอนการอ่านคำสั่งเข้ามา จนเขียนผลลัพธ์นี้ จะทำไปเรื่อยๆ จนกว่าจะจบไฟล์สคริปต์ ซึ่งก็คือทำการแปลเสร็จนั่นเอง

3.7 การออกแบบภาษาสคริปต์

ภาษาสคริปต์ เป็นภาษาที่ใช้สำหรับสร้างแต่ละหน้าจอของโปรแกรม

3.7.1 ภาพรวม

ภาษาสคริปต์ จะเป็นคำสั่งที่ใช้วาดแต่ละกราฟิก โดยแต่ละคำสั่งจะต้องอยู่คนละบรรทัดกัน แต่ในคำสั่งหนึ่งๆ สามารถต่อกันได้หลายบรรทัด โดยใช้เครื่องหมาย \ ปิดท้ายบรรทัด เพื่อบออกว่าคำสั่งนั้นมีต่อในบรรทัดถัดไป

3.7.2 คำสั่งในภาษาสคริปต์

คำสั่งในภาษาสคริปต์ ประกอบไปด้วยคำสั่ง 2 ชนิด คือคำสั่งวาดกราฟิก และคำสั่งที่ใช้ในการควบคุมการวาด คำสั่งสามารถใช้ตัวอักษรตัวเล็ก หรือตัวใหญ่ก็ได้

3.7.2.1 TEXTOUT

แสดงผลข้อความที่ตำแหน่งที่กำหนด ตามรูปแบบ, ลักษณะ, ขนาด และสีที่กำหนดไว้

รูปแบบ

TEXTOUT X Y Text

X-ตำแหน่งในแนวแกนนอน

Y-ตำแหน่งในแนวแกนตั้ง

Text-ข้อความที่ต้องการแสดง

หมายเหตุ

ตำแหน่งที่กำหนด จะเป็นตำแหน่งมุมบนซ้ายของข้อความ

ตัวอย่าง

TEXTOUT 100 150 สวัสดี Hello

3.7.2.2 RECTANGLE

วาดรูปสี่เหลี่ยมมุมฉากที่ตำแหน่ง และมีขนาดตามที่กำหนด

รูปแบบ

RECTANGLE X Y CX CY

X-ตำแหน่งในแนวแกนนอน

Y-ตำแหน่งในแนวแกนตั้ง

CX-ขนาดในแนวแกนนอน

CY-ขนาดในแนวแกนตั้ง

หมายเหตุ

ตำแหน่งที่กำหนด จะเป็นตำแหน่งมุมบนซ้ายของรูปสี่เหลี่ยม

จุดปัจจุบัน จะเป็นตำแหน่งมุมล่างขวาของรูปสี่เหลี่ยม

ตัวอย่าง

RECTANGLE 100 200 50 120

3.7.2.3 LINETO

ลากเส้นจากจุดปัจจุบันไปจุดที่กำหนด

รูปแบบ

LINETO X Y

X-ตำแหน่งในแนวแกนนอน

Y-ตำแหน่งในแนวแกนตั้ง

ลากเส้นจากจุดปัจจุบันไปจุดที่กำหนด โดยใช้สีปัจจุบัน

หมายเหตุ

จุดปัจจุบันของการวาดรูปคือจุดที่กำหนด

ตัวอย่าง

LINETO 150 200

3.7.2.4 MOVETO

กำหนดจุดปัจจุบันใหม่

รูปแบบ

MOVETO X Y

X-ตำแหน่งในแนวแกนนอน

Y-ตำแหน่งในแนวแกนตั้ง

ตัวอย่าง

MOVETO 250 50

3.7.2.5 SETCOLOR

กำหนดสีที่ใช้วาด

รูปแบบ

SETCOLOR R G B

R-ค่าของสีแดง

G-ค่าของสีเขียว

B-ค่าของสีน้ำเงิน

ค่า R, G และ B จะมีค่าตั้งแต่ 0 ถึง 255 โดยค่า 0 หมายถึงไม่มีสีนั้นๆ เลย ค่า 255 หมายถึงสีนั้นๆ สว่างเต็มที่

ตัวอย่าง

SETCOLOR 255 255 0

3.7.2.6 SETFILLCOLOR

กำหนดสีที่ใช้ระบาย

รูปแบบ

SETFILLCOLOR R G B

R-ค่าของสีแดง

G-ค่าของสีเขียว

B-ค่าของสีน้ำเงิน

ค่า R, G และ B เหมือนกับคำสั่ง SETCOLOR (ดูหัวข้อ 3.7.2.6)

ตัวอย่าง

SETFILLLCOLOR 0 255 255

3.7.2.7 POLYLINE

วาดรูปหลายเหลี่ยมโดยไม่ระบายสี

รูปแบบ

POLYLINE N X1 Y1 X2 Y2 X3 Y3 ...

N-จำนวนของมุมของรูปหลายเหลี่ยม

Xn-ค่าในแนวแกนนอนของจุดที่ n

Yn-ค่าในแนวแกนตั้งของจุดที่ n

หมายเหตุ

ตำแหน่งปัจจุบันคือ จุดสุดท้ายในอะเรย์

ตัวอย่าง

POLYLINE 5 100 100 200 150 150 250 50 200 100 100

3.7.2.8 POLYGON

วาดรูปหลายเหลี่ยมโดยรบายสีข้างในด้วย

รูปแบบ

POLYGON N X1 Y1 X2 Y2 X3 Y3 ...

N-จำนวนของมุมของรูปหลายเหลี่ยม

Xn-ค่าในแนวแกนนอนของจุดที่ n

Yn-ค่าในแนวแกนตั้งของจุดที่ n

หมายเหตุ

ตำแหน่งปัจจุบันคือ จุดสุดท้ายในอะเรย์

ตัวอย่าง

POLYGON 4 100 100 200 100 150 187 100 100

3.7.2.9 SETFONTTYPE

กำหนดรูปแบบตัวอักษร

รูปแบบ

SETFONTTYPE N

N-หมายเลขรูปแบบตัวอักษร

โดยหมายเลขรูปแบบตัวอักษรดูได้จากตารางที่ 3.1

ตัวอย่าง

SETFONTTYPE 1

3.7.2.10 SETFONTSIZE

กำหนดขนาดของตัวอักษร

รูปแบบคำสั่ง

SETFONTSIZE CX CY

CX-ความกว้างของตัวอักษร มีหน่วยเป็นจุด

CY-ความสูงของตัวอักษร มีหน่วยเป็นจุด

ตัวอย่าง

SETFONTTYPE 36 48

3.7.2.11 SETFONTSTYLE

กำหนดลักษณะของตัวอักษร

รูปแบบ

SETFONTSTYLE S

S-ลักษณะของตัวอักษร

ค่าของลักษณะตัวอักษรดูได้จากตารางที่ 3.2

ตัวอย่าง

SETFONTSTYLE 3

3.7.2.12 ELLIPSE

วาดรูปวงรีที่มีจุดศูนย์กลาง, รัศมีในแนวแกนตั้ง และรัศมีในแนวแกนนอนที่กำหนด โดยใช้สีปัจจุบัน

รูปแบบ

ELLIPSE X Y RX RY

X-จุดศูนย์กลางในแนวแกนนอน

Y-จุดศูนย์กลางในแนวแกนตั้ง

RX-รัศมีในแนวแกนนอน

RY-รัศมีในแนวแกนตั้ง

ตัวอย่าง

ELLIPSE 350 200 100 50

3.7.2.13 SETFILLSTYLE

กำหนดลักษณะการระบายสี

รูปแบบคำสั่ง

SETFILLSTYLE S

S-ลักษณะของการระบายสี

ลักษณะของการระบายสีดูได้จากตารางที่ 3.3

ตัวอย่าง

SETFILLSTYLE 0

3.7.2.14 BITMAP

แสดงรูปบิตแมพที่กำหนด ที่ตำแหน่งที่กำหนด

รูปแบบคำสั่ง

BITMAP X Y FileName

X-ตำแหน่งในแนวแแกนนอนของมุมบนซ้าย

Y-ตำแหน่งในแนวแแกนตั้งของมุมบนซ้าย

FileName-ชื่อไฟล์รูปบิตแมพที่ต้องการนำมาแสดง

ตัวอย่าง

BITMAP 200 100 THAILAND.BMP

3.7.2.15 BUTTON

สร้างปุ่มที่ตำแหน่งที่กำหนด โดยมีขนาด และข้อความตามที่กำหนด โดยเมื่อผู้ใช้กดปุ่ม โปรแกรมเทอร์มินัลจะส่งการตอบสนองไปยังโฮส ตามที่กำหนด

รูปแบบ

BUTTON X Y CX CY RetStr Text

X-ตำแหน่งในแนวแแกนนอนของมุมบนซ้าย

Y-ตำแหน่งในแนวแแกนตั้งของมุมบนซ้าย

CX-ขนาดในแนวแกนนอน

CY-ขนาดในแนวแกนตั้ง

RetStr-ข้อความตอบสนองสำหรับส่งไปโฮสเมื่อผู้ใช้กดปุ่มนี้

Text-ข้อความที่ต้องการพิมพ์บนปุ่ม

ตัวอย่าง

BUTTON 350 400 100 28 EXIT.GPS OK

3.7.2.16 ACTIVEAREA

สร้างพื้นที่รับการกด เมื่อเลื่อนเมาส์เข้ามาในพื้นที ตัวชี้เมาส์จะเปลี่ยนรูปร่างเป็นรูปมือชี้ เมื่อผู้ใช้กดปุ่มเมาส์ในบริเวณนี้ โปรแกรมเทอร์มินัลจะส่งการตอบสนองไปยังโฮสตามที่กำหนด

รูปแบบ

ACTIVEAREA X Y CX CY RetStr

X-ตำแหน่งในแนวแแกนนอนของมุมบนซ้าย

Y-ตำแหน่งในแนวแแกนตั้งของมุมบนซ้าย

CX-ขนาดในแนวแกนนอน

CY-ขนาดในแนวแกนตั้ง

RetStr-ข้อความตอบสนองสำหรับส่งไปโฮสเมื่อผู้ใช้กดปุ่มเมาส์ในพื้นที่นี้

ตัวอย่าง

ACTIVEAREA 100 200 200 200 102

3.7.3 ตัวอย่างภาษาสคริปต์

```
bitmap 20 20 thai3.bmp
setcolor 208 208 208
setfillcolor 208 208 208
setfillstyle 0
polygon 5 600 40 600 150 350 200 350 90 600 40
setfonttype 10
setfontsize 70 90
setfontstyle 1
setcolor 0 0 0
textout 303 53 พยากรณ์อากาศ
textout 503 103 ประจำวัน
setcolor 255 0 0
textout 300 50 พยากรณ์อากาศ
setcolor 0 0 128
textout 500 100 ประจำวัน
setcolor 0 0 255
setfonttype 5
setfontsize 40 40
setfontstyle 0
textout 300 320 ยินดีต้อนรับสู่ระบบพยากรณ์อากาศแบบ Online
setfontsize 30 30
setcolor 255 0 255
textout 320 360 คุณสามารถเลือกการพยากรณ์โดยการใช้เมาส์
textout 320 390 ไปกดที่ภาคที่ต้องการในแผนที่
button 580 430 100 28 106 Exit
setfillstyle 1
link 50 20 100 130 101
link 70 150 70 80 102
link 140 80 130 120 103
link 140 200 60 70 104
link 70 230 50 120 105
link 60 350 120 105 105
```

บทที่ 4

การทดลองและผลการทดลอง

4.1 การติดต่อโดยใช้ไฟล์

การติดต่อโดยใช้ไฟล์ โดยมากจะใช้ในการติดต่อผ่านระบบแลน ซึ่งจะมีระบบปฏิบัติการควบคุมการใช้งานไฟล์อยู่ เช่น เน็ตแวร์ (NetWare) การควบคุมการใช้งานไฟล์มีผลให้การติดต่อกับไฟล์พร้อมๆ กันโดย 2 โปรแกรม มีปัญหา เช่น ไม่สามารถลบไฟล์ได้, ไม่สามารถอ่านไฟล์ได้, ไม่สามารถสร้างไฟล์ได้ เป็นต้น นอกจากนั้น การใช้งานไฟล์ร่วมกันจากระบบปฏิบัติการที่ต่างกัน ยังทำให้เกิดปัญหาด้วย เพราะแต่ละระบบปฏิบัติการมีการใช้งานไฟล์ไม่เหมือนกัน

ดังนั้น การติดต่อกันโดยใช้ไฟล์จึงไม่สามารถใช้วิธีเขียน/อ่านไฟล์ตามปกติได้ สำหรับในโครงงานนี้ ได้ออกแบบการใช้งานไฟล์ โดยใช้ไฟล์ 4 ไฟล์ดังได้กล่าวมาแล้วข้างต้น ซึ่งจะแก้ปัญหานี้ได้ เพราะโปรโตคอลนี้ จะประกันว่าก่อนที่โปรแกรมฝ่ายรับจะเปิดไฟล์ข้อมูลนั้น โปรแกรมฝ่ายเขียนจะปิดไฟล์นั้นๆ แล้วเสมอ สำหรับการลบไฟล์หลังจากอ่านนั้น ก็เพื่อเตรียมพร้อมสำหรับการส่งข้อมูลครั้งต่อไป

จากการทดลองพบว่า การใช้โปรโตคอลนี้ ไม่พบว่ามีปัญหาใดๆ ทั้งการทำงานบนฮาร์ดดิสก์ หรือการทำงานบนระบบแลน ทั้งบนโอเอสทู และวินโดวส์

4.2 การติดต่อโดยใช้โมเด็ม

ในส่วนนี้สำหรับดอส ในขั้นแรกเป็นการเขียนโปรแกรมให้รับและส่งข้อมูลผ่านทางพอร์ตอนุกรม ซึ่งจะต้องมีการกำหนดค่าเริ่มต้นให้พอร์ตอนุกรมเสียก่อน โดยกำหนดความเร็วของข้อมูล ขนาดของข้อมูล จำนวนบิตเริ่มต้น จำนวนบิตหยุด และจำนวนบิตพาริตี้ หลังจากนั้นก็สามารถที่จะรับส่งข้อมูลผ่านทางพอร์ตอนุกรมได้ ในการส่งข้อมูลนั้นจะต้องตรวจสอบว่าพอร์ตอนุกรมนั้นว่าง หรือ พร้อมที่จะส่งหรือไม่ ก่อนที่จะส่งข้อมูลออกไป ส่วนการรับข้อมูลนั้นสามารถทำได้สองวิธี ตามที่ได้กล่าวมาแล้วคือ แบบโพลลิง และแบบอินเทอร์รัพต์ แต่ที่ใช้ในโครงงานนี้ จะใช้แบบอินเทอร์รัพต์ ในการใช้งานแบบอินเทอร์รัพต์จำเป็นจะต้องมีการกระทำบางอย่างก่อนจึงจะสามารถใช้ได้ ได้แก่ การกำหนดให้เป็นการรับแบบอินเทอร์รัพต์ โดยการเซตบิตที่ 0 ของพอร์ตอนุกรมที่เกี่ยวกับการจัดการอินเทอร์รัพต์ (IER) แล้วทำการอ่านข้อมูลจากพอร์ตอนุกรมทิ้ง เพื่อเป็นการขจัดข้อมูลที่ค้างอยู่ ซึ่งอาจจะทำให้ไม่สามารถเกิดอินเทอร์รัพต์ หลังจากนั้นต้องทำการกำหนดให้สามารถเกิดอินเทอร์รัพต์ที่เกี่ยวกับพอร์ตอนุกรม โดยบนคอมพิวเตอร์ส่วนบุคคลได้กำหนดให้ IRQ3 และ IRQ4 เป็นฮาร์ดแวร์อินเทอร์รัพต์ที่เกี่ยวข้องกับพอร์ตอนุกรม จะทำการเซตอินเทอร์รัพต์มาสค์เอเบิลรีจิสเตอร์ (Interrupt Maskable Register) เพื่อให้ตอบสนองต่อสัญญาณอินเทอร์รัพต์ IRQ3

หรือ IRQ4 แล้วแต่พอร์ตอนุกรมที่จะใช้ ต่อมาจะต้องเตรียมส่วนของการทำงานได้รับรอง เมื่อเกิดอินเทอร์รัพต์ขึ้น โดยจะนำข้อมูลที่ได้ไปเก็บไว้ในบัฟเฟอร์ที่ได้เตรียมไว้ เป็นบัฟเฟอร์แบบวงแหวน หลังจากทำตามขั้นตอนดังกล่าวแล้วก็สามารถรับส่งข้อมูลผ่านทางพอร์ตอนุกรมโดยการใช้อินเทอร์รัพต์ โปรแกรมที่เขียนทดสอบเป็นโปรแกรมเทอร์มินัลธรรมดาที่สามารถจะรับและส่งข้อมูลได้

ในขั้นที่สองเป็นการติดต่อกับโมเด็ม เมื่อสามารถส่งข้อมูลออกทางพอร์ตอนุกรมได้แล้ว ที่เหลือก็เพียงแค่รู้จักการใช้คำสั่งติดต่อกับโมเด็มเท่านั้น ส่วนแรกคือการกำหนดค่าเริ่มต้นให้โมเด็ม จะใช้คำสั่งง่ายๆ คือ ATZ ถ้าเป็นแฟกซ์โมเด็มจะต้องใช้คำสั่งในการยกเลิกการควบคุมการไหลโดยใช้ฮาร์ดแวร์ด้วย หลังจากนั้นก็เป็นการหมุนโทรศัพท์ โดยใช้คำสั่ง ATDT และตามด้วยหมายเลขโทรศัพท์ เมื่อโทรศัพท์เป็นระบบกดปุ่ม หรือ คำสั่ง ATDP และตามด้วยหมายเลขโทรศัพท์ เมื่อโทรศัพท์เป็นระบบหมุน หลังจากนั้นก็รอการตอบสนองว่าติดต่อกับปลายทางได้หรือไม่ ถ้าโมเด็มตอบรับว่า "CONNECT" ก็แสดงว่าสามารถติดต่อกับด้านปลายทาง ส่วนการยกเลิกการติดต่อจะใช้การยกเลิกโดยแบบใช้ฮาร์ดแวร์ช่วย ซึ่งได้กล่าวรายละเอียดไว้แล้ว จะไม่ขอกล่าวซ้ำอีกโปรแกรมที่เขียนทดสอบสามารถที่จะติอต่อกับระบบยูนิกซ์ (Unix) ได้ผ่านทางโมเด็ม

4.3 โปรโตคอลระดับการติดต่อสื่อสาร

ในการทดสอบส่วนนี้ ในขั้นแรกทดสอบโดยใช้สาย RS-232 ก่อน โดยให้ส่งข้อมูลจำนวนหนึ่งแพ็คเกจ พบว่ามีข้อผิดพลาดเกิดขึ้นจากการเข้ารหัสซีอาร์ซีผิด หลังจากแก้ไขก็สามารถส่งได้ การหาความผิดพลาดไม่สามารถใช้วิธีดีบักโปรแกรมแบบที่ใช้ทั่วไปได้ เพราะอินเทอร์รัพต์จะทำให้ระบบหยุดชะงักได้ จึงต้องใช้วิธีพิมพ์ออกที่หน้าจอแทน เพื่อดูว่าเกิดการผิดพลาดขึ้นที่ใด ต่อมาทำการเขียนส่วนนี้ใหม่ แต่ยังใช้แนวความคิดเดิม และได้ทำการทดสอบ ผลปรากฏว่าไม่มีปัญหาใดๆ มีการเปลี่ยนแปลงบางส่วนในส่วนการทำงานนี้ โดยเพิ่มส่วนการบอกขนาดทั้งหมดของข้อมูลที่จะส่งก่อน และการส่งข้อมูลจะส่งตามไปทันทีหลังจากที่ส่งไปบอกขนาดของแพ็คเกจแล้ว การทดสอบไม่มีปัญหาใดๆ

ในขั้นตอนที่สอง ทดสอบการส่งข้อมูลผ่านทางสายโทรศัพท์ โดยเขียนโปรแกรมสำหรับรับส่งไฟล์ข้อมูลจากที่หนึ่งไปอีกที่หนึ่งและใช้โปรโตคอลนี้ในการรับส่ง พบว่าจะต้องกำหนดเวลาในการตรวจสอบให้มากขึ้น หลังจากแก้ไขก็สามารถส่งไฟล์ได้อย่างถูกต้อง ไฟล์ขนาด 10 กิโลไบต์ใช้เวลาส่งประมาณ 12 วินาที ที่ความเร็ว 2400 บิตต่อวินาที

ในขั้นตอนที่สาม นำโปรโตคอลไปฝังไว้ในตัวโฮสต์และเทอร์มินัลทั้งโอเอสทูและวินโดวส์ โดยให้ตัวเทอร์มินัล และโฮสต์ติดต่อรับส่งข้อมูลระหว่างกันแล้วนำมาแสดงผล ผลปรากฏว่าไม่มีปัญหาใดๆ

4.4 โปรโตคอลระดับแอพพลิเคชัน

โปรโตคอลระดับแอพพลิเคชัน ได้สร้างขึ้นโดยอาศัยแนวคิดของโปรโตคอล RIP (Remote Information Protocol) และโปรโตคอล HTML (HyperText Markup Language) แต่ได้สร้างขึ้นโดยการเก็บข้อมูลเป็นแบบไบนารี แทนที่จะเป็นแบบแอสกี ดังนั้น ปัญหาของโปรโตคอลนี้ จะอยู่ที่สัญญาณรบกวน ถ้าข้อมูลมีข้อผิดพลาด อาจ

ทำให้หน้าจอมีการผิดพลาดไปมาก ดังนั้นจึงต้องออกแบบโปรโตคอลระดับการติดต่อสื่อสารให้รัดกุม เพื่อไม่ให้มีข้อผิดพลาดได้

สำหรับสิ่งที่สำคัญอีกอย่างของโปรโตคอล คือ การส่งข้อมูลที่เป็นคำสั่งกราฟิก แทนที่จะส่งบิตแมพไปทั้งหน้าจอ ซึ่งทำให้สามารถลดขนาดของข้อมูลลงไปได้มาก ส่วนการส่งบิตแมพนั้น จะใช้วิธีส่งไปเก็บก่อน

อย่างไรก็ตาม โปรโตคอลที่สร้างขึ้น สามารถใช้งานได้ดีพอสมควร รูปของหน้าจอหนึ่งๆ มีขนาดประมาณ 0.5-5 กิโลไบต์ ซึ่งนับว่าเล็กมาก และข้อมูลปริมาณนี้ ถ้าส่งผ่านโมเด็มที่ความเร็ว 14.4 กิโลบิต/วินาที จะใช้เวลาเพียง 0.5-4 วินาทีเท่านั้น

4.5 โฮส

การสร้างโฮส ประกอบไปด้วย 2 ขั้นตอน คือ การสร้างโฮสที่ทำการติดต่อผ่านทางไฟล์ และการขยายความสามารถ ให้สามารถติดต่อผ่านทางโมเด็มได้ เพราะการติดต่อผ่านทางไฟล์ สามารถทำได้ง่ายกว่า เมื่อทดสอบการติดต่อกับเทอร์มินัลผ่านทางไฟล์ จนได้ผลลัพธ์เป็นที่พอใจแล้ว จึงนำเอาโปรโตคอลระดับการติดต่อสื่อสารแบบโมเด็มมารวมเข้าไป ซึ่งต้องปรับปรุงการทำงานบางอย่างด้วย

4.6 เทอร์มินัลบนโอเอสทู

ดังที่ได้กล่าวมาแล้วว่า โปรแกรมเทอร์มินัลบนโอเอสทู แบ่งออกเป็นส่วนย่อยๆ ซึ่งการทดสอบก็จะแบ่งทดสอบในส่วนย่อยๆ แต่ละส่วนก่อน จึงจะนำมารวมกัน โดยส่วนแรกที่ทำการทดสอบ คือส่วนประมวลผลคำสั่งภาษา GRIPS ซึ่งจะทำการทดสอบทีละคำสั่งเช่นกัน ปัญหาหลักในที่นี้ ก็คือการอ้างตำแหน่งบนจอภาพในแนวแกนตั้ง ซึ่งในโอเอสทูค่าตำแหน่งจะเพิ่มขึ้นจากล่างขึ้นบน แต่ในโปรแกรมคอมพิวเตอร์อื่นๆ รวมทั้งวินโดวส์ ค่าตำแหน่งในแนวแกนตั้งจะเพิ่มขึ้นจากบนลงล่าง จึงทำให้ต้องแก้ไขค่าตำแหน่งในแนวแกนตั้งทุกครั้งที่ทำการแสดงผล

เมื่อทดสอบส่วนแรกเสร็จแล้ว ก็ทดสอบส่วนติดต่อกับไฟล์ โดยทดลองอ่านไฟล์ข้อมูลภาษา GRIPS ขึ้นมา และทดลองติดต่อไปยังโฮสโดยใช้ไฟล์ พบว่าโปรแกรมสามารถทำงานได้เป็นอย่างดี

ขั้นตอนสุดท้ายของการทดสอบ คือ การติดต่อกับโมเด็มผ่านทางพอร์ตอนุกรม และการควบคุมโมเด็ม รวมทั้งการติดต่อไปยังโฮสผ่านทางโมเด็ม ซึ่งมีข้อแตกต่างกับการติดต่อผ่านทางไฟล์บ้าง คือต้องมีการควบคุมโมเด็ม, การหมุนโทรศัพท์ไปยังโฮส เป็นต้น

4.7 เทอร์มินัลบนวินโดวส์

การทดสอบส่วนเทอร์มินัลบนวินโดวส์นี้ได้ทำการทดสอบเป็นส่วนๆ ตามส่วนประกอบสำคัญที่เป็นองค์ประกอบของโปรแกรมก่อน ได้แก่ส่วนจัดการบัฟเฟอร์ ซึ่งนับว่ามีความสำคัญและมีบทบาทเป็นอย่างมากในการทำงานของส่วนอื่นๆ ด้วย โดยจะต้องทดสอบรวมกับส่วนที่อ่านไฟล์เข้ามาเก็บไว้ในบัฟเฟอร์นี้ แล้วตรวจสอบว่าข้อมูลนั้นถูกต้องหรือไม่ เมื่อในไปรวมกับส่วนที่ทำหน้าที่ประมวลผลคำสั่งภาษา GRIPS แล้วจึงสามารถทดสอบการทำงานของทั้งสามส่วนได้ โดยการอ่านไฟล์ที่เก็บข้อมูลในรูปคำสั่งภาษา GRIPS แบบไบนารีที่สร้างขึ้นมาเพื่อใช้ใน

การทดสอบ แล้วดูผลลัพธ์ที่ปรากฏขึ้นนั้นว่าตรงกับที่กำหนดไปหรือไม่ นอกจากใช้วิธีการอ่านไฟล์มาแสดงเพื่อทดสอบความถูกต้องในการทำงานของส่วนต่างๆ ของโปรแกรมแล้ว ยังต้องเทียบผลลัพธ์ที่ได้ในระบบวินโดวส์กับระบบ โอเอสทูด้วยว่ามีความแตกต่างกันหรือไม่ แค่ไหน เกิดขึ้นจากอะไรแล้วทำการปรับให้ใกล้เคียงกันมากที่สุด

หลังจากที่ทดสอบหน้าที่การทำงานพื้นฐานหลักของส่วนต่างๆ ของโปรแกรมแล้ว ขั้นต่อไปจึงเริ่มทำการทดสอบในกรณีที่ใช้แสดงผลข้อมูลที่ได้มาจากการติดต่อสื่อสาร เริ่มต้นด้วยการติดต่อผ่านไฟล์ ซึ่งจะทดสอบส่วนที่ทำการจัดการไฟล์ในการติดต่อสื่อสาร การเชื่อมการติดต่อระหว่างโปรแกรมเทอร์มินัลกับโฮส การขอเข้าใช้บริการการปรับความสัมพันธ์กันระหว่างโปรแกรมเทอร์มินัลกับโฮส จนเมื่อผ่านการติดต่อสื่อสารเรียบร้อยจึงเทียบข้อมูลผลลัพธ์ที่ได้กับเมื่ออ่านจากไฟล์ตรงๆ แบบปกติ รวมทั้งทดสอบการตอบสนองต่อส่วนติดต่อกับผู้ใช้ว่าให้ค่าถูกต้องเป็นไปตามที่กำหนด ซึ่งค่านี้ทางด้านโฮสจะใช้ในการพิจารณาส่งข้อมูลส่วนต่อไป

เมื่อผ่านการตรวจสอบการทำงานส่วนนี้แล้ว จึงเป็นเรื่องของการขอยกเลิกการติดต่อในแบบไฟล์ ซึ่งจะต้องดูผลลัพธ์ทางด้านโฮสด้วยว่ารับรู้ถึงการขอยกเลิกการติดต่อนี้หรือไม่

หลักจากนี้จะเป็นการทดสอบส่วนที่จัดการการสื่อสารที่ต้องติดต่อกับพอร์ตโดยทดสอบเขียนและอ่านข้อมูลจากพอร์ตเพื่อไปควบคุมโมเด็ม ซึ่งสามารถทดสอบได้ง่าย เช่นส่งคำสั่งในการหมุนโทรศัพท์ไป ถ้าการจัดการส่วนนี้ถูกต้อง โมเด็มจะต้องทำงานหมุนโทรศัพท์ให้ พอเสร็จขั้นตอนการทดสอบรับส่งข้อมูลระหว่างพอร์ตแล้วต่อไปจะต้องทำการทดสอบส่วนติดต่อสื่อสาร ที่ใช้โปรโตคอลการติดต่อสื่อสารระดับเชื่อมต่อ โดยรวมส่วนของการจัดการเข้าไปในซอร์สโปรแกรม แล้วทดสอบการรับส่งข้อมูลผ่านเครือข่ายโทรศัพท์ ซึ่งมีโฮสอยู่ทางด้านรับแล้วตรวจสอบผลลัพธ์ว่าเหมือนกับกรณีที่ติดต่อแบบไฟล์หรือไม่ รวมทั้งถ้ามีปัญหาที่เกิดขึ้นจากการสื่อสารจะส่งผลอย่างไรต่อการติดต่อสื่อสารที่กำลังดำเนินอยู่ เพื่อจะได้นำไปเป็นข้อมูลในการปรับปรุงโปรแกรมต่อไป

บทที่ 5
บทวิจารณ์และสรุป

5.1 อุปสรรคในการพัฒนา

ในการทำโครงงานครั้งนี้มีเป้าหมายพัฒนาโปรแกรมประมวลผลสารสนเทศทางไกลในสภาพแวดล้อมแบบกราฟิก เมื่อพิจารณาแล้วจะเห็นได้ว่า อุปสรรคสำคัญอย่างหนึ่งคือความเข้ากันได้ของระบบจัดการสภาพแวดล้อมแบบกราฟิกที่ต่างกันสองระบบคือวินโดวส์และโอเอสทู ซึ่งเป็นต้นเหตุสำคัญของอุปสรรคต่างๆ ในการพัฒนาครั้งนี้ แต่อย่างไรก็ตาม โครงการพัฒนาครั้งนี้จะประสบความสำเร็จไปได้ด้วยดีในที่สุด ในส่วนต่อไปนี้จะกล่าวถึงอุปสรรคสำคัญในการพัฒนาแยกเป็นข้อๆ ดังนี้

5.1.1 ปัญหาเรื่องเครื่องมือในการพัฒนา

เป็นธรรมดาของเครื่องมือในการพัฒนาโปรแกรมในระบบกราฟิกเหล่านี้ ซึ่งต้องการทรัพยากรของระบบฮาร์ดแวร์สูงมาก โดยเฉพาะโอเอสทูที่เป็นระบบปฏิบัติการด้วยแล้ว ยิ่งต้องการเครื่องคอมพิวเตอร์ที่มีสมรรถนะสูง จึงทำให้ระบบที่ใช้ในการพัฒนาเกิดการขาดแคลนได้ นอกจากนั้นซอฟต์แวร์ที่ใช้นั้น ต่างก็เป็นซอฟต์แวร์ที่มีขนาดใหญ่มาก ต้องการเนื้อที่ฮาร์ดดิสก์ในการติดตั้งสูง ทำให้มีระบบที่ใช้จำกัด เป็นอุปสรรคในการเสียเวลาในการติดตั้ง การปรับค่า การศึกษาการใช้งานไปค่อนข้างมาก

5.1.2 ปัญหาเรื่องเสถียรภาพของระบบ

ทั้งระบบวินโดวส์และโอเอสทูต่างก็ปัญหาเช่นเดียวกัน นั่นคือระบบมักจะหยุดการทำงานไปดื้อๆ โดยไม่สามารถระบุสาเหตุได้บ่อยๆ บางครั้งนั้นร้ายแรงถึงขั้นที่ต้องติดตั้งใหม่หมดทั้งระบบ ยิ่งโดยเฉพาะการเขียนโปรแกรมที่ใช้ฟังก์ชันในการจัดการต่างๆ ของระบบ ถ้าใช้ผิดพลาดเพียงเล็กน้อยอาจจะก่อให้เกิดความเสียหายเป็นอย่างมากได้

5.1.3 ความแตกต่างของระบบที่พัฒนา

โดยส่วนใหญ่แล้วระบบวินโดวส์และโอเอสทูนั้นจะมีส่วนการจัดการโดยทั่วไปคล้ายกัน แต่ในบางเรื่องนั้นจะต่างกันโดยสิ้นเชิง เช่นพิกัดที่ใช้วินโดว์ที่ใช้ต่างกัน ในวินโดวส์นั้นปกติจะมีจุดเริ่มต้นอยู่ที่มุมบนด้านซ้ายของวินโดว์ แต่ในโอเอสทูนั้นมีจุดอ้างอิงเริ่มต้นอยู่มุมล่างด้านซ้าย อีกทั้งยังมีทิศทางกลับกันด้วย หลักการที่แตกต่างกัน

อย่างมากนั้นได้แก่การจัดการการแสดงผล ระหว่าง GDI กับ GPI ซึ่งมีหลักการต่างกันเป็นอย่างมาก ทำให้การพัฒนาต้องพิจารณาในเรื่องนี้ด้วย นอกจากความแตกต่างแล้ว ข้อด้อยและข้อได้เปรียบของแต่ละระบบยังเป็นอุปสรรคสำคัญอีกประการหนึ่ง บางครั้งระบบวินโดวส์ก็ได้เปรียบในการจัดการเรื่องหนึ่งที่โอเอสทูจัดการได้ไม่ดีเช่น การจัดการไฟล์เริ่มต้นของโปรแกรม (INI, initial file) เพราะมีฟังก์ชันให้มาเลย แต่โอเอสทูไม่มี บางเรื่องโอเอสทูก็ได้เปรียบมากกว่าเช่นการจัดการบริหารหน่วยความจำซึ่งมีการจัดการได้ดีมากกว่า เพราะโครงสร้างของระบบปฏิบัติการเป็นแบบ 32 บิตอย่างแท้จริง ไม่เหมือนกับวินโดวส์ที่ยังคงเป็น 16 บิตอยู่ ซ้ำยังมีการจัดการทรัพยากรของระบบที่ต้องประนีประนอมกับแอพพลิเคชันอีกด้วย

5.2 แนวทางการปรับปรุงและพัฒนาต่อไป

การพัฒนาต่อเนื่องที่จะเพิ่มความสมบูรณ์แก่โครงงานนี้มากยิ่งขึ้น สามารถทำได้หลายประการ ได้แก่ การเพิ่มส่วนติดต่อกับผู้ใช้แบบอื่นๆ เข้าไปในโปรโตคอลของแอพพลิเคชัน การเพิ่มส่วนที่จัดการกราฟิกให้มีคำสั่งมากขึ้น หรืออาจจะเพิ่มการสนับสนุนสื่อในรูปแบบอื่นๆ แบบมัลติมีเดีย เช่น เสียง ภาพเคลื่อนไหว

5.3 ข้อเสนอแนะ

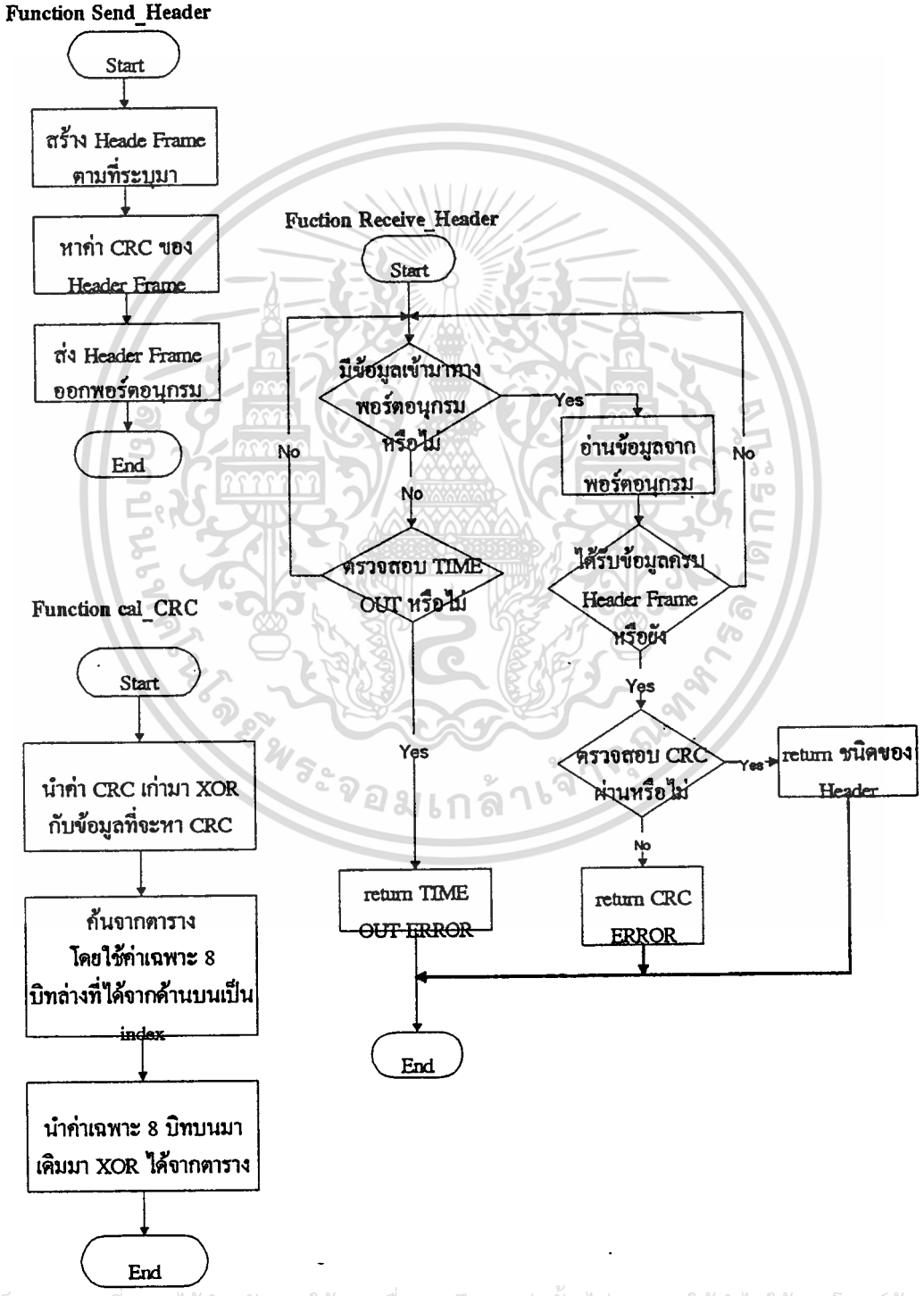
โครงงานนี้ สามารถจัดสร้างเป็นระบบสารสนเทศแบบออนไลน์ได้จริง หากแต่ยังขาดโปรแกรมช่วยในการทำงาน เช่น การสร้างหน้าจอ, การจัดการฐานข้อมูล เป็นต้น ถ้ามีการเขียนโปรแกรมต่างๆ เหล่านี้ ก็จะทำให้ระบบสมบูรณ์ยิ่งขึ้น นอกจากนี้ ยังสามารถพัฒนาระบบให้เข้ากันได้ กับระบบสารสนเทศที่มีใช้งานอยู่แล้ว เช่น WWW, Gopher เป็นต้น

5.4 สรุป

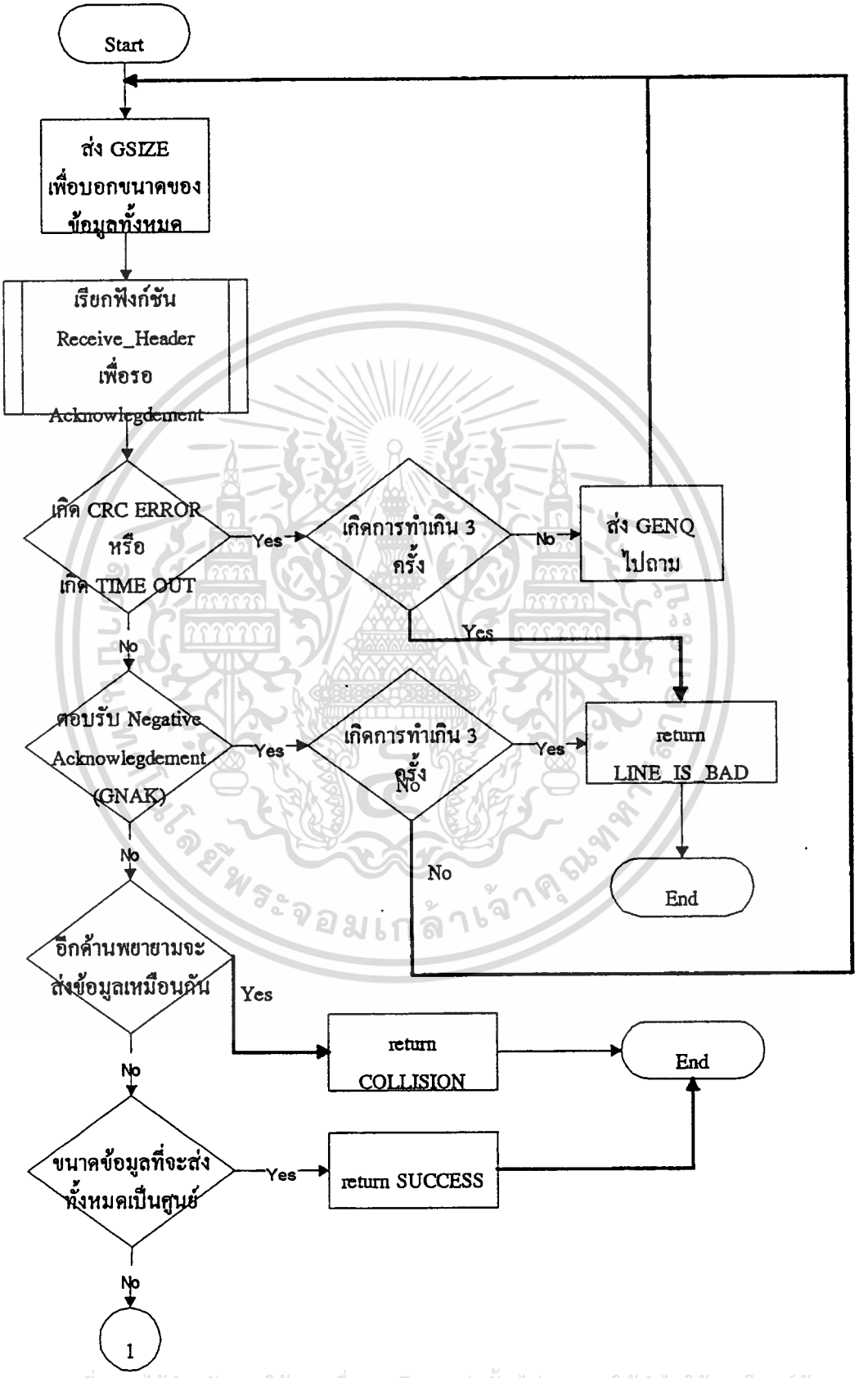
การจัดทำโครงงานชิ้นนี้ เป็นการเสนอแนวความคิดใหม่ ในระบบสารสนเทศ โดยนำเอาแนวคิดของหลายระบบที่มีอยู่ในท้องตลาดมาปรับปรุง ทำให้โครงงานสมบูรณ์ยิ่งขึ้น อย่างไรก็ตาม โครงงานชิ้นนี้จะมีคุณค่ามากขึ้น ถ้ามีการนำไปใช้งานจริง

ภาคผนวก

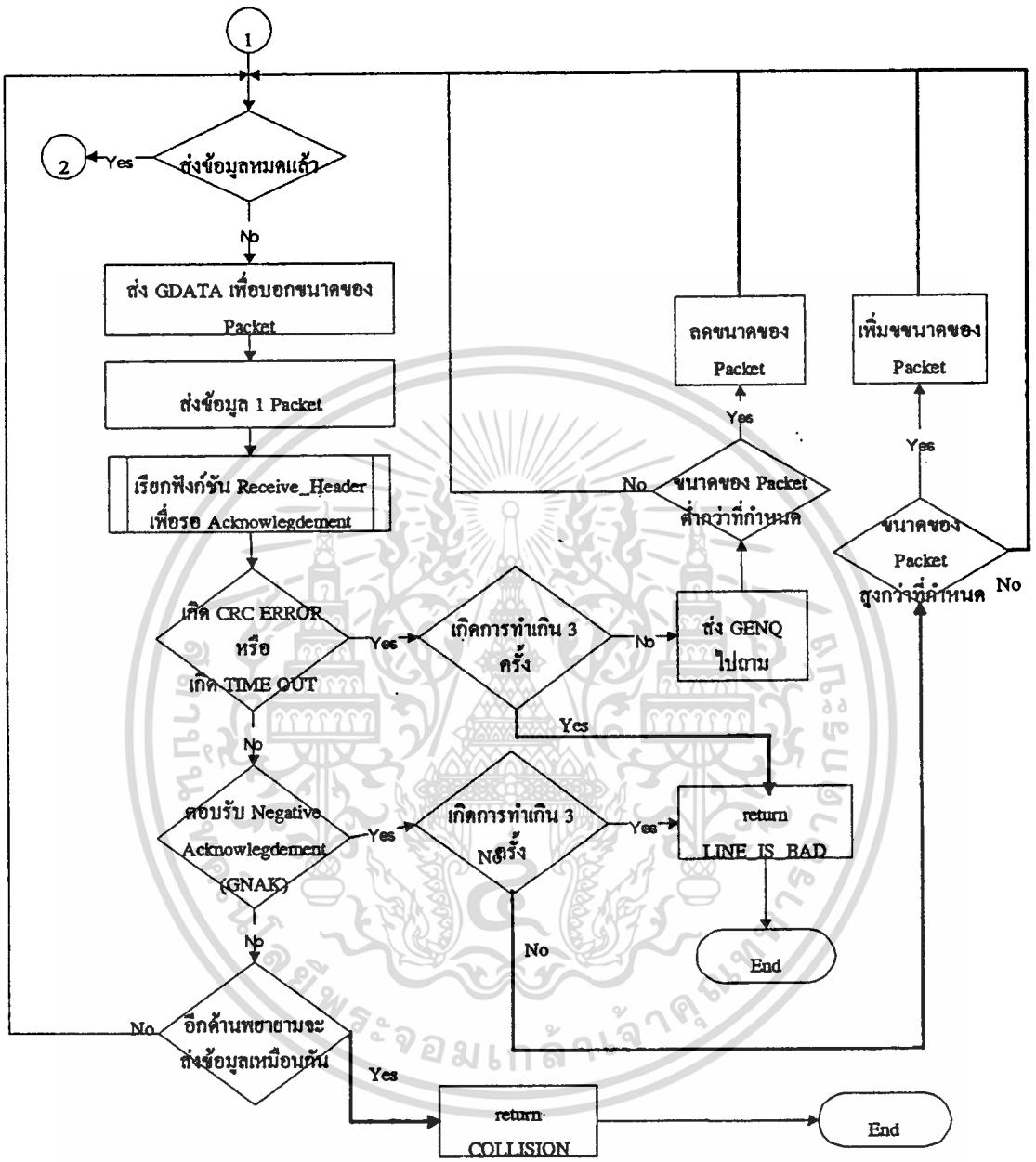
โฟลว์ชาร์ตส่วนโปรโตคอลระดับการสื่อสาร



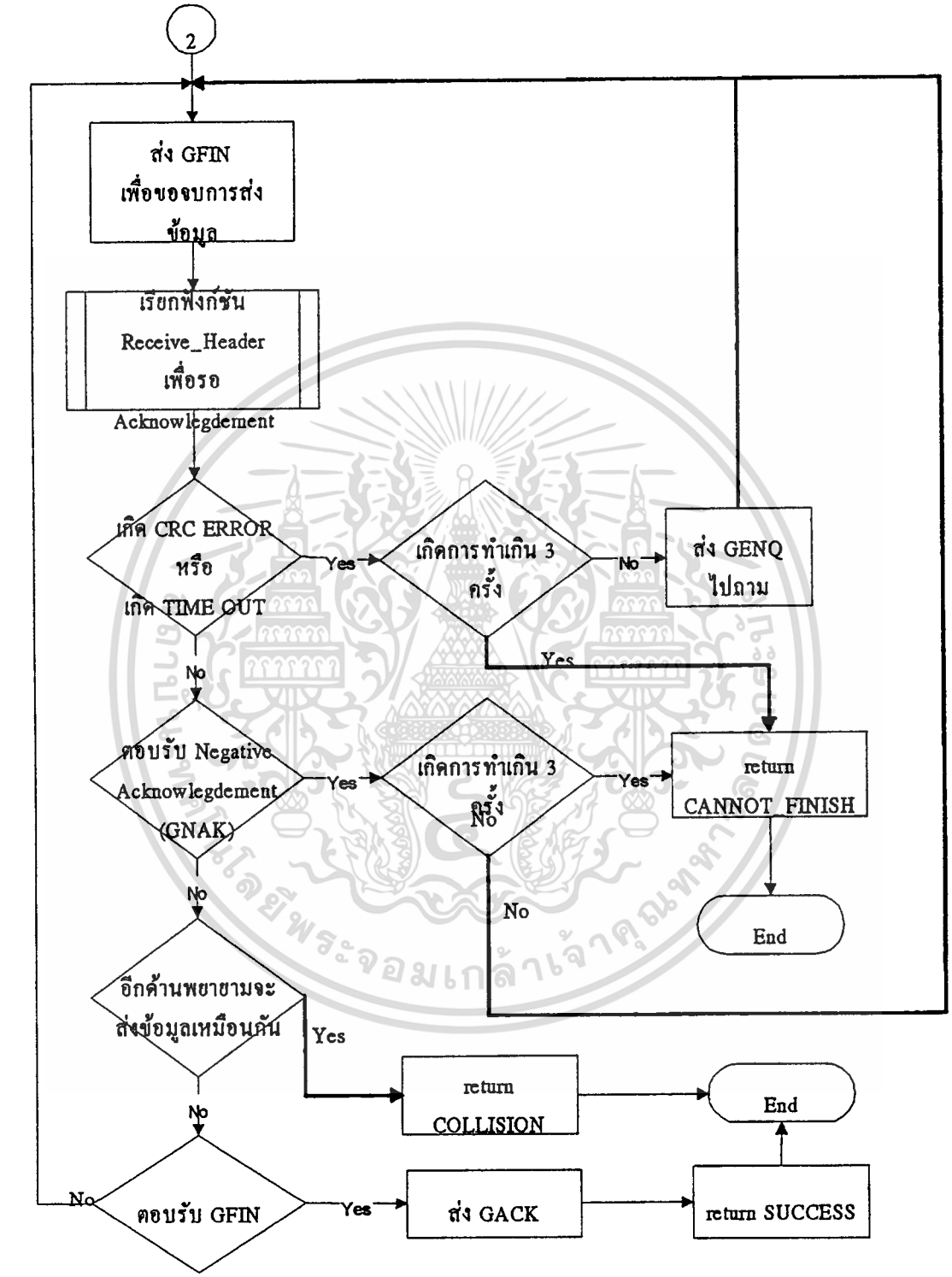
Function Send_Data

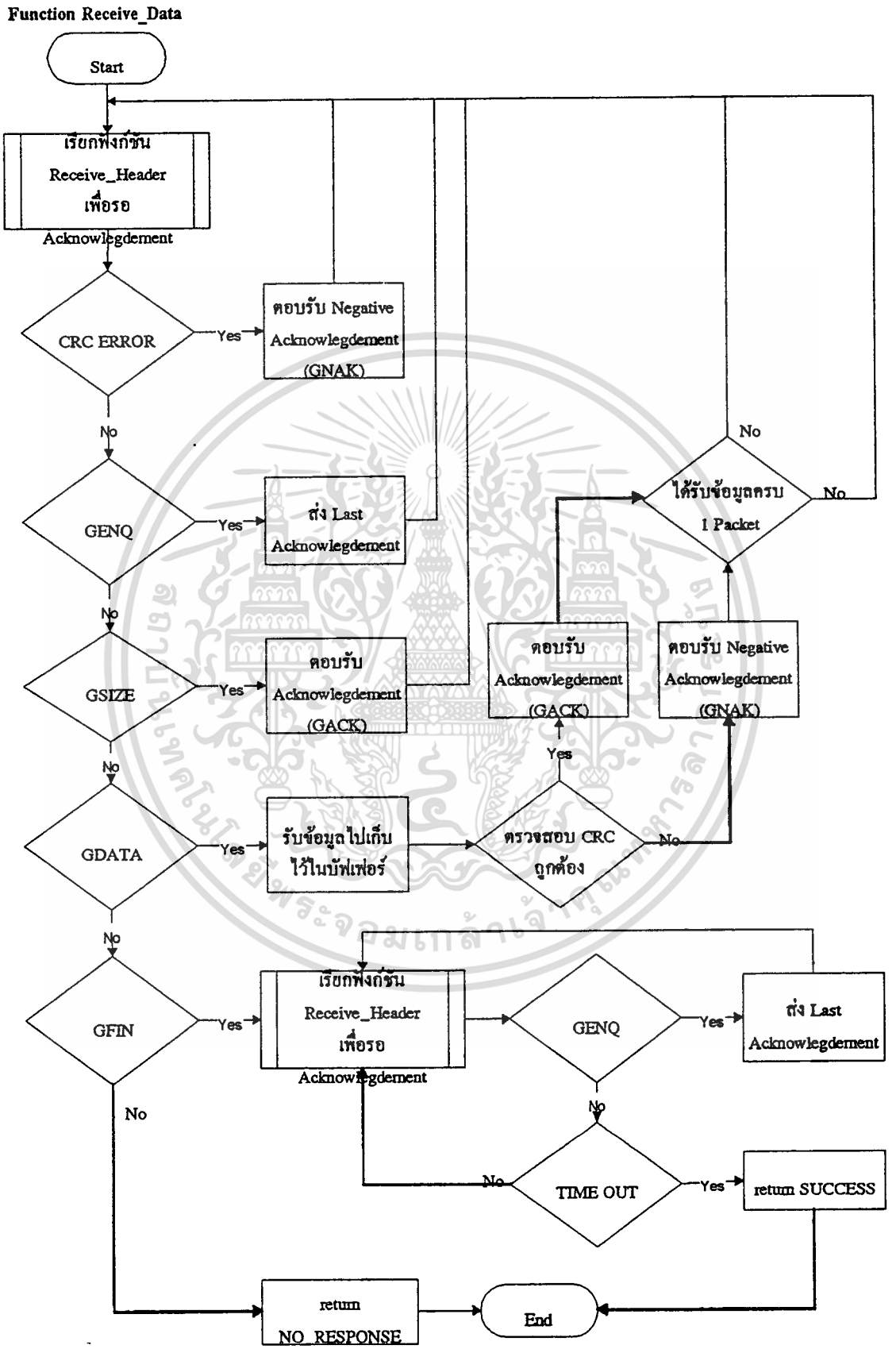


Function Send_Data



Function Send_Data





กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ เกิดขึ้นและสำเร็จได้ด้วยดี คณะผู้จัดทำขอขอบคุณบุคคลที่เกี่ยวข้องดังต่อไปนี้

1. อาจารย์ วัชระ ฉัตรวิริยะ อาจารย์ที่ปรึกษาที่คอยให้คำแนะนำ ตอบขอสักถาม เสนอแนะ และกระตุ้น ทำให้โครงงานสำเร็จได้
2. บริษัท ไอบีเอ็ม ประเทศไทย จำกัด ที่ให้ความอนุเคราะห์โปรแกรม OS/2 เพื่อใช้ในการทำปริญญานิพนธ์นี้
3. ขอขอบคุณเพื่อนๆ และ น้องๆ ที่เป็นกำลังใจให้

หนังสืออ้างอิง

1. จิรพัฒน์ จันทร์เจิดศักดิ์ และ วีระ นพนิราพาธ, "เขียนโปรแกรมบนไมโครซอฟต์วินโดวส์" บ. ซีเอ็ดยูเคชั่น จำกัด, 608 หน้า, 2536
2. มโน มงคลธนานนท์ และ มงคล มงคลธนานนท์, "คัมภีร์โปรแกรมเมอร์ ฉบับปีเตอร์ นอร์ตัน", บ. อินฟอร์เม ติก บิซิเนส พับลิเคชัน จำกัด, 518 หน้า, 2537
3. มีชัย เจริญด้วยศีล, "OS/2 2.1 ระบบปฏิบัติการแบบ 32 บิต สำหรับวันนี้", วารสารไอทีซอฟต์แวร์, ฉบับที่ 21, 2536, หน้า 109-125.
4. วีรศักดิ์ กาญจนรจิต, "บันทึกที่ไม่ลับของวินโดวส์", วารสารคอมพิวเตอร์รีวิว, ฉบับที่ 94, 2536, หน้า 159-282
5. วีรศักดิ์ กาญจนรจิต, "หลากหลาย Dialog Box", วารสารคอมพิวเตอร์รีวิว, ฉบับที่ 112, 2536, หน้า 263-282
6. วีรศักดิ์ กาญจนรจิต, "Drawing in Windows ตอนที่ 2", วารสารคอมพิวเตอร์รีวิว, ฉบับที่ 106, 2536, หน้า 208-219
7. วีรศักดิ์ กาญจนรจิต, "Child windows และ Control", วารสารคอมพิวเตอร์รีวิว, ฉบับที่ 111, 2536, หน้า 223-250
8. ศิริวัฒน์ ศิวะบวร, "ปฐมบทการเขียนโปรแกรมบน Microsoft Windows", วารสารคอมพิวเตอร์รีวิว, ฉบับที่ 94, 2536, หน้า 147-158
9. สุพจน์ ปุณณชัยยะ, "ชุดคู่มือการใช้คอมพิวเตอร์ MODEM ", บ. อินฟอร์เมติก บิซิเนส พับลิเคชัน จำกัด, 149 หน้า, 2534
10. Charles Petzold, "OS/2 Presentation Manager Programming", Ziff-Davis Press, 934 pages, 1994.
11. Derrel R.Blain, Kurt R.Delimon and Jeff English, "Real-World Programming for OS/2 2.1", SAMS PUBLISHING, 868 pages, 1993.
12. IBM Corp. "OS/2 2.0 Technical Library, Programming Guide Volume I", IBM Corp., 1992.
13. IBM Corp. "OS/2 2.0 Technical Library, Programming Guide Volume II", IBM Corp., 1992.
14. IBM Corp. "OS/2 2.0 Technical Library, Programming Guide Volume III", IBM Corp., 1992.
15. IBM Corp. "OS/2 2.0 Technical Library, Presentation Manager Programming Reference Volume I", IBM Corp., 1992.
16. IBM Corp. "OS/2 2.0 Technical Library, Presentation Manager Programming Reference Volume II", IBM Corp., 1992.

17. IBM Corp. "OS/2 2.0 Technical Library, Presentation Manager Programming Reference Volume III", IBM Corp., 1992.
18. Joe Campbell, "C Programmer 's Guide To Serial Communications", HOWARD W.SAMS & COMPANY, 655 pages, 1989
19. Timothy S. Monk, "Window Programming's Guide to Serial Communication", SAMS Publishing, 427 pages, 1992
20. William H. Murray, III and Chris H. Pappas, "Window 3.1 Programming", Osborne/McGraw-Hill, 805 pages, 1992