เวิร์ดโปรเซสเซอร์ บน วินโดวส์

WORDPROCESSOR for Windows

โดย

นายโกศล ธรรมบัณฑิตย์

นายนรินทร์ เอราวัติ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขา วิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2536

ปริญญานิพนธ์ ปีการศึกษา 2536

ภาควิชา วิศวกรรมคอมพิวเตอร์
คณะ วิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เวิร์ดโปรเซสเซอร์ บน วินโดวส์

ผู้จัดทำ

นายโกศล ธรรมบัณฑิตย์
นายนรินทร์ เอราวัติ

..... อาจารย์ที่ปรึกษา
(.....)

เวิร์ดโปรเซสเซอร์ บน วินโดวส์

นายโกศล ธรรมบัณฑิตย์
นายนรินทร์ เอราวัติ
ดร.บุญธีร์ เครือตราชู อาจารย์ที่ปรึกษา
ปีการศึกษา 2536

บทคัดย่อ

โปรเจคนี้เป็นการสร้างเวิร์ดโปรเซสเซอร์(ไทย/อังกฤษ)ที่รันบนระบบ Multitasking ของ Microsoft Windows 3.1 ซึ่งเป็นแบบ Multiple Document Interface ที่สามารถเปิดเอกสารได้มากที่สุด 8 เอกสาร มีคุณสมบัติในการตัดคำ และเช็คคำผิดทั้งภาษาไทยและภาษาอังกฤษ นอกจากนี้ยังใช้คุณสมบัติพิเศษที่วินโดวส์มีไว้ให้เช่น การที่วินโดวส์รันบนกราฟฟิคโหมด ดังนั้นเอกสารจึงสามารถรวมภาพกราฟฟิคเข้าไปได้ด้วย รวมทั้งการพิมพ์เอกสารซึ่งสามารถพิมพ์ได้ทั้งเครื่อง Dot Metrix และ Laser

WORDPROCESSOR for Windows

KOSON TUMBUNDIT

NARIN ARAWAT

Dr.BOONTEE KURTACHU Advisor

1993

Abstract

This Project is Word Processor(Thai/Eng) that run on Multitasking with Microsoft Windows 3.1. The Multiple Document on my project can be opened maximum 8 document and can check spelling word. Because Microsoft Windows run on graphic mode so that we can load graphic file for insert it to document. And document can print with Dot Metrix printer and Laser printer

สารบัญ

แนวความคิดหลัก

1. โปรแกรมต้องการพื้นที่หน่วยความจำหลัก(main memory)จำกัด เพื่อใช้ในการแก้ไขข้อความในเอกสารที่มีขนาดได้ไม่จำกัด จากแนวความคิดนี้ โปรแกรมจะต้องสามารถเปิดเอกสารที่มีขนาดเท่าใดก็ได้(ตามข้อจำกัดของโปรแกรม) ที่ถูกสร้างโดยโปรแกรมนี้บนเครื่องที่มี hardware ตามที่กำหนดไว้เป็นอย่างน้อย ตัวอย่างเช่น โปรแกรมจะต้องสามารถเปิดไฟล์เอกสารขนาด 10 MB ได้ บนเครื่องที่มีหน่วยความจำอย่างน้อย 2 MB เป็นต้น จากรูป จะเห็นว่ามีการส่งข้อมูลไป-มาระหว่าง main memory กับ Disk โดยอาศัยหลักที่ว่าในขณะใดขณะหนึ่งผู้ใช้ที่เปิดเอกสารขึ้นมาเพื่อแก้ไข, ป้อนข้อความ, ตรวจทาน จะไม่ได้ทำบน data ของเอกสารทั้งหมด แต่จะทำแค่ส่วนใดส่วนหนึ่งของเอกสารเท่านั้น แนวความคิดคือ ต้องการรักษาข้อมูลบางส่วนที่กำลังใช้ในปัจจุบันให้อยู่ใน main memory เพียงพอที่การตอบสนองต่อผู้ใช้ไม่ช้าจนยอมรับไม่ได้ ซึ่งพื้นที่หน่วยความจำส่วนนี้จะมีขนาดแน่นอน ดังนั้นไม่ว่าจะเปิดเอกสารที่มีขนาดเท่าใดก็ตาม ก็สามารถจะทำได้ถ้าสามารถจองหนวยความจำส่วนนี้ได้

2. เพื่อให้การตอบสนองต่ออินพุทจากผู้ใช้เป็นไปอย่างรวดเร็ว โปรแกรมจะต้องหลีกเลี่ยงการเคลื่อนย้ายข้อมูลเป็นจำนวนมาก ๆ ทุกครั้งที่มีการป้อนอินพุทจากผู้ใช้ ดังนั้นโปรแกรมจึงมีการกันพื้นที่ใน main momory ส่วนหนึ่งเพื่อใช้เป็น buffer รับข้อมูลจากผู้ใช้ก่อนที่จะนำไปเก็บในโครงสร้างข้อมูลหลักต่อไป

สำหรับ buffer ส่วนนี้จะเป็น buffer ของข้อมูลหนึ่งบรรทัดในเอกสาร เพื่อรับข้อมูลครบหนึ่งบรรทัดแล้ว โปรแกรมก็จะนำไปเก็บในโครงสร้างข้อมูล เพื่อที่จะรอรับข้อมูลของบรรทัดใหม่ต่อไป

3. เพื่อหลีกเลี่ยงการเคลื่อนย้ายข้อมูลทีละมาก ๆ เมื่อมีการแทรกข้อมูลตรงกลางโครงสร้างข้อมูลหลักจะต้องรองรับในจุดนี้ได้ ซึ่งจะกล่าวในรายละเอียดต่อไป

4. สำหรับเอกสารที่ถูกสร้างโดยโปรแกรม จะต้องสามารถที่จะผสมข้อมูลหลายประเภทเข้าด้วยกันได้เพื่อให้โครงสร้างข้อมูลหลักสามารถที่จะเก็บข้อมูลหลาย ๆ ชนิด (เช่น ตัวอักษร, รูปภาพ, เสียง ฯลฯ) เข้าด้วยกันได้ โปรแกรมจึงมีส่วนประกอบส่วนหนึ่งเรียกว่า "Display Manager" เพื่อใช้แยกแยะข้อมูลแต่ละชนิด นำไปแสดงผลได้อย่างถูกต้อง โดยไม่ขึ้นกับอุปกรณ์ ที่จะแสดงผลลัพธ์นั้นด้วย

5. เพื่อให้ง่ายต่อการ implement ข้อมูลประเภทต่าง ๆ ที่เก็บไว้รวมกันในโครงสร้างข้อมูลหลักจะถูกแบ่งออกเป็น paragraph คือ ข้อมูลที่อยู่ใน paragraph เดียวกันจะต้องเป็นข้อมูลชนิดเดียวกัน เช่น paragraph ของตัวอักษร เป็นต้น ในเอกสารสามารถประกอบด้วย paragraph ของข้อมูลชนิดเดียวกันได้หลาย paragraph แต่ละ paragraph มี style ไม่เหมือนกัน เช่น paragraph ของตัวอักษรมี rightindent, justify, linespacing ต่างกัน เป็นต้น

ปัญหาในการบริหารหน่วยความจำ

เนื่องจากระบบ Window มีการบริหารหน่วยความจำโดยการใช้ตาราง descripter (ตารางที่ใช้เก็บข้อมูลเกี่ยวกับพื้นที่หน่วยความจำ) เพียงตารางเดียว โดยใช้ร่วมกับแอพพลิเคชันทุกตัวที่กำลังรันอยู่ในระบบ ดังนั้น slot ของตาราง descriptor ที่จะมีจำกัด (เท่ากับ 4096 slots ใน standard modeและ 8192 slots ใน 386 enhance mode) ซึ่งแต่ละ slot สำหรับพื้นที่หน่วยความจำส่วนเล็ก ๆ เพื่อใช้เก็บข้อมูล จะทำให้เปลืองทรัพยากรของระบบ คือ slot ในตาราง descriptor (เพราะว่าต้องใช้slot เหล่านี้ร่วมกัน) นอกจากนี้ overhead ของแต่ละเซกเมนต์สูง ถ้าเซกเมนต์นั้นมีขนาดเล็ก

การแก้ปัญหาของโปรแกรมนี้จะ allocate พื้นที่หน่วยความจำขึ้นมา 1 เซกเมนต์ (ซึ่งใช้เพียง 1 slot ในตาราง descriptor) ซึ่งมีขนาดแน่นอนภายในเซกเมนต์ โปรแกรมจะบริหารหน่วยความจำเองเพื่อแบ่งเป็น block ของข้อมูลขนาดเท่า ๆ กัน เมื่อมีการขอพื้นที่หน่วยความจำเพิ่ม โปรแกรมจะนำพื้นที่ใน segment 1 blockไปให้กับ routine ที่ขอใช้พื้นที่หน่วยความจำ สำหรับรายละเอียดของการแบ่งข้อมูลในเซกเมนต์จะกล่าวในหัวข้อต่อไป

โครงสร้างสำหรับข้อมูลหลัก

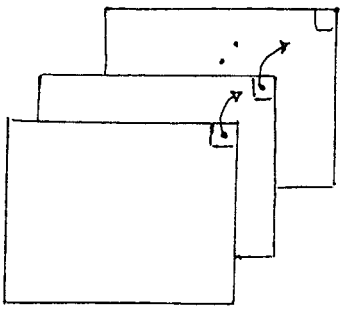
สำหรับโครงสร้างข้อมูลหลักสำหรับเก็บข้อมูลนั้น จะกล่าวถึงเฉพาะส่วนที่ใช้เก็บข้อมูลประเภทตัวอักษรเท่านั้น เพราะข้อมูลประเภทตัวอักษรเป็นข้อมูลที่สามารถแทรกข้อมูลใหม่ลงตรงกลางของข้อมูลเดิมได้ ซึ่งทำให้โครงสร้างข้อมูลหลักของข้อมูลประเภทตัวอักษร ซับซ้อนกว่าโครงสร้างข้อมูลหลักของข้อมูลชนิดอื่น

โครงสร้างข้อมูลหลักของข้อมูลตัวอักษรจะต้องสามารถที่จะแทรกคำ, ลบคำ, นำออกกิแสดงผลได้อย่างรวดเร็ว ข้อกำหนดของการทำงานของโครงสร้างหลัก มีดังนี้

1. ในการแทรกคำ, ลบคำ โปรแกรมจะต้องไม่เคลื่อนย้ายข้อมูลทีละมาก ๆ ต่อการแทรกคำลบคำหนึ่งครั้ง

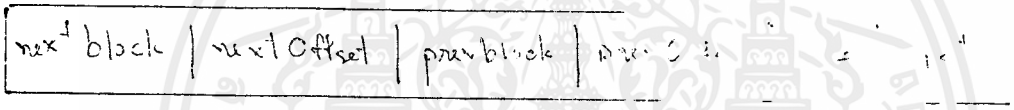
2. ในการนำข้อมูลออกแสดงผล, คำนวณเพื่อจัดบรรทัด ข้อมูลที่เก็บในพื้นที่หน่วยความจำจะต้องไม่กระจัดกระจาย และเรียงตามลำดับที่อยู่ในเอกสารจริง ๆ

จากข้อกำหนดข้างต้น จึงได้โครงสร้างข้อมูลที่เป็น list ของ block ของข้อมูลกลุ่มหนึ่งแสดงดังรูป



ในหนึ่ง block ของข้อมูลคงที่และเพื่อให้ข้อมูลที่จะนำออกแสดงผลได้อย่างรวดเร็วข้อมูลของเอกสารหนึ่งบรรทัดจะถูกเก็บอยู่ใน block ของข้อมูลเดียวกัน ดังนั้นพื้นที่ใน block ของข้อมูลจึงอาจไม่ได้ใช้หมดซึ่งที่เหลือเป็น fragment ที่เกิดขึ้น เพื่อแลกกับประสิทธิภาพในการแสดงผลของข้อมูล

สำหรับโครงสร้างข้อมูลหลักของแต่ละ block ข้อมูลบระกอบด้วย field ดังต่อไปนี้



ซึ่งแต่ละ field มีความหมายต่อไปนี้

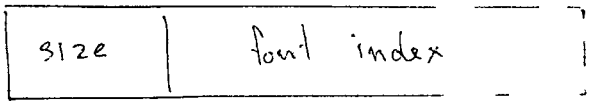
- nextBlock, nextOffset :- เป็น pointer ของข้อมูล block ถัดไปในหน่วยความจำและใน disk ถ้า nextBlock =-1 แสดงว่า block ของข้อมูลถัดไป อยู่ใน disk ใน swap file ที่มี offset = next offset ถ้า nextBlock = 0 แสดงว่าไม่มี block ถัดไป

- prevBlock, prevOffset :- เป็น pointer ของข้อมูล block ก่อนหน้านั้นในหน่วยความจำและใน disk ถ้า prevBlock = -1 แสดงว่า block ของข้อมูลก่อนหน้านั้นอยู่ใน disk ใน swap file ที่มี offset = prevOffset ถ้า prevBlock = 0 แสดงว่า block นี้เป็น block ที่อยู่ต้น list ของข้อมูล

- size :- เก็บขนาดของข้อมูลที่มีอยู่ใน Block นี้

- list :- เป็น pointer ของ block ถัดไปใน LRU-list

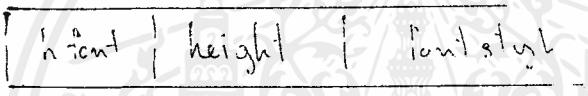
- script :- เนื่องจากโปรแกรมสามารถที่จะแสดงตัวอักษรได้หลายขนาด หลายฟอนต์จึงต้องมีโครงสร้างข้อมูลเพิ่มเติม เพื่อใช้กำกับข้อมูลตัวอักษรในแต่ละช่วงของ Block ว่าฟอนต์อะไร, ขนาดเท่าไร, มี style เป็นอย่างไร ซึ่งโครงสร้างข้อมูลนี้แสดงดังรูป



แต่ละ field มีความหมายต่อไปนี้

- size :- คือขนาดของกลุ่มตัวอักษรที่ใช้ฟอนต์ชนิดเดียวกัน

- fontindex :- เป็นหมายเลขของ font ในตาราง fonttable ซึ่งเป็นตารางที่ใช้เก็บลักษณะเฉพาะตัวของแต่ละฟอนต์ ซึ่งมีโครงสร้างข้อมูลแสดง ดังรูป



แต่ละ field มีความหมายต่อไปนี้

- hfont :- เป็น handle ของฟอนต์ ที่ Windows ใช้กำหนดฟอนต์แต่ละชนิด

- height.- ขนาดความสูงของฟอนต์

- fontstyle :- เป็นหมายเลขกำหนดลักษณะของ font โดยที่ฟอนต์ที่มีลักษณะเหมือนกันแต่ขนาดต่างกันจะใช้ fontstyle เดียวกันมีโครงสร้างข้อมูลของfontstyle แสดงดังรูป

แต่ละ field มีความหมายต่อไปนี้

- fontFamily :- หมายเลขกำกับว่าฟอนต์นี้อยู่ใน family อะไร
- typeface :- ชื่อของฟอนต์
- fontCharacter :- ลักษณะของฟอนต์ เช่น ตัวเอียง, ตัวหนา, ตัวปกติ เป็นต้น

- Data เป็นฟิลด์ที่ใช้เก็บข้อมูลตัวอักษรเพียงอย่างเดียว ไม่มีการแทรกอักษรควบคุมพิเศษเข้าไประหว่างข้อมูลเดิม

จากโครงสร้างข้อมูลของ TextBlock ถ้าให้ขนาดของข้อมูลที่มากที่สุดที่เก็บในแต่ละ block = 512ไบต์แล้ว โครงสร้างข้อมูลนี้จะต้องเก็บข้อมูลเพิ่มพิเศษเพื่อใช้ในโปรแกรมเพิ่มอีก โดยส่วนที่เพิ่มนี้คำนวณได้ดังนี้

prevBlock, nextBlock = 4 ไบท์

prevOffset, nextOffset = 8 ไบท์

size = 2 ไบท์ (เพื่อใช้ในการอ้างข้อมูลขนาด 512 ไบท์)

list = 2 ไบท์

script :- ถ้าให้หนึ่ง Block สามารถเก็บข้อมูลที่ใช้ฟอนต์แตกต่างกันได้มากที่สุด 16 แบบ ซึ่งแต่ละของ script มีขนาด = 2 + 2 = 4 ไบท์

พื้นที่ที่ใช้เก็บ script Table = 16 x 4 = 64 ไบท์

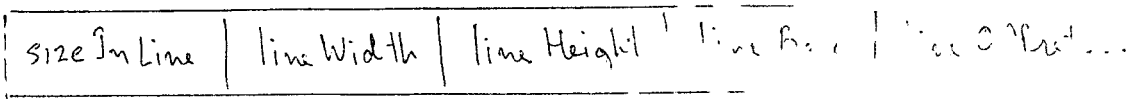
พื้นที่ Overhead = 64 + 2 + 2 + 8 + 4 = 80 ไบท์

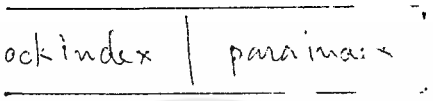
เปอร์เซ็นต์ของ Overhead = 80/512 x 100% = 16 %

นอกจากโครงสร้างข้อมูล TextBlock และส่วนที่เกี่ยวข้องแล้ว โปรแกรมต้องเก็บข้อมูลเพิ่มอีกเพื่อใช้ในการจัดบรรทัดในเอกสาร เพื่อให้การแสดงผลสามารถทำได้อย่างรวดเร็ว โปรแกรมต้องเก็บข้อมูลเกี่ยวกับการจัดบรรทัดในเอกสารไว้เพื่อที่โปรแกรมจะไม่ต้องทำการคำนวณจัดบรรทัดใหม่ทุกครั้งที่มีการแสดงผลแต่เนื่องจากพื้นที่หน่วยความจำมีจำกัด การที่จะเก็บข้อมูลของการจัดบรรทัดทุกบรรทัดในเอกสารจึงไม่สามารถทำได้ ในโปรแกรมจะเก็บข้อมูลเหล่านี้เพียงบางส่วนในขนาดที่ทำให้ประสิทธิภาพในการแสดงข้อมูลในเอกสารสามารถยอมรับได้

ส่วน Display Manager จะเก็บข้อมูลในการจัดบรรทัดในเอกสารไว้ในโครงสร้างข้อมูลแบบ Display Table เมื่อมีการแสดงผล Display Manager จะนำข้อมูลส่วนนี้มาใช้ประกอบในการดึงข้อมูลในโครงสร้างข้อมูล Text Block เพื่อนำออกแสดงผลใน window

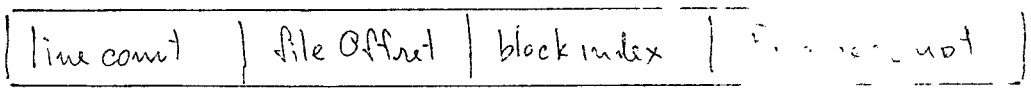
โครงสร้างข้อมูล Display Table แสดงดังรูป





แต่ละฟิลด์มีความหมายต่อไปนี้

- sizeInLine :- จำนวนไบท์ของข้อมูลในบรรทัด
- lineWidth :- ความกว้างของ string ของตัวอักษรในบรรทัด
- lineHeight :- ความสูงของ string ของตัวอักษรในบรรทัด
- lineBase :- ความสูงของตัวอักษรในบรรทัด
- lineOffset :- offset ใน TextBlock ที่ข้อมูลในบรรทัดอยู่
- blockindex :- pointer ของ TextBlock ที่ข้อมูลในบรรทัดอยู่
- paraIndex :- เนื่องจาก ในเอกสารสามารถมีได้หลาย paragraph โดยแต่ละ paragraph มี style ไม่เหมือนกัน สำหรับฟิลด์ paraIndex เป็น index ของ paragraph Table ซึ่งเป็นตารางที่ใช้เรียงลำดับแต่ละ paragraph ในเอกสารและเก็บลักษณะเฉพาะต่าง ๆ ของแต่ละ paragraph ซึ่งโครง สร้างข้อมูลของ paragragh table แสดงดังรูป

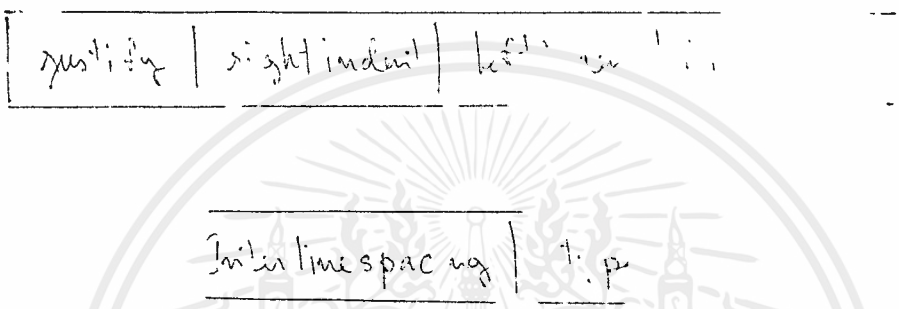


แต่ละฟิลด์มีความหมายดังต่อไปนี้

- linecount :- จำนวนบรรทัดใน paragraph

- fileOffset, blockindex :- pointer ของข้อมูล Block แรกของ paragraph ในดิสค์และในหน่วยความจำ ถ้า blockindex = -1 แสดงว่าข้อมูลใน block แรกของ paragraph อยู่ในดิสค์ที่ offset = fileOffset ของ swap file

- paradescript :- เป็น index ของตาราง paradescript table ซึ่งเป็นตารางที่ใช้เก็บ style ต่าง ๆ ของ paragraph ซึ่งโครงสร้างข้อมูลแสดงดังรูป



ซึ่งแต่ละฟิลด์มีความหมายต่อไปนี้

- justify :- กำหนด style ในการแสดงข้อความในแต่ละบรรทัดใน paragraph ได้แก่ left justify, right justify, center justify, full justify

- rightindent :- ระยะห่างจากขอบกระดาษทางด้านขวามือ

- leftindent :- ระยะห่างจากขอบกระดาษทางด้านซ้ายมือ

- firstleftindent :- ระยะห่างจากขอบกระดาษทางด้านซ้ายมือของบรรทัด แรกใน paragraph

- Interlinespacing :- ระยะห่างระหว่างบรรทัด

- Type :- กำหนดชนิดของข้อมูลใน paragraph ได้แก่ ตัวอักษร, รูปภาพ

หลักการในการ swap ข้อมูล

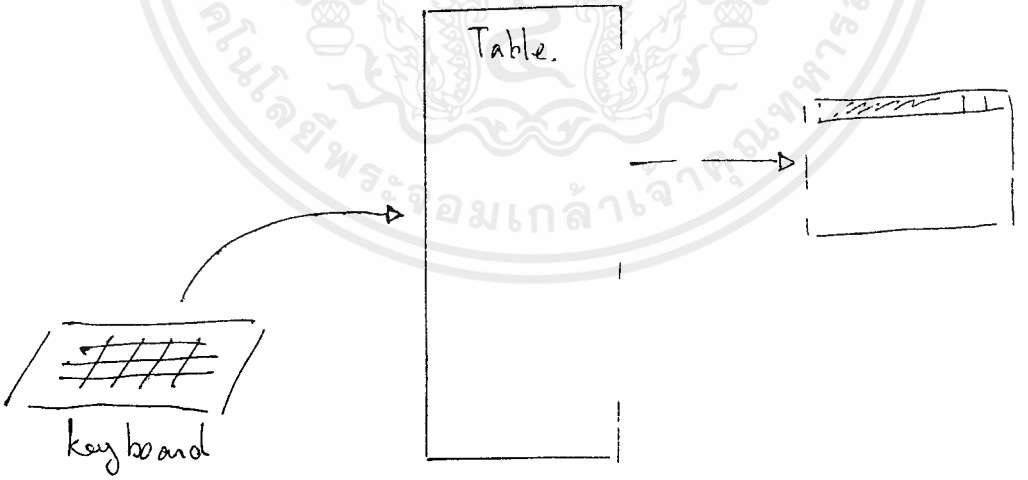
เพื่อให้โปรแกรมสามารถเปิดไฟล์เอกสารที่มีขนาดใหญ่กว่าพื้นที่หน่วยความจำในระบบโปรแกรมจะต้องมีการ swap ข้อมูลบางส่วนที่ไม่ได้ใช้ในขณะหนึ่งลงไปไว้ใน Disk ซึ่งหลักที่ใช้ในการ swap ข้อมูลส่วนใดลงในดิสค์ มีหลักดังนี้

1. จากโครงสร้างข้อมูลหลักที่ข้อมูลแบ่งเก็บเป็น black โดยแต่ละ paragraph มี list ของ block ข้อมูลเป็นของตัวเอง การ swap จะใช้ขนาดที่แน่นอนคือ 1 block จาก main memory ไปแทนที่ข้อมูลชนิดเดียวกัน 1 block ในดิสก์

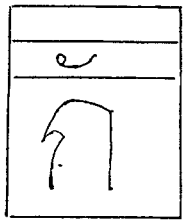
2. ข้อมูลที่น่าจะถูก swap ลงไป disk คือข้อมูลที่ไม่ได้อยู่ใน Display Table ข้อมูลที่อยู่ในหน่วยความจำซึ่งไม่ได้อยู่ใน display Table จะถูกบรรจุอยู่ใน LRU list (Least Recently Use List) เมื่อข้อมูลถูกเลื่อนออกไปจาก display Table ข้อมูลส่วนนั้นจะถูกนำไปต่อท้ายของ LRU list เมื่อต้องการพื้นที่ในหน่วยความจำเพิ่มโปรแกรมจะ swap ข้อมูลส่วนที่อยู่ต้นของ list ไปเก็บไว้ในดิสก์ ข้อมูลส่วนถัดมาก็จะถูกเลื่อนขึ้นมาอยู่ต้น list แทน ถ้าข้อมูลที่อยู่ใน list ไม่ถูกเลื่อนเข้าไปอยู่ในdisplay Table เลย ข้อมูลส่วนนั้นก็จะถูก swap ลงไปเก็บไว้ในดิสก์ในไม่ช้า

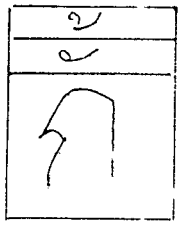
การแสดงผลภาษาไทย

โปรแกรมที่ได้ออกแบบมาเพื่อให้สามารถแสดงฟอนต์ได้หลายชนิด ดังนั้นถ้ามีการติดตั้งฟอนต์ภาษาไทยแบบ True Type ฟอนต์ โปรแกรมสามารถที่จะแสดงภาษาไทยได้ เพราะลักษณะของ True Type ฟอนต์เอื้ออำนวยต่อการแสดงผลภาษาไทย โดยที่โปรแกรมไม่ต้องจัดระดับของสระ และวรรณยุกต์เอง หรือสร้าง routine ที่ใช้คำนวณความกว้างของ string ภาษาไทยใหม่สามารถใช้ API เดียวกับภาษาอังกฤษได้ตรงแต่โปรแกรมจะต้องปรับเปลี่ยนการเลื่อน caret และการ delete เล็กน้อย และจะต้อง map คีย์บอร์ดและโค้ดของตัวอักษรที่จะแสดงผล เพิ่มเติม แสดงดังรูป



สำหรับสาเหตุที่ต้องมีตาราง map internal code - display code เพื่อความสวยงามของการแสดงผล ตัวอย่างเช่น





จะเห็นได้ว่าทั้งสองรูปนี้ อยู่ต่างระดับกัน การแสดงภาษาไทยใน window เหมือนกับนำแผ่นใส หลายแผ่นมาซ้อนกัน จากรูปที่ 1 เกิดจากการนำเอาภาพของ ก มาซ้อนกับภาพของ และภาพของ ซึ่ง ของ ภาพที่ 1 และ ภาพที่ 2 ต่างก็มีรหัสที่แตกต่างกัน ดังนั้นจึงต้องมีการ map เพื่อเลือก ที่ถูกต้องก่อนที่จะแสดงผล

FONT

องค์ประกอบของ FONT

ในการศึกษาเรื่องฟอนท์จำเป็นที่จะต้องเข้าใจความหมายของศัพท์ที่ใช้ในการเรียกชื่อองค์ประกอบต่างๆรวมถึงความหมายขององค์ประกอบที่สำคัญต่าง ๆ

TypeFace เป็นกลุ่มของตัวอักษรที่มีลักษณะพิเศษร่วมกัน

Font คือกลุ่มของตัวอักษรที่มี TYPEFACE และขนาดเดียวกัน

WINDOWS จัดแบ่งฟอนท์ออกเป็นแฟมิลี่โดยอาศัย stroke width (ความกว้างของเส้นที่ประกอบเป็นตัวอักษร)และ serif(คือเส้นตามแนวนอนที่ปลายของ stroke TYPEFACE ที่ไม่มี serif ที่ปลาย stroke เรียกว่า SanSerif TypeFace)

ภายในฟอนท์ยังสามารถแยกย่อยออกไปได้อีกโดยอาศัยคุณสมบัติสองข้อคือ ความเข้ม(weight) และ ความเอียง(slant) ความเข้มแบ่งออกเป็น extra light, light, demi, demi bold, book,bold,heavy bold, extra bold และ black สำหรับความเอียงของฟอนท์แบ่งออกเป็น roman (ตัวตรง), oblique (ตัวเอียง) และ italic (ตัวเอียง และค่อนข้างเหมือนตัวเขียนธรรมดา) โดยปกติฟอนท์แฟมิลี่ไม่รวม oblique และ italic ไว้ด้วย (เพราะว่าสามารถที่จะสร้างขึ้นมาได้) ใน GDI แบ่งฟอนท์ออกเป็น 5 แฟมิลี่ แยกเป็นรายละเอียดดังนี้

Font-family Name Description

FF_DECORATIVE เป็นฟอนท์ที่มีลักษณะพิเศษ เช่นตัวอักษรภาษาอังกฤษโบราณ

FF_MODERN เป็นฟอนท์ที่มี stroke width คงที่และเป็นฟอนท์แบบ SanSerif ฟอนท์ที่มีช่วงกว้างคง

ขนาดใหญ่กว่าฟอนท์ขนาดเดียวกันที่แสดงทางพรินเตอร์ ฟอนท์ที่แสดงบนจอแสดงผลมีขนาดใหญ่กว่าเพราะว่าจอแสดงผลมีความละเอียดไม่พอที่จะแสดงฟอนท์ขนาด 10 พอยน์มารถดูรู้เรื่อง และโดยปกติการอ่านตัวอักษรบนจอแสดงผลจะห่างมากกว่าการอ่านอักษรบนกระดาษถึงแม้นว่า *logical inch* แก้ปัญหาการแสดงฟอนท์ให้สามารถอ่านได้รู้เรื่องบนจอแสดงผล แต่ผลที่ตามมาก็คือ ตัวอักษรที่ปรากฏบนจอแสดงผลกับบนกระดาษมีขนาดที่แตกต่างกัน

การใช้ฟอนท์ในแอพพลิเคชัน

Stock Font

Stock Font คือฟอนท์ที่มีอยู่ใน GDI แล้วถูกสร้างเมื่อระบบเริ่มทำงาน ใน GDI มีฟอนท์ให้เลือกใช้อยู่หลายชนิดซึ่งเพียงพอสำหรับงานแสดงผลข้อมูลธรรมดา การเรียกใช้ stock font ทำได้โดยการเรียกฟังก์ชัน GetStockObject ซึ่งจะรีเทอร์นแฮนเดิลของลอจิกคอลฟอนท์ที่เลือกกลับมา เมื่อแอพพลิเคชั่นเลือกฟอนท์ใน device context (โดยใช้ฟังก์ชัน SelectObject) font mapper ใช้ลอจิกคอลฟอนท์ ในการสร้างฟิสิคัลฟอนทที่ใช้งานจริง หลังจากนั้นแอพพลิเคชั่นสามารถใช้ฟอนท์ที่เลือกในการแสดงตัวอักษรได้

ตารางข้างล่างแสดง stock font ที่ GDI มีให้เลือกใช้

Font *Description*

ANSI_FIXED_PITCH ฟอนท์ที่มีช่วงกว้างคงที่ ใช้ชุดอักษรของ WINDOWS โดยปกติคือ Courier Typeface

ANSI_VAR_FONT ฟอนท์ที่ช่วงกว้างไม่คงที่ใช้ชุดอักษรของ WINDOWS โดยปกติคือ MS San Serif Typeface

DEVICE_DEFAULT_FONT ฟอนท์ที่เหมาะสมกับอุปกรณ์แสดงผล ฟอนท์ที่ใช้เปลี่ยนไปตามอุปกรณ์ที่ใช้

OEM_FIXED_PITCH ฟอนท์ที่มีช่วงกว้างคงที่ ใช้ชุดอักษรของ OEM ซึ่งเปลี่ยนไปตามอุปกรณ์ที่ใช้สำหรับเครื่อง IBM PC และ เครื่องคอมแพทิเบิลใช้ชุดอักษรของ IBM

SYSTEM_FONT ฟอนท์ที่มีช่วงกว้างไม่คงที่ ใช้ชุดอักษรของ WINDOWS ซึ่ง WINDOWS ใช้แสดงตัวอักษรต่าง ๆ ของระบบเช่น window title ,เมนู, ข้อความใน dialog box

ถ้าไม่มี stock font ในระบบ ฟังก์ชั่น GetStockObject จะรีเทอร์นแฮนเดิลของ SYSTEM_FONT สำหรับแอพพลิเคชั่นที่ใช้ฟังก์ชั่น GetStockObject เพื่อที่จะได้แฮนเดิลของลอจิกคัลฟอนท์ควรที่จะมี mapping mode ใน MM_TEXT

การสร้าง Logical Font

ลอจิกคัลฟอนท์คือ แอททริบิวท์ต่างๆของฟอนท์เช่นความสูง(height), ความกว้าง(width),ชุดของตัวอักษร(character set) และ typeface แอพพลิเคชั่นสร้างลอจิกคัลเพื่ออธิบายลักษณะพิเศษของฟอนท์ที่เหมาะสมกับงานของแอพพลิเคชั่น font mapper ใช้ลอจิกคัลฟอนท์ในการเลือกฟิสิคัลฟอนท์ที่มีอยู่โดยให้มีลักษณะใกล้เคียงกับลอจิกคัลฟอนท์มากที่สุด

แอพพลิเคชั่นสามารถใช้ฟังก์ชั่น CreateFont หรือCreateFontIndirct ในการสร้างลอจิกคัลฟอนท์แอพพลิเคชั่นส่วนใหญ่นิยมใช้ฟังก์ชั่น CreateFontIndirect เพราะว่ามีพารามิเตอร์ที่ต้องส่งไปให้ฟังก์ชั่นน้อยกว่าแบบแรก ฟังก์ชั่นทั้งสองรีเทอร์นแฮนเดิลของลอจิกคัลฟอนท์ และสามารถที่จะนำไปใช้กับ device context ได้

ตัวอย่างข้างล่างแสดงฟังก์ชั่นที่มีอินพุทคือ แฮนเดิลของ device context, ชื่อของฟอนท์และขนาดของฟอนท์ ฟังก์ชั่นนี้จะสร้างลอจิกคัลฟอนท์ที่มีขนาดและ typeface ตามที่อินพุทเข้ามาแล้วเปลี่ยนฟอนท์ที่ใช้ใน device context เป็นฟอนท์ที่สร้างขึ้นมาใหม่

```
BOOL FAR PASCAL     CreateLogFont(hdc, pszFace, PointSize)
HDC         hdc;
PSTR        pszFace,
int         PointSzie;
{
    HFONT     hfnt, hfntOld;
    PLOGFONT  plf = (PLOGFONT)LocalAlloc(GPTR, sizeof(LOGFONT));

    if (GetMapMode(hdc) != MM_TEXT)
    {
        TextOut(hdc, 100, -200, "Mapping mode must be MM_TEXT", 28);
        return FALSE;
    }

    plf->lfHeight = -MulDiv(PointSize, GetDeviceCaps(hdc, LOGPIXELSY), 72),
    lstrcpy(plf->lfFaceName, pszFace);


    hfnt = CreateFontIndirect(plf);
```

```
    hfntOld = SelectObject(hdc, hfnt);

        /* Use font for text output */

    LocalFree((LOCALHANDLE)plf);
    SelectObject(hdc, hfntOld);
    return TRUE;
}
```

หน่วยความจำที่ใช้สำหรับลอจิกคัลฟอนท์ถูก allocate และเซ็ทค่าเริ่มต้นเป็นศูนย์(โดยใช้ฟังก์ชั่น LocalAlloc และแอททริบิวท์คือ GPTR) ซึ่งหมายความว่าลอจิกคัลฟอนท์ที่ถูกสร้างจากฟังก์ชั่น Create FontIndirct ใช้ค่า default ยกเว้นในฟิลด์ lfFaceName และ lfHeight

ฟังก์ชั่นในตัวอย่างนี้ใช้ WINDOWS ฟังก์ชั่น MulDiv เพื่อที่จะแปลงขนาดของพอยน์ที่กำหนดให้เหมาะสมกับอุปกรณ์แสดงผลฟังก์ชั่น MulDiv จะคูณขนาดพอยน์ที่ต้องการด้วยผลลัพธ์ที่ได้จากการหารระหว่างจำนวนพิกเซลในหนึ่งลอจิกอินทช์กับจำนวนพอยน์ในหนึ่งฟิสิคัลอินทช์ จากนั้นจะกำหนดค่าลบของผลลัพธ์ที่ได้ให้กับฟิลด์ lfHeight (เป็นค่าลบเพื่อที่จะบอกกับระบบว่าค่าในฟิลด์ lfHeight เป็นค่าของความสูงของ gyph สำหรับค่าบวกจะแทนความสูงของเซลซึ่งรวม internal leading, gyph+internal leading) เมื่อแอพพลิเคชั่นเลือกแฮลอจิกคัลฟอนท์ใน device context font mapper จะรีเทอร์นแฮนเดิลของฟิสิคัลฟอนท์ที่ใกล้เคียงกับลอจิกคัลฟอนท์

ฟังก์ชันที่ใช้ในการแสดงข้อความ

แอพพลิเคชั่นสามารถเรียกใช้ฟังก์ชั่นเหล่านี้เพื่อใช้แสดงข้อความได้

DrawText แสดงข้อความที่มีรูปแบบ (format text) ภายในสี่เหลี่ยม รูปแบบต่างๆที่ฟังก์ชั่นนี้สามารถทำได้เช่น ขยาย tab เป็นช่องว่างที่เหมาะสม, จัดแนวของข้อความ(TextAlignment)เช่น Left, Right, Central,แบ่งข้อความออกเป็นบรรทัดที่สามารถแสดงในสี่เหลี่ยมที่กำหนดได้ เป็นต้น

ExtTextOut แสดงสตริงของตัวอักษรในพื้นที่สี่เหลี่ยมที่กำหนด พื้นที่สี่เหลี่ยมนี้เป็นแบบ opaque (คือเขียนทับด้วยสีพื้น) และ เป็นขอบเขตที่สามารถแสดงตัวอักษร (Clipping region)

TabbedTextOut แสดงสตริงของตัวอักษร พร้อมทั้งขยาย tab เป็นช่องว่างตามที่กำหนด

TextOut แสดงสตริงของตัวอักษรที่ตำแหน่งที่กำหนด เป็นฟังก์ชั่นพื้นฐานที่ใช้แสดงข้อความ ทำงานได้เร็วที่สุด

การกำหนดการจัดวางข้อความ (Text Alignment)

แอพพลิเคชั่นสามารถกำหนดการจัดวางข้อความได้โดยใช้ฟังก์ชั่น SetTextAlign การจัดวางข้อความเป็นการกำหนดลักษณะการวางข้อความเมื่อเทียบกับตำแหน่งที่กำหนด ข้อความสามารถกำหนดให้อยู่ในแนวเดียวในทางขวา, ซ้ายหรือ ตรงกลางของตำแหน่งที่กำหนด สามารถกำหนดการวางข้อความให้อยู่บนหรือล่างของตำแหน่งที่กำหนด นอกจากนี้ฟังก์ชั่น SetTextAlign สามารถที่จะจำตำแหน่งสุดท้ายที่แสดงผลข้อความของฟังก์ชั่นแสดงข้อความ

ตัวอย่างข้างล่างแสดงการใช้ฟังก์ชั่น SetTextAlign ในการจำตำแหน่งสุดท้ายของการแสดงข้อความเมื่อเรียกฟังก์ชั่น TextOut ตัวอย่างนี้ cArial คือจำนวนเต็มที่แสดงจำนวนของฟอนท์แบบ Arial

```
UINT      uAlignPrev;
char      szCount[8];

uAlignPrev = SetTextAlign(hdc, TA_UPDATECP);
MoveTo(hdc, 10, 50);
TextOut(hdc, 0, 0, "Number of Arial fonts: ", 23);
itoa(cArial, szCount, 10);
TextOut(hdc, 0, 0, (LPSTR)szCount, strlen(szCount));
SetTextAlign(hdc, uAlignPrev);
```

ฟังก์ชั่น SetTextJustification เพิ่มช่องว่างพิเศษที่ break character (ตัวอักษรที่ใช้จบคำโดยปกติคือช่องว่าง) ในบรรทัดของข้อความ แอพพลิเคชั่นใช้ฟังก์ชั่น GetTextExtent หาความยาวของข้อความที่แสดงลบออกจากพื้นที่ที่ข้อความอยู่ และใช้ฟังก์ชั่น SetTextJustification เพื่อกระจายช่องว่างที่เหลือให้กับ break character ในบรรทัดนั้นทำให้ข้อความสามารถแสดงผลได้เต็มบรรทัด ฟังก์ชั่นSetTextCharacterExtra ใช้เพิ่มช่องว่างพิเศษให้กับทุกตัวอักษรในฟอนท์ที่เลือกใน devicecontext

การใช้หน่วยความจำของโปรแกรม

โปรแกรมที่รันบนสภาวะของ Windowsสามารถที่จะใช้หน่วยความจำได้มากถึง 16 KBytes ใน standard mode และสี่เท่าของหน่วยความจำที่มีอยู่จริงใน 386 Enhance Mode เช่นในระบบมีหน่วยความจำอยู่ 16 KBytes โปรแกรมที่รันใน 386 Enhance Mode มีหน่วยความจำให้ใช้ได้ถึง 64 KBytes ถ้ามีพื้นที่บนฮาร์ดดิสค์พอสำหรับ swap file ทำให้โปรแกรมพ้นขีดจำกัด 640 KBytes ของดอสที่มีมาในอดีต

ศัพท์เทคนิคที่สำคัญ

ในการจัดการเกี่ยวกับข้อมูลในหน่วยความจำมีความจำเป็นที่จะต้องเข้าใจศัพท์เทคนิคที่ใช้ ที่ต้องใช้ทับศัพท์ภาษาอังกฤษเพราะว่าได้คำที่กะทัดรัด มีความหมายตรงตัว และใช้กันแพร่หลาย ถ้าแปลเป็นภาษาไทยแล้วจะได้ความหมายไม่ครบถ้วนบางทีอาจจะต้องแปลไทยเป็นไทยอีก

"ALLOCATE" คือการจองพื้นที่ในหน่วยความจำเพื่อใช้เก็บข้อมูลคอมไพล์เลอร์ allocate พื้นที่หน่วยความจำเมื่อมีการประกาศชื่อตัวแปรในโปรแกรมโปรแกรม allocate พื้นที่หน่วยความจำเพิ่มในขณะรันเพื่อเก็บข้อมูลที่ได้จากการประมวลผล (เรียกการallocate พื้นที่หน่วยความจำแบบนี้ว่า "DYNAMIC ALLOCATION") Windows allocateพื้นที่หน่วยความจำเมื่อมีการสร้างเมนู,รูปภาพ หรือ GDI Object (คือแอททริบิวท์ที่ GDIใช้ในการแสดงผลเช่น Pen สำหรับลากเส้น, Brush สำหรับระบายสีพื้นหลัง) เช่นเมื่อโปรแกรมสร้าง pen Windows จะ allocate พื้นที่ใน Local Heap ของ GDI ให้สำหรับ Objectใหม่ที่สร้างขึ้นมา

"VISIBILITY" อธิบายว่าใครสามารถเข้ามาใช้ข้อมูลได้ ตัวแปรแบบ automatic สามารถใช้ได้เฉพาะในฟังก์ชั่นที่ประกาศ เซกเมนต์ที่ได้จากการ allocate จาก Global Heap (พื้นที่หน่วยความจำที่ Windows บริหาร)สามารถใช้ได้ทั้งระบบ เหมาะสำหรับในการแลกเปลี่ยนข้อมูลระหว่างโปรแกรม เช่น คลิปบอร์ด เป็นต้น

"LIFE TIME" คือช่วงเวลาที่ข้อมูลจะถูก allocate อยู่ในหน่วยความจำ โดยปกติเมื่อจบโปรแกรมพื้นที่หน่วยความจำที่ถูก allocate สำหรับโปรแกรมนั้นจะถูกยกเลิกไปด้วย แต่ GDI Object ที่ถูกสร้างในโปรแกรมจะไม่ถูกยกเลิกเมื่อโปรแกรมจบการทำงาน ดังนั้นจึงเป็นหน้าที่ของโปรแกรมที่จะยกเลิก GDI Object เหล่านี้มิฉะนั้นพื้นที่หน่วยความจำที่ครอบครองโดย Object เหล่านี้จะเสียไปโปรแกรมอื่นจะเหลือพื้นที่หน่วยความจำที่ใช้งานน้อยลง

"OVERHEAD" พื้นที่หน่วยความจำที่ต้องถูก allocate เพิ่มจากพื้นที่หน่วยความจำที่ขอไปโดยที่โปรแกรมที่ allocate ไม่สามารถใช้พื้นที่ส่วนนี้เก็บข้อมูลได้พื้นที่ส่วน Overhead จะใช้เก็บข้อมูลเพื่อใช้บริหารพื้นที่หน่วยความจำที่ถูก allocate ตัวอย่างเช่น ในการ allocate เซกเมนต์จาก Global Heap ต้องเสียพื้นที่หน่วยความจำไป 24 ไบท์เสมอไม่ว่าเซกเมนต์ที่ขอ allocate จะมีขนาดใหญ่เท่าไรก็ตาม

Default Data Segment

Windows สามารถรันไฟล์ได้สองชนิดคือ แอพพลิเคชั่นโปรแกรมที่มีนามสกุล .EXE และ Dynamic Link Library ที่มีนามสกุล .DLL โดยทั้งสองเรียกรวมว่า โมดูล (Module) โมดูลในแอพพลิเคชั่นโปรแกรมทำหน้าที่ติดต่อกับผู้ใช้จัดการอินพุทที่ผู้ใช้ติดต่อกับโปรแกรม ประมวลผล แสดงเอาท์พุทให้ผู้ใช้เห็นสำหรับโมดูลในไฟล์ .DLL ทำหน้าที่ให้บริการแก่โมดูลในแอพพลิเคชั่นโปรแกรม โมดูลใน Dynamic-Link Library จะขึ้นมาทำงานก็ต่อเมื่อมีโมดูลอื่นมาเรียกใช้ซึ่งต่างจากโมดูลในแอพพลิเคชั่นโปรแกรมที่จะขึ้นมาทำงานเมื่อมีแมสเสจมาที่แมสเสจคิวของโปรแกรมนั้น โดยปกติโมดูลใน Dynamic-Link Library จะถูก allocate ไว้ในโค้ดเซกเมนต์แบบยกเลิกได้ ดังนั้นโมดูลที่ไม่มีโมดูลอื่นเรียกใช้จะเก็บอยู่บนดิสค์ เมื่อมีโมดูลอื่นเรียกโมดูลนั้นก็จะถูกโหลดมาไว้ในหน่วยความจำเพื่อทำงานต่อไปทำให้เหลือพื้นที่หน่วยความจำให้กับแอพพลิเคชั่นโปรแกรมมากขึ้นระบบ Windows ส่วนใหญ่ประกอบด้วยโมดูลประเภท Dynamic-Link Library ในสามไฟล์หลักได้แก่ USER, KERNEL และ GDI โมดูลประเภทDynamic-Link Library ที่สร้างโดยผู้ใช้เป็นเหมือนความสามารถพิเศษที่ผู้ใช้เพิ่มให้กับระบบ Windows

ทุกโมดูลจะมีดาต้าเซกเมนต์เป็นของตัวเองเรียกว่า Default Data Segment ซึ่งหน้าที่ในการจัดการให้แต่ละโมดูลสามารถเรียกใช้ข้อมูลในดาต้าเซกเมนต์ของตัวเองได้ถูกต้องจะเป็นหน้าที่ของ Windows ในขณะที่กำลังรันโค้ดในโมดูลใดค่าของรีจิสเตอร์ DS จะชี้ไปที่เซกเมนต์ที่เป็น Default Data Segment ของโมดูลนั้น ค่าของรีจิสเตอร์ DS จะเปลี่ยนทุกครั้งที่มีการข้ามขอบเขตของโมดูลการข้ามขอบเขตของโมดูลจะเกิดขึ้นเมื่อมีการเรียกใช้ฟังก์ชั่นที่ถูกประกาศเป็น export function (คือฟังก์ชั่นที่ประกาศในส่วน EXPORT ของไฟล์ DEF โมดูลไม่สามารถเรียกใช้ฟังก์ชั่นนี้ได้โดยตรง Windows เป็นผู้เรียกใช้ฟังก์ชั่นเหล่านี้เอง เช่น window procedure,call-back function, dialog box function เป็นต้น) นอกจากนี้ยังรวมถึง Windows Function ที่ถูกเรียกในโมดูลของแอพพลิเคชั่นโปรแกรม เนื่องจากว่า Windows มีการจัดการดาต้าเซกเมนต์ของโปรแกรมเพียงเซกเมนต์เดียวคือDefault Data Segment ดังนั้นโปรแกรมโดยส่วนใหญ่ที่รันบน Windows จึงถูกคอมไพล์โดยมี memory model เป็นแบบ small model (โปรแกรมประกอบด้วย หนึ่งโค้ดเซกเมนต์ และ หนึ่งดาต้าเซกเมนต์) หรือแบบ medium model (โปรแกรมประกอบด้วยหลายโค้ดเซกเมนต์ และ หนึ่งดาต้าเซกเมนต์) สำหรับโมเดลอื่นที่สามารถมีดาต้าเซกเมนต์ได้หลายเซกเมนต์เช่น Large Model ก็สามารถที่จะรันบนสภาวะของ Windows ได้เช่นกันแต่โปรแกรมประเภทนี้จะมีข้อจำกัดอยู่คือสามารถรันได้เพียงก็อปปี้เดียวและในเรียลโหมดดาต้าเซกเมนต์เหล่านี้จะต้องอยู่กับที่ซึ่งทำให้ขนาดของโลคัลฮีพ(Local Heap) ไม่สามารถขยายได้มากกว่าขนาดที่กำหนดใน HEAPSIZE ของไฟล์ DEF เนื่องจากข้อจำกัดเหล่านี้โปรแกรมส่วนใหญ่ที่รันบนสภาวะของ Windows จึงมีดาต้าเซกเมนต์เดียว ถ้าขนาดของข้อมูลที่ใช้มีขนาดมากกว่าหนึ่งเซกเมนต์โปรแกรมสามารถ Allocate หน่วยความจำเพิ่มจาก Global Heap ได้

Default Data Segment เป็นพื้นที่หน่วยความจำที่ได้จากการ Allocate พื้นที่หน่วยความจำใน Global Heap โดยมีแอททริบิวท์เป็นแบบเคลื่อนย้ายได้ (Moveable Segment) แต่มีลักษณะพิเศษที่แตกต่างจากเซกเมนต์แบบเคลื่อนย้ายได้อันอื่นคือ Default Data Segmentจะถูกล็อคโดยอัตโนมัติ เมื่อโมดูลที่เป็นเจ้าของของดาต้าเซกเมนต์ได้รับเมสเสจ(Messages) ซึ่งเซกเมนต์อื่นที่ไม่ใช่ Default Data Segment โมดูลจะต้องล็อคเซกเมนต์เองเมื่อต้องการใช้ข้อมูลในเซกเมนต์นั้นโดยเรียกใช้ Windows Function ชื่อ GlobalLock และข้อแตกต่างอีกอย่างหนึ่งของ Default Data Segment คือ ข้อมูลในเซกเมนต์สามารถเรียกใช้ได้โดยพอยน์เตอร์แบบ NEAR (ระบุเฉพาะออ

ฟเซท) เพราะว่ารีจิสเตอร์ DS จะถูกเซ็ทให้มาชี้ที่ Default Data Segment โดยปกติ Default Data Segment จะประกอบด้วยเฮดเดอร์ (Header), พื้นที่ของข้อมูลแบบ static, สแต็ก, local heap และอาจจะมี atom table

Header

เฮดเดอร์มีขนาด16 ไบท์ประกอบด้วยพอยน์เตอร์ที่ Windowsใช้ในการบริหาร Default Data Segment พื้นที่ส่วนนี้จะถูก Allocate ตอนคอมไพล์และลิงค์ และจะถูกบริหารโดยWindows ขณะที่รันภายในเฮดเดอร์ประกอบด้วยพอย์เตอร์ 5 ตัว ที่สำคัญที่สุดคือ pLocalHeap ซึ่งชี้ไปที่ตำแหน่งเริ่มต้นของ Local Heap ในดาต้าเซกเมนต์สำหรับ stack จะมีพอยน์เตอร์ชี้อยู่สามตัวคือ pStackBotชี้ตำแหน่งล่างสุดของ stack, pStackMin ชี้ตำแหน่งปัจจุบันของข้อมูลในstack ,pStackTopชี้ตำแหน่งบนสุดของ stack (stack จะเริ่มจากบนลงล่างดังนั้นตำแหน่งของpStackTopจึงอยู่ต่ำกว่า pStackBot)พอยน์เตอร์ทั้งสามมีประโยชน์ในการตรวจสอบ stack overflow ขณะทำการ debug โปรแกรม สำหรับพอยน์เตอร์ตัวสุดท้ายใช้สำหรับชี้ตำแหน่งของ Atom Table ซึ่งใช้เก็บสตริงของตัวอักษรที่มีความยาวต่าง ๆ กันในตารางโดยการอ้างผ่านทาง handle (เลขจำนวนเต็มค่าหนึ่ง) ซึ่งมีขนาดสองไบท์ โดยความเป็นจริงแล้ว Atom Table เป็นส่วนหนึ่งของLocal Heap

Static Data Area

ตัวแปรแบบstatic คือตัวแปรที่ถูกประกาศภายนอกฟังก์ชั่น หรือ ตัวแปรที่มีคีย์เวิร์ด static นำหน้าตอนประกาศชื่อตัวแปรมี LIFE TIME ตลอดโปรแกรม ตัวแปรแบบ static จะถูก allocate ใน default data segment โดยคอมไพล์เลอร์ในพื้นที่ส่วนที่เรียกว่า static data area ดังนั้นตัวแปรแบบ static จึงสามารถโดยการติดต่อแบบ near(เพราะว่ารีจิสเตอร์ DS ชี้อยู่ที่ Default Data Segment) ในพื้นที่นี้นอกจากเป็นพื้นที่หน่วยความจำที่ใช้เก็บค่าของตัวแปรแบบ static แล้วกลุ่มของตัวอักษรที่ปรากฏในโค้ดของโปรแกรมก็จะถูก allocate ไว้ในพื้นที่ส่วนนี้ด้วย

```
char *pchFile,
long lLength;

long FAR PASCAL   WndProc(HWND   hwnd, WORD   wMsg,
                          WORD   wParam, LONG   lParam)
{
static int     lCount;
PAINTSTRUCT    ps;

switch(wMsg)
{
case WM_PAINT:
    BegintPaint(hwnd, &ps);
```

```
TextOut(ps.hdc, 10, 10, "Windows", 7);
EndPaint (hwnd, &ps);
```

จากโค้ดข้างบนนี้คอมไพล์เลอร์จะ allocate พื้นที่ในstatic data area ให้กับตัวแปรสองตัวที่ประกาศนอกฟังก์ชั่นคือ pchFile และ lLength และตัวแปรที่ประกาศในฟังก์ชั่นWndProc ที่มีคีย์เวิร์ด static นำหน้าคือ lCount ตัวแปร lCount จะต่างกับตัวแปรที่ประกาศภายนอกคือตัวแปร lCount จะมองเห็นเฉพาะภายในฟังก์ชั่น WndProc ฟังก์ชั่นอื่นไม่สามารถเรียกใช้ค่าในตัวแปรนี้ได้แต่มี Life Time ตลอดโปรแกรมเหมือนกับตัวแปรที่ประกาศภายนอกฟังก์ชั่น นอกจากตัวแปรทั้งสามตัวที่ถูก allocate พื้นที่ให้ใน static data area แล้วกลุ่มของตัวอักษรคำว่า Windows ในฟังก์ชั่น TextOut ก็จะถูก allocate พื้นที่ไว้ใน static data area ด้วย กลุ่มของตัวอักษรที่ปรากฏในโปรแกรมจะถูก allocate ไว้ใน static data area เสมอถึงแม้นว่าจะมีกลุ่มของตัวอักษรที่เหมือนกันใน static data area แล้วก็ตาม ดังนั้นเพื่อเป็นการประหยัดพื้นที่หน่วยความจำในโปรแกรมที่มีการเรียกใช้กลุ่มของตัวอักษรที่เหมือนกันในโปรแกรมหลายแห่ง กลุ่มของตัวอักษรที่ซ้ำกันควรจะนำมาเก็บไว้ใน string table ซึ่งถือว่าเป็นทรัพยากรอย่างหนึ่งของโปรแกรม เมื่อโปรแกรมต้องการใช้ข้อความใน string table ก็สามารถโหลดมาได้จากดิสค์ เมื่อไม่ต้องการใช้ก็สามารถยกเลิกพื้นที่หน่วยความจำที่ข้อความนั้นถูก allocateอยู่ได้ (ในกรณีนี้ข้อ string table จะไม่ถูก allocate ใน static data area ของ default data segment แต่จะถูก allocate ไว้ในเซกเมนต์ที่ยกเลิกได้)

STACK

สแต็กเป็นพื้นที่หน่วยความจำที่ถูกบริหารโดยภาษาระดับสูงเช่นภาษา C ในซีพียูตระกูล 80X86 มีรีจิสเตอร์ที่ช่วยในการบริหารงานสแต็กโดยเฉพาะคือSS (stack segment สำหรับชี้ตำแหน่งเริ่มต้นของสแต็กเซกเมนต์), BP (base pointer : สำหรับชี้ตำแหน่งที่ต้องการในสแต็ก), SP (stack pointer: สำหรับชี้ตำแหน่งบนสุดของสแต็ก) (โปรแกรมที่รันบน Windows จะมี SS = DS ยกเว้น dynamic-link library) สแต็กจะถูกใช้งานเพื่อส่งพารามิเตอร์ระหว่างฟังก์ชั่น, เก็บแอดเดรสที่จะมาทำงานต่อเมื่อรีเทอร์นกลับจากฟังก์ชั่นที่เรียก, เป็นพื้นที่หน่วยความจำที่ถูก allocate ให้กับตัวแปรแบบ automatic (คือตัวแปรที่ประกาศในฟังก์ชั่นและไม่มีคีย์เวิร์ดstaticนำหน้าจะถูก allocate เมื่อเข้ามาทำงานในฟังก์ชั่น และถูกยกเลิกเมื่อออกจากฟังก์ชั่น)ขนาดพื้นที่ของ stack ใน default data segment จะมีขนาดที่คงที่เท่ากับที่ประกาศไว้ใน STACKSIZE ในไฟล์ DEF โปรแกรมที่มีการใช้ตัวแปรแบบ automatic มากควรที่จะกำหนดขนาดของสแต็กให้มีค่ามากเพราะว่าขนาดของสแต็กไม่สามารถเพิ่มขึ้นได้ถ้าพื้นที่ที่ต้องการมีมากกว่าขนาดของสแต็กที่กำหนดไว้ในตอนแรกขนาดของสแต็กน้อยที่สุดคือ 5KB ถึงแม้นว่าใน STACKSIZE จะกำหนดขนาดของสแต็กน้อยกว่า 5 KB Windows จะทำการ allocate ขนาดของสแต็กให้เท่ากับ 5 KB

ในรูปแสดงการเรียกฟังก์ชั่นในภาษา C และแอสแซมบลี้โค้ด พร้อมทั้งสแต็กหลังจากเรียกฟังก์ชั่นแล้ว ขั้นแรกจะต้องทำการเก็บค่าพารามิเตอร์ที่จะส่งให้กับฟังก์ชั่นที่เรียกลงในสแต็กคำสั่ง CALL จะเก็บค่าแอดเดรสที่จะรีเทอร์นกลับไว้ในสแต็กและกระโดดไปทำงานที่ฟังก์ชั่นที่เรียก ในตอนต้นของฟังก์ชั่นจะมีชุดคำสั่งที่ใช้เซ็ทสแต็ก หลังจากเซ็ทสแต็กเรียบร้อยแล้วรีจิสเตอร์BP จะใช้ในการเรียกใช้ข้อมูลในสแต็กทั้งที่เป็นพารามิเตอร์ที่ส่งมาให้กับฟังก์ชั่นและตัวแปรแบบ automatic ที่ประกาศในฟังก์ชั่นนั้น โดยการอ้างออฟเซ็ทที่เป็นบวกเทียบกับรีจิสเตอร์ BP ก็จะสามารถเรียกใช้ค่าของพารามิเตอร์ที่ส่งมาให้กับฟังก์ชั่นได้เช่น ถ้าต้องใช้ค่าของพารามิเตอร์ตัวที่สาม สามารถเรียกใช้โดยคำสั่ง

```
MOV  AX, [BP+04]
```

ถ้าในฟังก์ชั่นนั้นมีตัวแปรแบบ automatic อยู่สามตัวโดยแต่ละตัวมีขนาดสองไบท์ คำสั่งที่ใช้ในการเซ็ทสแต็กให้กับตัวแปรแบบ automatic ได้แก่

```
SUB  SP, 06
```

การเรียกใช้ตัวแปรแบบ automatic ในสแต็กทำได้โดยการอ้างออฟเซ็ทที่เป็นลบโดยเทียบกับตำแหน่งที่รีจิสเตอร์ BP ชี้อยู่เช่นถ้าต้องการเรียกใช้ตัวแปรแบบ automatic ตัวที่สาม สามารถเรียกใช้โดยคำสั่ง

```
MOV  AX, [BP-06]
```

LOCAL HEAP

พื้นที่ในส่วน Local Heap เป็นพื้นที่หน่วยความจำใน Default Data Segment ที่โปรแกรมสามารถ allocate สำหรับเก็บข้อมูลในขณะรันได้โดยที่ไม่จำเป็นจะต้องจองพื้นที่หน่วยความจำไว้ก่อนเมื่อต้องการใช้จึงทำการ allocate จาก Local Heap โดยปกติใน Default Data Segment จะมี Local Heap อยู่ด้วยเสมอโดยโปรแกรมโหลดเดอร์เป็นผู้เซ็ทพื้นที่หน่วยความจำในส่วนนี้ให้กับโปรแกรมเมื่อเริ่มโหลดโปรแกรมเข้ามาทำงานโดยมีขนาดเริ่มต้นเท่ากับที่กำหนดใน HEAPSIZE ในไฟล์ .DEF พื้นที่ส่วนนี้สามารถเพิ่มขึ้นได้ถ้าโปรแกรมต้องการ allocate พื้นที่หน่วยความจำที่มีขนาดใหญ่กว่าพื้นที่หน่วยความจำที่ว่างใน Local Heap ถ้าดาต้าเซ็กเมนต์นั้นมีแอททริบิวต์เป็นแบบเคลื่อนย้ายได้(Moveable Segment) สำหรับพื้นที่หน่วยความจำที่ allocateได้จาก Local Heap สามารถมีแอททริบิวท์ได้เหมือนกับพื้นที่หน่วยความจำที่ allocate จาก Global Heap คือแบบอยู่กับที่(Fixed Block), แบบเคลื่อนย้ายได้(Moveable Block), แบบยกเลิกได้(Discardable Block) พื้นที่หน่วยความจำที่ allocate ได้จาก Local Heap จะเป็น blockของข้อมูลโดยการจัดเรียงพื้นที่แต่ละส่วนมีลักษณะเหมือนกับการจัดเรียงใน Global Heap คือพื้นที่หน่วยความจำแบบอยู่กับที่จะอยู่ล่างสุดพื้นที่หน่วยความจำแบบยกเลิกได้จะอยู่บนสุดโดยมีพื้นที่หน่วยความจำแบบเคลื่อนย้ายได้อยู่ระหว่างกลาง

Windows มีฟังก์ชั่นที่ใช้จัดการเกี่ยวกับLocal Heap อยู่ 12 ฟังก์ชั่นแสดงในรูปโดยปกติเมื่อต้องการ allocate พื้นที่หน่วยความจำจาก Local Heap ควรที่จะใช้ฟังก์ชั่นที่Windowsจัดมาให้เพราะว่าสามารถ allocate พื้นที่หน่วยความจำที่มีแอททริบิวท์ต่างๆ กันได้โดยที่ฟังก์ชั่น malloc ของภาษา C สามารถ allocate พื้นที่หน่วยความจำแบบอยู่กับที่ได้เพียงอย่างเดียว

เช่นเดียวกับพื้นที่หน่วยความจำแบบเคลื่อนย้ายได้ใน Global Heap การเรียกใช้ข้อมูลในพื้นที่หน่วยความจำแบบเคลื่อนย้ายได้จะติดต่อผ่านทาง HANDLE ก่อนที่จะใช้ข้อมูลในพื้นที่หน่วยความจำส่วนนี้ได้จะต้องทำการล็อคข้อมูลก่อนโดยการเรียกใช้ Windows Function ชื่อ LocalLock ฟังก์ชั่นนี้จะรีเทอร์นแอดเดรสของข้อมูลให้พร้อมทั้งเพิ่มค่า Lock Count ขึ้นหนึ่ง Windows ไม่สามารถเคลื่อนย้ายข้อมูลที่ถูกล็อคได้จนกว่าค่า Lock Count จะเป็นศูนย์การล็อคข้อมูลใน Local Heap ไม่ควรที่จะล็อคข้ามแมสเสจควรที่จะ unlock ทันทีที่ไม่ต้องการใช้ข้อมูลนั้นแล้ว

```
HANDLE    hMem,
PSTR      pstr,

/* Allocate a 15-byte moveable block */
hMem = LocalAlloc(LMEM_MOVEABLE, 15),
if (!hMem)
    return (ERROR),

/* Lock the block, getting a pointer */
pstr = LocalLock(hMem),
if (!pstr)
    return(ERROR),

lstrcpy(pstr, "Hello World!"),

/* Unlock Block */
LocalUnlock (hMem),
```

จากโปรแกรมข้างบนแสดงการ allocate พื้นที่หน่วยความจำใน Local Heap และการใช้พื้นที่หน่วยความจำแบบเคลื่อนย้ายได้ใน Local Heapจะเห็นว่าต้องมีการตรวจสอบค่าที่รีเทอร์นกลับมาเมื่อทำการล็อคข้อมูลหรือ allocate ข้อมูลเสมอเพราะว่าพื้นที่หน่วยความจำที่เหลืออยู่อาจจะไม่พอที่ขอไป(โดยปกติพื้นที่หน่วยความจำที่รี

เทอร์นให้ต้องมีขนาดไม่น้อยกว่าที่ขอมา) หรือเซกเมนต์ที่ข้อมูลที่ต้องการจะล็อคไม่อยู่ในหน่วยความจำก็เป็นได้

Global Heap VS Local Heap

โปรแกรมที่รันบนสภาวะของ Windows สามารถ allocate พื้นที่หน่วยความจำได้จากสองพื้นที่คือ

1.จาก Local Heap ใน Default Data Segment ของโปรแกรมนั้น พื้นที่หน่วยความจำที่allocate ได้จะเป็นบล็อคของข้อมูลมีขนาดไม่เกิน 64 KBytes เนื่องจากพื้นที่ใน Default Data Segmentส่วนหนึ่งถูกใช้ในการเก็บตัวแปรแบบ static และสแต็กของโปรแกรมทำให้เหลือพื้นที่สำหรับ Local Heap ประมาณ 30 Kbytes - 5000 มูลที่มีขนาดเล็กๆ ข้อเสียคือมีพื้นที่หน่วยความจำให้ใช้น้อย

2.จาก Global Heap ที่ดูแลโดย Windows พื้นที่หน่วยความจำที่ allocate ได้จะเป็นเซกเมนต์ขนาดไม่เกิน 64 KBytes โปรแกรมสามารถที่จะเข้ามาใช้งานพื้นที่ในหน่วยความจำนี้ได้ทั้งหมดโดยไม่ต้องกันพื้นที่ไว้สำหรับเก็บข้อมูลชนิดอื่นข้อดีคือสามารถallocate พื้นที่หน่วยความจำกี่เซกเมนต์ก็ได้ทำให้โปรแกรมมีพื้นที่หน่วยความจำที่สามารถ allocate ได้มากเท่ากับพื้นที่หน่วยความจำที่เหลือในระบบขณะนั้น(ในความเป็นจริงจำนวนเซกเมนต์ที่สามารถ allocate ทั้งหมดในระบบต้องไม่เกิน 8192 เซกเมนต์ ใน Real Mode และ 386 Enhance Mode สำหรับใน Standard Mode จำนวนเซกเมนต์ทั้งหมดต้องไม่เกิน 4096 เซกเมนต์) ข้อเสียคือมี overhead สูงเท่ากับ 24 ไบท์ต่อหนึ่งเซกเมนต์ จึงไม่เหมาะสำหรับข้อมูลที่มีขนาดเล็ก

ในWindows เวอร์ชั่น 3.1 ได้ตัดโหมดการทำงานใน Real Mode ออกไปทำให้การบริหารหน่วยความจำของ Windows เปลี่ยนไปเล็กน้อย คือ Windows ในเวอร์ชั่น 3.1 จะมีการทำงานในเฉพาะ Protect Mode เท่านั้น การอ้างแอดเดรสใน Protect Mode ไม่ใช้การอ้างผ่านทาง SEGMENT.OFFSET แต่เป็นการอ้างโดยใช้ SELECTOR OFFSET โดยที่ SELECTORเป็นตัวชี้ตำแหน่งใน Descriptor Table (ตารางที่ใช้เก็บข้อมูลของเซกเมนต์ต่างๆ ในระบบเช่นแอดเดรสที่เริ่มต้น, ขนาดของเซกเมนต์, แอททริบิวท์ต่างๆ ของเซกเมนต์ เป็นต้น) ซึ่ง SELECTOR จะมีลักษณะคล้ายกับ HANDLE ที่ใช้ใน REAL MODE เมื่อ Windows มีการย้ายตำแหน่งของเซกเมนต์ Windows จะเข้าไปแก้ไขแอดเดรสเริ่มต้นใหม่ของเซกเมนต์ที่ย้ายไปใน Descriptor Table โดยที่ SELECTOR ยังคงค่าเดิมดังนั้นการอ้างถึงข้อมูลในเซกเมนต์คงเป็นข้อมูลที่ถูกต้องการล็อคเซกเมนต์ในพื้นที่หน่วยความจำแบบเคลื่อนย้ายได้จึงเป็นการเพิ่มค่า Lock Count ของเซกเมนต์นั้นแต่เซกเมนต์ยังสามารถเคลื่อนย้ายได้อยู่โดยที่ไม่ต้องเปลี่ยนค่าของ Selector ของเซกเมนต์นั้นการล็อคเซกเมนต์จะมีผลเฉพาะเซกเมนต์ที่ยกเลิกได้เพราะว่าWindowsไม่สามารถยกเลิกเซกเมนต์แบบยกเลิกได้ที่มีค่า Lock Count มากกว่าศูนย์ได้

Global Heap + Local Heap

โดยปกติแล้วพื้นที่หน่วยความจำ Local Heap จะอยู่ใน Default Data Segment เท่านั้นแต่โปรแกรมก็สามารถที่จะสร้างLocalHeapในดาต้าเซกเมนต์อื่นที่ได้จากการ allocate จาก Global Heap โดยใช้คำสั่ง LocalInit(เป็นฟังก์ชั่นที่ไม่มีเอกสารกำกับในคู่มือของ SDK)

```
HANDLE    hMem,
int       pStart, pEnd;
LPSTR     lp,
WORD      wSeg,

hMem = GlobalAlloc(GMEM_MOVEABLE, 4096); /* allocate segment */
if (!hMem)
    goto ErrorOut,

lp = GlobalLock(hMem); /* lock segment, return far pointer */
wSeg = HIWORD(lp), /* segment value in high word */

pStart = 16, /* reserve 16 bytes for header */
pEnd = (int)GlobalSize(hMem) - 1,
LocalInit(wSeg, pStart, pEnd);
GlobalUnlock(hMem),
GlobalUnlock(hMem),
```

จากชุดคำสั่งข้างบนแสดงการ allocate ดาต้าเซกเมนต์จาก Global Heap และการ initial Local Heap ในเซกเมนต์ที่ allocateได้ จะเห็นว่าโปรแกรมมีการสงวนพื้นที่ไว้16 ไบท์เพื่อเป็นเฮดเดอร์ของดาต้าเซกเมนต์ โดยที่ออฟเซ็ท 6H เป็นพอย์นเตอร์ชี้ตำแหน่งเริ่มต้นของ Local Heap ในดาต้าเซกเมนต์ และในตอนท้ายของชุดคำสั่งมีการ unlock เซกเมนต์สองครั้ง โดยครั้งแรกเป็นการ unlock ที่เกิดจากการล็อคเซกเมนตจากคำสั่งข้างบน ครั้งที่สองเป็นการ unlock เนื่องจากคำสั่ง LocalInit เพราะว่าคำสั่งนี้จะล็อคเซกเมนต์ค้างไว้

โปรแกรมสามารถใช้งาน Local Heap ที่สร้างขึ้นมาในดาต้าเซกเมนต์ใหม่นี้ได้เหมือนกับ Local Heap ที่อยู่ใน Default Data Segment ดังนั้นโปรแกรมสามารถมี Local Heap ได้หลายๆ ที่ในดาต้าเซกเมนต์ที่ต่างกัน

ปัญหาในการใช้รูทีนที่จัดการกับ Local Heap ในดาต้าเซกเมนต์ที่ไม่ใช่ Default Data Segment คือ รีจิสเตอร์ DS จะต้องเก็บค่าของเซกเมนต์ หรือ selector ของเซกเมนต์ที่ Local Heap อยู่ วิธีการคือใช้ inline assembly ซึ่งสามารถทำได้ใน Microsoft C เวอร์ชั่น 6.0 ขึ้นไป เพื่อเปลี่ยนค่ารีจิสเตอร์ DS เป็นค่าเซกเมนต์ หรือ selector ของเซกเมนต์ที่ Local Heap อยู่

```
LPSTR     lp,
HANDLE    hMem,
WORD      wHeapDS,
```

```
lp = GlobalLock(hMem), /* lock segment, return far pointer */
wHeapDS = HIWORD(lp), /* segment value in high word */
_asm {
    push    DS /* save segment value of default data segment */
    mov     AX, wHeapDS
    mov     AX, DS /* change content in DS */
    }
hMem = LocalAlloc(LMEM_MOVEABLE, 16),
_asm {
    pop     DS /* restore old DS */
    }
GlobalUnlock (hmem),
```

จากชุดคำสั่งข้างบนแสดงการ allocate พื้นที่หน่วยความจำจาก Local Heap ที่อยู่ในดาต้าเซกเมนต์อื่น เมื่อเปลี่ยนค่าของรีจิสเตอร์ DS เป็นค่าของเซกเมนต์หรือ selector ของดาต้าเซกเมนต์อื่นแล้ว โปรแกรมไม่สามารถที่จะใช้ค่าในตัวแปร static ที่อยู่ใน Default Data Segment ได้

ในการใช้ข้อมูลใน Local Heap ที่ allocate ได้โปรแกรมจะต้องทำการล็อกข้อมูลทั้งสองระดับคือล็อกเซกเมนต์ที่ Local Heap อยู่โดยใช้ฟังก์ชั่น GlobalLock และล็อกบล็อกที่ข้อมูลอยู่โดยใช้ LocalLock และเพื่อให้ Windows บริหารหน่วยความจำได้อย่างมีประสิทธิภาพ เมื่อเลิกใช้ข้อมูลแล้วโปรแกรมต้องทำการ unlock ข้อมูลทั้งสองระดับเช่นกัน

MESSAGE

MESSAGE

MESSAGE เป็นอินพุทที่ Windows ใช้ติดต่อกับแอพพลิเคชั่น Windows จะส่ง Message ไปให้กับแอพพลิเคชั่นเมื่อต้องการให้แอพพลิเคชั่นตอบสนองต่อการเปลี่ยนแปลงต่างๆที่เกี่ยวกับแอพพลิเคชั่นนั้นตัวอย่างเช่นเมื่อหน้าต่างของแอพพลิเคชั่นนั้นมีความเสียหายเกิดขึ้น(อาจจะเนื่องจากมีการเลื่อนหน้าต่างของแอพพลิเคชั่นอื่นมาทับเป็นต้น)Windows จะส่ง MESSAGE ที่เป็นการบอกให้แอพพลิเคชั่นนั้นทำการแก้ไขข้อมูลที่แสดงในหน้าต่างให้ถูกต้อง

MESSAGE ถูกประกาศให้เป็นตัวแปรแบบ structure ในภาษา C ภายในประกอบด้วยฟิลด์ที่ใช้แสดงหมายเลขประจำตัวของ MESSAGE, พารามิเตอร์ของ MESSAGE นั้นโดยความหมายของพารามิเตอร์เหล่านี้ขึ้นอยู่กับชนิดของ MESSAGE รูปแบบของ structure ของ MESSAGE เป็นดังนี้

```
typedef struct tagMSG
{
    HWND   hwnd,     ค่า handle ของวินโดว์ที่แมสเสจจะถูกส่งไป
    WORD   message;  Message ที่จะส่ง
    WORD   wParam;   Message parameter(16 bits)
    LONG   lParam;   Message parameter(32 bits)
    DWORD  time;     เวลาขณะที่ Message ถูกส่งมาที่ Application queue
    POINT  pt,       ตำแหน่งของ mouse ขณะที่ Message ถูกส่งมาที่ Application queue
}
```

การสร้าง และ ใช้งาน MESSAGE

WINDOWS สร้างอินพุทแมสเสจเมื่อมีการป้อนอินพุทเกิดขึ้นเช่น ผู้ใช้เลื่อน MOUSE หรือกดคีย์บอร์ด WINDOWS จะรวบรวมอินพุทเหล่านี้ไว้ในคิวของ WINDOWS เองเรียกว่า"HARDWARE QUEUE" หลังจากนั้น WINDOWS จะส่ง MESSAGE เหล่านี้ไปให้กับแอพพลิเคชั่นต่าง ๆ โดยนำ MESSAGE ไปใส่ใน APPLICATION QUEUE (ซึ่งเป็นคิวเฉพาะของแต่ละแอพพลิเคชั่น) โดยปกติ MESSAGE ที่เข้ามาในคิวก่อนจะถูกนำออกไปประมวลผลก่อน(FIRST IN FIRST OUT) ยกเว้น WM_TIMER(MESSAGE ของ TIMER), WM_PAINT(MESSAGE ที่ให้ซ่อมแซมหน้าต่าง) และ WM_QUIT(MESSAGE ที่จะจบการทำงาน) จะถูกนำไปประมวลผลก็ต่อเมื่อแอพพลิเคชั่นได้ประมวลผล MESSAGE อื่นในคิวหมดแล้วเมื่อ WINDOWS ใส่ MESSAGE ใน APPLICATION QUEUE แล้ว แอพพลิเคชั่นจะอ่าน MESSAGE จากคิวของแอพพลิเคชั่นเองโดยใช้ฟังก์ชั่น GetMessage และส่ง MESSAGE ที่ได้ไปให้ window procedure ทำการประมวลผลต่อไป โดยใช้ฟังก์ชั่น -DispatchMessage

บางที WINDOWS สามารถที่จะส่ง MESSAGE ไปให้กับ window procedure ได้โดยตรงโดยที่ไม่ต้องส่ง MESSAGE ไปที่คิวของแอพพลิเคชั่นนั้นก่อน MESSAGE ชนิดนี้เรียกว่า "UNQUEUE MESSAGE" ฟังก์ชั่น SendMessage ใช้ส่ง MESSAGE ไปที่ window procedure โดยปกติ MESSAGE เหล่านี้จะเกี่ยวข้องกับหน้าต่างเพียงอย่างเดียวตัวอย่างเช่น ฟังก์ชั่น CreateWindow จะทำให้ WINDOWS ส่ง WM_CREATE ไปที่ window procedure ของแอพพลิเคชั่นและรอจนกระทั่ง window procedure ได้ทำการประมวลผล MESSAGE นั้นเสร็จเรียบร้อย WINDOWS ส่ง WM_CREATE ไปที่ window procedure โดยตรงไม่ต้องผ่านแอพพลิเคชั่นคิวก่อน

นอกจาก WINDOWS จะเป็นผู้สร้าง MESSAGE แล้วแอพพลิเคชั่นก็สามารถสร้าง MESSAGE และส่ง MESSAGE เหล่านั้นไปที่แอพพลิเคชั่นคิวของตัวเองหรือของแอพพลิเคชั่นอื่นก็ได้ แอพพลิเคชั่นต่างๆโดยปกติประกอบด้วยฟังก์ชั่น WinMain เป็นฟังก์ชั่นหลัก (เหมือนกับฟังก์ชั่น main ของโปรแกรมที่รันบน DOS ปกติ) ฟังก์ชั่นนี้นอกจากจะเซ็ทค่าเริ่มต้นต่างๆ ตั้งแต่สร้าง windows class ขึ้นมาใหม่ ลงทะเบียนคลาสใหม่ที่สร้างขึ้นมาบอกลักษณะของหน้าต่างของแอพพลิเคชั่น สร้างหน้าต่างในตอนท้ายของฟังก์ชั่นนี้จะมีลูปที่วนรับ MESSAGE

จากแอพพลิเคชั่นคิวของตัวเองเพื่อส่งต่อไปให้กับ window procedure ประมวลผลต่อไป เรียกว่า"MESSAGE LOOP" โดยปกติใน MESSAGE LOOP นี้จะใช้ฟังก์ชั่น GetMessage ในการนำเอา MESSAGE ออกจากแอพพลิเคชั่นคิว GetMessage จะไปตรวจสอบที่แอพพลิเคชั่นคิว ถ้ามี MESSAGEในแอพพลิเคชั่นคิวก็จะรีเทอร์น MESSAGE ที่อยู่บนสุดของคิวกลับมา ถ้าคิวว่างฟังก์ชั่น GetMessage จะรอจนกระทั่งมี MESSAGE เข้ามาที่แอพพลิเคชั่นคิว ในขณะที่รอให้มี MESSAGE เข้ามาที่คิวฟังก์ชั่น GetMessage ก็คืนการใช้งานซีพียูให้กับ WINDOWS เพื่อนำไปให้กับแอพพลิเคชั่นอื่นที่มี MESSAGE อยู่ในคิว ด้วยการใช้ฟังก์ชั่นนี้จึงทำให้แอพพลิเคชั่นอื่นสามารถขึ้นมารันได้ แอพพลิเคชั่นอื่นไม่สามารถขึ้นมารันได้เลยถ้าแอพพลิเคชั่นที่กำลังใช้งานซีพียูไม่มี MESSAGE LOOP จึงทำให้ระบบ WINDOWS เป็นระบบมัลติทาสกิงแบบ COOPERATIVE MULTITASKING

WINDOWS จะสร้าง MESSAGE ขึ้นมาทุกครั้งที่มีการกดคีย์บอร์ด ใน MESSAGE จะบรรจุ virtual-key codeเพื่อแสดงว่าคีย์ไหนถูกกด แต่ไม่แสดงรหัสของตัวอักษรของคีย์ที่ถูกกด เพื่อให้ได้รับค่าของตัวอักษรที่ถูกกด ใน MESSAGE LOOPควรจะมีฟังก์ชั่นที่ทำการแปลง virtual-key code ให้เป็นรหัสของตัวอักษรโดยใช้ฟังก์ชั่น TranslateMessage ฟังก์ชั่นนี้จะสร้าง MESSAGE ที่มีรหัสของคีย์ที่ถูกกดส่งมาด้วย (ชื่อ WM_CHAR) แล้วส่งไปที่แอพพลิเคชั่นคิวของแอพพลิเคชั่นนั้น โดยปกติ MESSAGE LOOP ควรที่จะใช้ฟังก์ชั่น TranslateMessage ทุก MESSAGE ที่ได้รับเพื่อเป็นการรับประกันว่า MESSAGE ของคีย์บอร์ดอินพุทถูกแปลงอย่างถูกต้อง (ถ้าเป็น MESSAGE อื่นฟังก์ชั่นนี้จะไม่มีผลต่อ MESSAGE นั้น)

โค้ดข้างล่างนี้แสดงต้นแบบของ MESSAGE LOOP ในฟังก์ชั่น WinMain เพื่อทำการรับ MESSAGE จากแอพพลิเคชั่นคิวและส่ง MESSAGE ต่อไปให้กับ window procedure ทำงานต่อไป

```
int PASCAL WinMain(HINSTANCE hinstCurrent, HINSTANCE hinstPrevious,
                   LPSTR lpszCmdLine, int nCmdShow)
{
    MSG msg;



    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg); /* translate virtual-key code */
        DispatchMessage(&msg); /* dispatches message to window */
    }

    return (int)msg.wParam;
}
```

การส่ง MESSAGE

นอกจาก WINDOWS จะเป็นผู้ส่ง MESSAGE แล้วแอพพลิเคชั่นก็สามารถที่จะส่ง MESSAGE เองได้เพื่อที่จะติดต่อกับหน้าต่างอื่นฟังก์ชั่น SendMessage และ PostMessage ส่ง MESSAGE ให้กับหน้าต่างของแอพพลิเคชั่นอื่น

ฟังก์ชั่นPostMessage จะทำให้ WINDOWS นำ MESSAGE ไปใส่ไว้ในแอพพลิเคชั่นคิวของหน้าต่างที่ต้องการจะติดต่อด้วย ฟังก์ชั่นนี้จะรีเทอร์นกลับมาที่แอพพลิเคชั่นที่เรียกฟังก์ชั่นนี้แล้วทำงานอย่างอื่นต่อไป สำหรับ MESSAGE ที่ส่งไปนั้นจะถูกนำขึ้นมาประมวลผลก็ต่อเมื่อ MESSAGE นั้นถูกเรียกขึ้นมาจากแอพพลิเคชั่นคิว ฟังก์ชั่นSendMessage จะทำให้ WINDOWS ส่ง MESSAGE ไปที่ window procedure ของหน้าต่างที่ต้องการจะติดต่อด้วยโดยไม่ต้องผ่านแอพพลิเคชั่นคิวของหน้าต่างนั้น WINDOWS จะยังไม่รีเทอร์นกลับไปที่แอพพลิเคชั่นที่เรียกใช้ฟังก์ชั่นนี้จนกระทั่ง window procedure ของหน้าต่างที่ส่ง MESSAGE ไปได้ประมวลผล MESSAGE นั้นเสร็จเรียบร้อยหรือคืนการควบคุมให้โดยเรียกใช้ฟังก์ชั่น ReplyMessage

แอพพลิเคชั่นที่ต้องใช้ผลที่รีเทอร์นกลับมาเมื่อส่ง MESSAGE ไปแล้ว ควรจะใช้ฟังก์ชั่น SendMessage ในการส่ง MESSAGE ค่าที่รีเทอร์นกลับมานั้นจะเหมือนกับค่าที่รีเทอร์นโดย window procedure หลังจากประมวลผล MESSAGE นั้นแล้ว สำหรับฟังก์ชั่น PostMessage จะรีเทอร์นกลับมาทันที ดังนั้นค่าที่รีเทอร์น จึงเป็นค่าทางลอจิกคือ TRUE เมื่อ MESSAGE ถูกใส่ในแอพพลิเคชั่นคิวแล้วหรือ FALSE เมื่อไม่สามารถใส่ MESSAGE ในแอพพลิเคชั่นคิวได้

MESSAGE DEADLOCK

โดยปกติเมื่อแอพพลิเคชั่นเรียกใช้ฟังก์ชั่น SendMessage แอพพลิเคชั่นนั้นจะรอจนกว่าฟังก์ชั่น SendMessage จะรีเทอร์นกลับมาจาก window procedure ที่ส่ง MESSAGE ไป เมื่อ windows procedure ที่ได้รับ MESSAGE คืนการใช้งานซีพียูให้กับระบบ แอพพลิเคชั่นที่ส่ง MESSAGE มาไม่สามารถที่จะรันต่อไปได้เพราะต้องรอให้ฟังก์ชั่น SendMessage รีเทอร์นกลับมาก่อนทำให้เกิดสภาพที่เรียกว่า "MESSAGE DEADLOCK" การหลีกเลี่ยง MESSAGE DEADLOCK ทำได้โดยแอพพลิเคชั่นที่ประมวลผล MESSAGE ต้องไม่คืนการใช้งานซีพียูให้กับระบบโดยตรงโดยการเรียกใช้ฟังก์ชั่นต่อไปนี้

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

ดังนั้นก่อนที่แอพพลิเคชั่นที่ประมวลผลMESSAGE จะเรียกใช้ฟังก์ชั่นเหล่านี้ ก็ควรที่จะเรียกใช้ฟังก์ชั่น InSendMessage ก่อนเพื่อตรวจสอบว่าMESSAGE ที่กำลังประมวลผลอยู่เกิดจากการเรียกใช้ฟังก์ชั่น SendMessage ของฟังก์ชั่นอื่นหรือไม่ ถ้าฟังก์ชั่น InSendMessage รีเทอร์นค่าที่ไม่ใช่ศูนย์ window procedure ต้องเรียกฟังก์ชั่น ReplyMessage ก่อนที่จะเรียกใช้ฟังก์ชั่นข้างบนนี้

การตรวจสอบ MESSAGE ใน Application queue

โดยปกติถ้ามีการทำงานในแอพพลิเคชั่นใดนานๆ แอพพลิเคชั่นตัวอื่นที่อยู่ในระบบจะไม่มีโอกาสขึ้นมารัน เพื่อให้แอพพลิเคชั่นตัวอื่นได้มีโอกาสขึ้นมาทำงาน ในขณะที่แอพพลิเคชั่นมีงานที่ต้องทำนานๆ ควรที่จะตรวจสอบแอพพลิเคชั่นคิวของตัวเองเสมอๆโดยถ้าไม่มี MESSAGE ในคิวแล้วจะได้ให้แอพพลิเคชั่นตัวอื่นสลับขึ้นมาทำงาน การตรวจสอบว่าในแอพพลิเคชั่นคิวมี MESSAGE อยู่หรือไม่ทำได้โดยใช้ฟังก์ชั่น PeekMessage ฟังก์ชั่นนี้จะตรวจสอบในคิวว่ามี MESSAGE เข้ามาหรือไม่โดยไม่ดึง MESSAGE ออกจากคิว ฟังก์ชั่นจะรีเทอร์นค่าจริงถ้ามี MESSAGEอยู่ในคิวมิฉะนั้นจะรีเทอร์นค่าเท็จ ฟังก์ชั่นนี้จะคืนการใช้ซีพียูให้กับแอพพลิเคชั่นอื่นถ้าในคิวไม่มี MESSAGE

ฟังก์ชั่น PeekMessage ต่างจากฟังก์ชั่น GetMessage ที่ฟังก์ชั่นนี้จะรีเทอร์นค่าเท็จเมื่อไม่มี MESSAGE ในคิวแต่ฟังก์ชั่น GetMessage จะรีเทอร์นค่าเท็จเมื่อ MESSAGE ที่อยู่ต้นคิวเป็น WM_QUIT (MESSAGE ที่บอกการจบโปรแกรม)ถ้าเป็น MESSAGE อื่นรีเทอร์นค่าจริง สำหรับข้อแตกต่างที่สำคัญอีกประการหนึ่งคือ GetMessage ไม่รีเทอร์นกลับมาจนกระทั่งมี MESSAGE เข้ามาในคิว แต่ฟังก์ชั่น PeekMessage รีเทอร์นกลับมาไม่มาไม่ว่าจะมี MESSAGE เข้ามาที่คิวหรือไม่ การรีเทอร์นของ PeekMessage จะรีเทอร์นเมื่อมี MESSAGE เข้ามาหรือแอพพลิเคชั่นอื่นไม่มี MESSAGE ในคิวของตัวเอง

```
BOOL  PeekMessage(lpmsg, hwnd, uFilterFirst, uFilterLast, fuRemove)
MSG       FAR *lpmsg,
HWND          hwnd;
UINT          uFilterFirst;
UINT          uFilterLast,
UINT          fuRemove;
```

ฟังก์ชั่น PeekMessage รีเทอร์นค่าที่เป็นจริงหรือเท็จเท่านั้น สำหรับพารามิเตอร์ต่าง ๆ มีความหมายดังนี้

lpmsg far พอย์นเตอร์ชี้ไปที่ MSG structure ซึ่งใช้รับข้อมูลของ MESSAGE จากแอพพลิเคชั่นคิว

hwnd เลขประจำตัวของหน้าต่างที่เป็นเจ้าของ MESSAGE ถ้าเป็น NULL หมายถึง หน้าต่างทุกบานที่เป็นของแอพพลิเคชั่นที่เรียกฟังก์ชั่นนี้

uFilterFirst กำหนดค่าเริ่มต้นของช่วงของ MESSAGE ที่จะตรวจสอบ

uFilterLast กำหนดค่าสุดท้ายของช่วงของ MESSAGE ที่จะตรวจสอบ

fuRemove บอกวิธีการจัดการกับ MESSAGE ในคิว ค่าต่าง ๆ ของพารามิเตอร์นี้สามารถรวมกันได้โดยการ OR ซึ่งค่าต่าง ๆ มีดังนี้

PM_NOREMOVE MESSAGE ไม่ถูกดึงออกจากคิวหลังจากทำฟังก์ชั่น PeekMessage

PM_NOYIELD ไม่คืนการใช้งานซีพียูให้กับแอพพลิเคชั่นอื่นแม้นว่าจะไม่มี MESSAGE ในคิวของตัวเองก็ตาม

ถ้าพารามิเตอร์ uFilterLast และ uFilterFirst เป็นศูนย์ทั้งคู่ ฟังก์ชั่น PeekMessage จะตรวจสอบ MESSAGE ทุก MESSAGE ที่อยู่ในคิว สำหรับ WM_PAINT ฟังก์ชั่น PeekMessage ไม่ดึงออกจากคิวจนกว่า MESSAGE จะถูกประมวลผลในขณะที่แอพพลิเคชั่นอยู่ใน PeekMessage ลูป ซีพียูจะทำงานตลอดเวลา ดังนั้นแอพพลิเคชั่นไม่ควรอยู่ใน PeekMessage ลูปถ้าไม่จำเป็น

การใช้งานฟังก์ชัน PeekMessage

แอพพลิเคชั่นใน WINDOWS ส่วนมากใช้ฟังก์ชั่น PeekMessagae ในการทำ Background Processing ซึ่งวิธีนี้แอพพลิเคชั่นต้องระวังให้ระบบอยู่ในสถานะว่าง(idle) ทันทีที่แอพพลิเคชั่นทำ Background Processingเสร็จแล้วเพื่อที่งานที่ทำในสถานะว่างของระบบเช่น การทำออพติไมซ์ page table, การบริหารกระแสไฟฟ้าในระบบกรณีที่ใช้แบตเตอรี่ เป็นต้น ได้มีโอกาสขึ้นมาทำมิฉะนั้นประสิทธิภาพการทำงานของระบบจะตกลง

แอพพลิเคชั่นส่วนใหญ่ใช้ฟังก์ชั่น PeekMessage เพื่อที่จะดึง MESSAGE ระหว่างการทำงานที่ใช้เวลานานซึ่งต้องทำเป็น Background Processing เช่นการพิมพ์เอกสารทางพรินเตอร์, การคำนวณข้อมูลขนาดมากๆ เป็นต้น ที่ไม่ใช้ฟังก์ชั่น GetMessage เพราะว่า GetMessage ต้องรอให้มี MESSAGE เข้ามาในคิวก่อนที่จะรีเทอร์นกลับมา

แอพพลิเคชั่นไม่ควรที่จะใช้ฟังก์ชั่น PeekMessage ถ้าไม่มีการทำงานใน Background ถ้าแอพพลิเคชั่นเพียงแค่คอย MESSAGE เข้ามาแอพพลิเคชั่นควรใช้ฟังก์ชั่น GetMessage หรือ WaitMessage แทน แอพพลิเคชั่นที่อยู่ใน PeekMessage loop จะได้ใช้ซีพียูบ่อยกว่าแอพพลิเคชั่นตัวอื่นซึ่งเป็นการแย่งเวลาการใช้ซีพียูจากแอพพลิเคชั่นอื่น

โค้ดในหน้าถัดไปแสดง PeekMessage loop ที่ไม่ยอมให้ซีพียูอยู่ในสถานะว่าง

```
for (;;)
{
    while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    {
        if (msg.message == WM_QUIT)
            return (TRUE);
        TranslateMessage(&msg);
```

```
        DispatchMessage(&msg);
    }
    BackgroundProcessing();
}
```

โค้ดข้างบนนี้สามารถที่จะเขียนใหม่ได้สองทาง ซึ่งต่างก็มีคุณสมบัติดังนี้

1. โค้ดใหม่ทั้งคู่จะประมวลผล MESSAGE ที่เข้ามาก่อนที่จะทำ Background Process ทำให้การตอบสนองผู้ใช้เป็นไปอย่างรวดเร็ว

2.แอพพลิเคชั่นจะรอรับ MESSAGE (อยู่ในสถานะว่าง) เมื่อ Background Process เสร็จเรียบร้อย

วิธีที่หนึ่งมีโค้ดดังข้างล่างนี้

```
// Improved PeekMessage loop
for (;;)
{
    while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    {
        if (msg.message == WM_QUIT)
            return TRUE;

        TranslateMessage(&msg),
        DispatchMessage(&msg);
    }

    if (IfBackgroundProcessingRequired())
        BackgroundProcessing();
    else
        WaitMessage();
}
```

สำหรับวิธีที่สองแสดงดังนี้

```
//another improved PeekMessage loop
for (;;)
{
    for (;;)
    {
        if (IfBackgroundProcessingRequired())
        {
            if (!PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
                break;
        }
        else
            GetMessage(&msg, NULL, 0, 0, 0);

        if (msg.message == WM_QUIT)
            return TRUE;

        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    BackgroundProcessing();
}
```

ขั้นตอนการสร้างโปรแกรม

สำหรับขั้นตอนการสร้างโปรแกรมแบ่งออกเป็นสองขั้นใหญ่ ๆ ดังนี้

1 ขั้นออกแบบ และสร้างโปรแกรมต้นแบบ :- ในขั้นนี้เป็นการออกแบบ data structure หลักของโปรแกรม รวมทั้งตารางย่อยต่าง ๆ ในโปรแกรมเนื่องจากการออกแบบ data structure ไม่สามารถที่จะพิสูจน์โดยวิธีการใดวิธีการหนึ่งได้ว่าเหมาะสมกับโปรแกรมตามที่ต้องการหรือไม่ ดังนั้นวิธีที่เหมาะสมที่สุดในการตรวจสอบว่า data structure และโครงสร้างโปรแกรมส่วนอื่น ๆ สามารถทำงานได้ตามที่ต้องการคือการสร้างโปรแกรมต้นแบบเพื่อทดสอบ data structure รวมทั้งองค์ประกอบต่าง ๆ ในโปรแกรมว่าสามารถทำงานร่วมกันได้อย่างมีประสิทธิภาพหรือไม่ โดยที่โปรแกรมต้นแบบนี้เป็นโปรแกมที่เขียนง่าย ๆ ใช้เวลาในการพัฒนาไม่นานนัก สามารถแก้ไขเพิ่มเติมได้อย่างรวดเร็ว มีฟังก์ชั่นการทำงานพื้นฐานพอเพียงที่สามารถใช้ตรวจสอบประสิทธิภาพการทำงานโดย

รวมของระบบ และตรวจสอบความยืดหยุ่นของโครงสร้างที่ออกแบบไว้

2. ขั้นการ implement :- หลังจากได้สร้างโปรแกรมตัวอย่างและทดสอบจนมั่นใจว่าองค์ประกอบหลักของโปรแกรมสามารถทำงานร่วมกันได้อย่างดี จะนำ data structure และองค์ประกอบเหล่านั้นมาพัฒนาจริง โดย implement ในรายละเอียดและฟังก์ชั่นการทำงานต่าง ๆ อย่างครบถ้วน

จากขั้นตอนที่แบ่งออกเป็นสองขั้นใหญ่ ๆ ในขั้นต้นทำให้การพัฒนาโปรแกรมทำได้ง่าย ลดความเสี่ยงต่อการทำงานที่ไม่เข้ากันขององค์ประกอบต่าง ๆ ในระบบ เวลาส่วนใหญ่ในการสร้างโปรแกรมจะอยู่ในขั้นตอนที่หนึ่ง ซึ่งใช้เวลามากที่จะหา data structure และโครงสร้างของระบบที่เหมาะสม รวมทั้งการหาจุดที่เหมาะสมที่แสดงว่าได้ data structure และองค์ประกอบต่าง ๆ เหมาะสม สามารถทำงานสอดประสานกันได้อย่างมีประสิทธิภาพ เพราะว่าไม่สามารถวัดค่าความเหมาะสมต่าง ๆ เหล่านี้ออกเป็นตัวเลข หรือจำนวนใด ๆ ที่จะบ่งชี้ถึงความเหมาะสมนั้น

ขั้นตอนการทำงานทั้งสองขั้นต้องการใช้เครื่องมือช่วยในการพัฒนาที่แตกต่าง ตามวัตถุประสงค์ของแต่ละขั้นตอน ซึ่งจะกล่าวดังต่อไปนี้

1 ในขั้นแรก คือการสร้างโปรแกรมตัวอย่าง เครื่องมือที่เหมาะสมในการสร้างโปรแกรมตัวอย่างจะต้องสามารถสร้างโปรแกรมได้อย่างรวดเร็ว มี editor ที่มีความสามารถ ใช้งานง่ายทั้งในแง่ของการสร้างโปรแกรม และการ debug โปรแกรม เครื่องมือที่ใช้ในขั้นตอนนี้ได้แก่ Microsoft QuickC for Windows version 1.0 มีความเหมาะสมในการที่จะสร้างโปรแกรมต้นแบบที่มีขนาดเล็ก และมีการเปลี่ยนแปลงบ่อย ต้องการความเร็วในการสร้างและ debug โปรแกรม เนื่อง compiler นี้ถูกออกแบบมาเพื่อสร้างโปรแกมแอพพลิเคชั่นบน Windows เวอร์ชั่น 3.0 ดังนั้นจึงต้องมีการปรับแต่ง compiler นี้เล็กน้อยเพื่อให้สามารถสร้างโปรแกรมแอพพลิเคชั่นที่สามารถใช้ความสามารถของระบบ Windows เวอร์ชั่น 3.1 ได้ ซึ่งมีขั้นตอนดังนี้

1.1 ในการสร้างโปรแกรมที่รันบน *Windows* จะต้องประกอบด้วยไฟล์ที่จะต้องนำมา *link* เพิ่มกับไฟล์โปรแกรมอีก 2 ไฟล์คือ ไฟล์ *libw.lib* คือไฟล์ที่เก็บรายชื่อและตำแหน่งของ *export function* ที่ *Windows* มีให้โปรแกรมแอพพลิเคชั่นเรียกใช้ , ไฟล์ *xlibcew.lib* คือไฟล์ที่ใช้เก็บ *run-time function* ซึ่งเป็นฟังก์ชั่นมาตราฐานในภาษา C โดยที่ x แทนชื่อย่อของโมเดลของโปรแกรมที่สร้าง (*s* สำหรับ *small*, *m* สำหรับ *medium* เป็นต้น) ซึ่งทั้งสองไฟล์นี้มากับซอฟแวร์ชุด SDK (Software Development Kits) สำหรับสาเหตุที่ต้องใช้ไฟล์เหล่านี้มีดังนี้

-ไฟล์ libw.lib :- เนื่องจากระบบ Windows ประกอบด้วยจากไฟล์ที่เรียกว่า DLL (Dynamic Link Library) ซึ่งเป็นไฟล์ที่ใช้เก็บฟังก์ชั่นต่าง ๆ ที่โปรแกรมอื่นสามารถเรียกใช้ โดยที่ฟังก์ชั่นเหล่านั้นจะไม่ถูก link รวมเข้าไปในโปรแกรมที่เรียกใช้ฟังก์ชั่นนั้นตอนสร้างโปรแกรม แต่ฟังก์ชั่นเหล่านี้จะถูกเรียกใช้เมื่อต้องการเท่านั้น การที่โปรแกรมจะเรียกใช้ฟังก์ชั่นเหล่านี้จากไฟล์ DLL ได้อย่างถูกต้อง โปรแกรม link จะใส่ข้อมูลต่าง ๆ เหล่านี้ในโปรแกรมเมื่อตอนสร้างโปรแกรม ซึ่งข้อมูลเหล่านี้โปรแกรม link จะนำมาจากไฟล์ที่เรียกว่า import library ถูก

สร้างจากไฟล์ DLL ซึ่งเป็นไฟล์ที่เก็บข้อมูลต่าง ๆ ของฟังก์ชั่นที่มีอยู่ในไฟล์ DLL ซึ่งโปรแกรมอื่นสามารถเรียกใช้ได้ โดยโปรแกรมจะใช้ข้อมูลเหล่านี้ในการเรียกฟังก์ชั่นได้ถูกต้อง สำหรับ export ไฟล์ที่โปรแกรมแอพพลิเคชั่นสามารถเรียกใช้จากระบบ Windows ได้เก็บอยู่ในไฟล์ libw.lib

-ไฟล์ xlibcew.lib เป็น run-time function มาตราฐานของภาษา C เพื่อให้ฟังก์ชั่นมาตราฐานเหล่านี้สามารถทำงานได้ถูกต้องในระบบ Windows ฟังก์ชั่นเหล่านี้จะถูก link รวมในโปรแกรมที่เรียกใช้ ซึ่งแตกต่างจากฟังก์ชั่นที่เก็บในไฟล์ DLL ทำให้ขนาดของโปรแกรมมีขนาดใหญ่

1.2 ไฟล์ที่ต้องการใช้ในการสร้างโปรแกรมทั้งสองไฟล์นี้มีมากับโปรแกรม QuickC for Windows แล้วแต่เนื่องจากไฟล์เหล่านี้ใช้สำหรับโปรแกรมที่รันบน Windows เวอร์ชั่น 3.0 เพื่อให้โปรแกรมสามารถใช้คุณสมบัติที่เพิ่มมาของ Windows เวอร์ชั่น 3.1 โปรแกรม compiler ต้องใช้ไฟล์เหล่านี้ที่ใช้สำหรับเวอร์ชั่น 3.1 โดยการ set path ใหม่ของไฟล์เหล่านี้ในโปรแกรม QuckC ก่อน แสดงดังรูปที่ 1

เมื่อทำการ set path ใหม่แล้วก็สามารถใช้โปรแกรม QuckC for Windows สร้างโปรแกรมแอพพลิเคชั่นที่สามารถใช้คุณสมบัติใหม่ ๆ ของ Windows เวอร์ชั่น 3.1

2.ในขั้นตอนที่สอง หลังจากที่ได้ data structure และองค์ประกอบของโปรแกรมที่เหมาะสมแล้ว ในขั้นนี้เป็นการทำ implement จริงซึ่งต้องลงในรายละเอียดมาก โปรแกรมมีความซับซ้อนเพิ่มขึ้น เครื่องมือที่ใช้ในการพัฒนาโปรแกรมจำเป็นต้องมีความสามารถเพิ่มขึ้นตามความซับซ้อนของโปรแกรม ในขั้นตอนนี้แบ่งรายละเอียดออกเป็น 2 ขั้นตอนคือ

2.1 ขั้นพัฒนาโปรแกรม ในขั้นตอนนี้โปรแกรมมีความซับซ้อนเพิ่มมากขึ้น การหา bugs ในโปรแกรมทำได้ยากขึ้น ดังนั้นจึงต้องการเครื่องมือที่มีความสามารถในการ debug โปรแกรมเพิ่มขึ้น ซึ่งเครื่องมือที่ใช้ในการ debug โปรแกรมในขั้นตอนนี้ได้แก่โปรแกรม Code View for Windows เวอร์ชั่น 4.0 ซึ่งเป็นโปรแกรมที่มาพร้อมกับ package Microsoft C เวอร์ชั่น 7.0 โปรแกรม CVW มีความสามรถในการ debug มากเช่น สามารถ save ตำแหน่ง breakpoint, ตัวแปร static หรือ global ที่กำลังตรวจสอบค่า, สามารถกำหนดขอบเขตของพื้นที่หน่วยความจำที่จะทำการตรวจสอบเมื่อมีการเปลี่ยนค่าในพื้นที่หน่วยความจำนั้น เป็นต้น

ในขั้นตอนนี้โปรแกรมที่สร้างยังไม่ต้องการ code ที่ optimize แต่ต้องการโปรแกรมที่สร้างได้เร็ว ดังนั้นในขั้นตอนนี้ใช้โปรแกรม QuckC for Windows เป็นโปรแกรม compiler เหมือนเดิม แต่เพื่อให้ compiler สร้างโปรแกรมที่อยู่ใน format ที่โปรแกรม debugger (CVW) รู้จัก ต้องมีการเปลี่ยนแปลง option บางส่วนของโปรแกรม compiler

ใน option menu เลือก menu project จากนั้นเลือก link option set ค่าใน debug option เป็น CV 3.X format และก่อนที่จะทำการ debug โปรแกรมจะต้องรันโปรแกรม cvpack.exe ซึ่งเป็นโปรแกรมที่มากับชุด

2.2 ขั้นสุดท้าย หลังจากที่ทำการ *implement* โปรแกรมและตรวจสอบ *bug* ในโปรแกรมเรียบร้อยแล้ว โปรแกรมในขั้นตอนสุดท้ายจะนำไปทำการ *optimize* เพื่อให้ได้ *code* ที่ทำงานได้อย่างมีประสิทธิภาพ และมีขนาดไม่ใหญ่มาก สำหรับเครื่องมือที่ใช้ในการ *optimize code* ของโปรแกรมในขั้นสุดท้ายคือ *Microsoft C* เวอร์ชั่น 7.0

Hardware Independent

อุปกรณ์แต่ละชนิดหรือชนิดเดียวกัน(แต่ผู้ผลิตต่างกัน-รุ่นต่างกัน) ย่อมมีลักษณะการควบคุมการทำงานภายในและการส่งข้อมูลที่แตกต่างกัน โอเอสที่ดีจึงควรจะสามารถควบคุมอุปกรณ์เหล่านี้ให้ได้มากชิ้นมากยี่ห้อ เข้ากันได้กับหลายผู้ผลิต จึงเกิดแนวคิดที่จะพยายามรวมโปรแกรมที่ควบคุมอุปกรณ์จากหลายค่ายเข้ามาเป็นอันหนึ่งอันเดียวกันกับโอเอส โดยมีความหวังที่จะสร้างโอเอสแบบครอบจักรวาล แต่ปัญหาที่เกิดขึ้นคือขนาดของโอเอสใหญ่เกินไป เนื่องจากในตลาดคอมพิวเตอร์มีผู้ผลิตมากมายหลายสำนัก อุปกรณ์ของแต่ละสำนักยังแบ่งย่อยออกเป็นหลายรุ่น หลายเวอร์ชัน ถ้าจะรวบรวมโปรแกรมควบคุมอุปกรณ์จากหลายสำนัก หลายเวอร์ชันย่อมใหญ่โตมโหฬารมาก นอกจากนี้ยังเจอปัญหาการอัพเดตโอเอสให้ทันสมัยทุกครั้งที่มีอุปกรณ์ยี่ห้อใหม่หรือเวอร์ชันใหม่ออกมาดังนั้นแนวคิดนี้จึงต้องล้มเลิกไป

ต่อมาผู้พัฒนาโอเอสได้ออกแบบโดยแยกเอาโปรแกรมที่ควบคุมอุปกรณ์ออกจากตัวโอเอสและเรียกโปรแกรมที่ควบคุมอุปกรณ์ต่างๆนี้ว่า "Device Driver" การทำงานของไดรเวอร์จะแบ่งเป็น อินพุท(Keyboard, Mouse,)และ เอาท์พุท(Screen, Printer, ..) อินพุทไดรเวอร์จะควบคุมการรับข้อมูลแล้วส่งผลลัพธ์(ในรูปแบบมาตรฐานที่โอเอสกำหนด)ไปให้โอเอสจัดการต่อไป ส่วนเอาท์พุทไดรเวอร์จะรับข้อมูลจากโอเอส(ในรูปแบบมาตรฐานที่โอเอสกำหนด)แล้วนำมาแสดงผลออกทางอุปกรณ์ วิธีนี้จะไม่ทำให้ขนาดของโอเอสเปลี่ยนแปลงไปโอเอสไม่จำเป็นต้องรู้เลยว่าอุปกรณ์นั้นทำงานอย่างไรเพราะไดรเวอร์จะเป็นผู้ควบคุมดูแลแทนทั้งหมด โอเอสเพียงแต่รับ-ส่งข้อมูลกับไดรเวอร์เท่านั้นซึ่งเป็นลักษณะที่คล้ายคลึงกับ Object-Oriented Technology นอกจากนี้ ถ้าเกิดผู้ใช้ต้องการเปลี่ยนอุปกรณ์ เช่น เปลี่ยนจอภาพจาก EGA เป็น VGA ก็ง่ายดายนัก แค่นำไดรเวอร์ที่ควบคุมจอ EGA ออกไปแล้วนำไดรเวอร์ของจอ VGA มาแทนที่ก็เป็นอันเสร็จเรียบร้อยไม่ต้องอัพเดตโอเอสให้วุ่นวาย และนี่ก็คือที่มาของคำว่า "Hardware Independent" หรือ ความเป็นอิสระต่อฮาร์ดแวร์ นั่นเอง

เมื่อเรารัน Microsoft Windows ไม่ว่าจะเป็นโหมดใดก็ตามจะมีการติดตั้ง Device Driver ต่างๆในระบบเสมอ ขณะที่กดคีย์ครั้งหนึ่งจะมีรหัส Scan-code เกิดขึ้นตามตำแหน่งของคีย์ที่ถูกกด บริษัทผู้ผลิตคีย์บอร์ดจะเป็นผู้ออกแบบวางจรว่าตำแหน่งไหนจะเกิดรหัสที่มีค่าเท่าใด ถ้าเป็นคีย์บอร์ดที่คอมแพททิเบิลกับ IBM Keyboard ก็มักจะมีค่าสแกนโค้ดที่เหมือนกัน คีย์บอร์ดไดรเวอร์จะนำค่าสแกนโค้ดมาสร้างเป็นรหัสที่วินโดวส์เข้าใจ(คือรูป

บอร์ดที่ไม่คอมแพททิเบิลซึ่งอาจจะมีค่าสแกนโค้ดไม่เหมือนกัน มีการควบคุมที่ต่างกัน ก็ต้องขอไฟล์คีย์บอร์ดไดรเวอร์สำหรับวินโดวส์จากผู้ผลิตคีย์บอร์ดเพื่อทำหน้าที่ควบคุมคีย์บอร์ดและเปลี่ยนค่าสแกนโค้ด(ที่ไม่ตรงกับค่าสแกนโค้ดของ IBM Keyboard)ให้เป็น Virtual-key Code ที่ถูกต้องตามแบบฉบับของ Microsoft Windows จะเห็นว่าวินโดวส์ไม่รับรู้เลยว่าเรากำลังใช้คีย์บอร์ดยี่ห้ออะไรแต่เมื่อใดก็ตามที่คีย์บอร์ดถูกกดวินโดวส์ต้องได้รับ Virtual-key Code ที่ถูกต้องจากคีย์บอร์ดไดรเวอร์เสมอ

การทำงานของ Keyboard

การกดคีย์บอร์ดหนึ่งครั้งหมายถึงการกด-แล้วปล่อยซึ่งจะเกิดแมสเซจ 2 ตัวคือ WM_KEYDOWN ตามด้วย WM_KEYUP นอกจากนี้ยังมีการกดแบบเชิงประกอบเช่น Alt+K(กด Alt - กด K - ปล่อย K -ปล่อย Alt ตามลำดับ) วินโดวส์ได้ออกแบบไว้ว่าปุ่มที่ถูกกดหรือปล่อยในขณะที่ปุ่ม Alt กำลังถูกกดอยู่จะถือว่าเป็น System-key และจะสร้าง WM_SYSKEYDOWN แทน WM_KEYDOWN สร้าง WM_SYSKEYUP แทนWM_KEYUP แมสเซจที่กล่าวมาทั้งหมดนี้จะถูกบรรจุลงในซิสเต็มคิวเพื่อให้วินโดวส์แจกจ่ายให้แก่แอปพลิเคชันคิวต่อไป

วินโดวส์จะมีวิธีจ่ายแมสเซจที่เกิดจากการกดคีย์บอร์ดให้แก่แอปพลิเคชันอย่างไรในเมื่อระบบของวินโดวส์สามารถรันแอปพลิเคชันพร้อมๆกันได้หลายตัว แอปพลิเคชันตัวใดควรจะได้รับแมสเซจ แต่ละแอปพลิเคชันจะมีแอปพลิเคชันคิวอันเดียวเท่านั้น ไม่ว่าแอปพลิเคชันนั้นจะมีการสร้างวินโดว์กี่บานก็ตามทุกวินโดว์ที่สร้างขึ้นจะใช้แอปพลิเคชันคิวร่วมกัน สมมติว่ากำลังรันแอปพลิเคชันพร้อมกันสามตัวก็จะมีแอปพลิเคชันคิวในระบบ 3 คิว แต่ถ้าลองสังเกตให้ดีจะพบว่ามีเพียงแอปพลิเคชันเดียวเท่านั้นที่กำลังแอคทีฟอยู่โดย Title-bar จะมีแถบสว่างซึ่งปกติจะเป็นสี Dark Blue และในทำนองเดียวกันแอปพลิเคชันที่กำลังแอคทีฟอยู่นั้นจะมีเพียงวินโดว์เดียวที่มีโฟกัส(Focus)อยู่ วินโดว์ที่มีโฟกัสหมายถึงวินโดว์ที่จะได้รับแมสเซจที่เกิดจากการกดคีย์บอร์ด (โฟกัสคือ วินโดว์ที่กำลังแอคทีฟ ใน แอปพลิเคชันที่แอคทีฟอยู่) วินโดวส์จะจ่ายแมสเซจที่เกิดจากการกดคีย์บอร์ด(ที่อยู่ในซิสเต็มคิว)ไปที่คิวของแอปพลิเคชันที่กำลังแอคทีฟ หลังจากนั้นแมสเซจลูปจะนำมาแพ็คให้อยู่ในรูปของ Structureของ MSG แล้วส่งไปให้วินโดว์ฟังก์ชันของวินโดว์ที่มีโฟกัส สิ่งที่ส่งผ่านไปทางอาร์กิวเมนต์ของวินโดว์ฟังก์ชันก็คือ ค่าแฮนเดิลของวินโดว์ที่มีโฟกัสอยู่, แมสเซจ(WM_KEYDOWN or WM_SYSKEYDOWN or WM_KEYUP or WM_SYSKEYUP), wParam และ lParam ข้อมูลใน wParam และ lParam ของแมสเซจทั้ง 4 ตัวที่กล่าวถึงอยู่นี้จะมีความหมายเหมือนกันคือ ค่าใน wParam(16 บิต) จะเป็น Virtual-keyCode ที่ได้มาจากคีย์บอร์ดไดรเวอร์ ส่วนค่าใน lParam(32 บิต) จะแบ่งเป็น LOWORD(16 บิตล่าง) กับ HIWORD(16 บิตบน) โดยมีความหมายดังนี้

- Repeat Count ปกติค่านี้จะเป็น 1 แต่ถ้ากดคีย์ค้างไว้ค่านี้จะมากกว่า 1 ประโยชน์มีไว้เพื่อป้องกันการเกิด Overflow
- OEM Scan Code มี 8 บิต(บิตที่ 16 - 23) เก็บค่า Scan-code ของคีย์บอร์ด
- Extended Flag จะเป็น 1 ถ้าคีย์ที่ถูกกดเป็นคีย์เพิ่มเติมพิเศษ เช่น ลูกศร, Home,End, Ctrl(right),
- Context Code จะเป็น 1 ถ้าคีย์ Alt กำลังถูกกดอยู่

- OEM Scan Code มี 8 บิต(บิตที่ 16 - 23) เก็บค่า Scan-code ของคีย์บอร์ด

- Extended Flag จะเป็น 1 ถ้าคีย์ที่ถูกกดเป็นคีย์เพิ่มเติมพิเศษ เช่น ลูกศร, Home,End, Ctrl(right),

- Context Code จะเป็น 1 ถ้าคีย์ Alt กำลังถูกกดอยู่

- Previous State จะเป็น 1 ถ้าก่อนหน้านี้ คีย์ชนิดนี้กำลังถูกกดอยู่ มักใช้ประโยชน์กับปุ่มที่มี 2 ตำแหน่งเช่น ปุ่ม Ctrl(มี 2 ปุ่มบนคีย์บอร์ด)

- Transition State จะเป็น 1 เมื่อกำลังปล่อยคีย์

ถ้าลองทบทวนการทำงานของคีย์บอร์ดจะพบว่าการสร้างเวิร์ดโปรเซสเซอร์นั้นจะมีปัญหาเกิดขึ้นมากมายเพราะค่าที่ได้จากคีย์บอร์ดไดรเวอร์ไม่ใช่รหัส ASCII แต่เป็น Virtual-key Code เราจะต้องตรวจสอบเรื่องอักขระตัวใหญ่ - ตัวเล็ก อักขระสัญลักษณ์(เช่น %, @, ..) เอาเอง สิ่งที่เราจะต้องเช็คมีดังนี้

1 ขณะกด A - Z ถ้า Caps Lock ON ให้พิมพ์ตัวใหญ่ แต่ถ้าคีย์ Shift ถูกกดอยู่ให้พิมพ์ตัวเล็ก

2. ขณะกด A - Z ถ้า Caps Lock OFF ให้พิมพ์ตัวเล็ก แต่ถ้าคีย์ Shift ถูกกดอยู่ให้พิมพ์ตัวใหญ่

3 ขณะกด 0 - 9 ไม่ว่า Caps Lock ON หรือ OFF ให้พิมพ์ 0 - 9 ถ้าคีย์ Shift ถูกกดอยู่ให้พิมพ์สัญลักษณ์เช่น %, @,

4. นอกจากนี้ยังมีอักขระอื่นอีกที่มีปัญหาเช่น '/', '?' จะตรวจสอบได้อย่างไร(สังเกตปุ่มบนคีย์บอร์ด)

ถ้าเราได้รหัส ASCII จากคีย์บอร์ดไดรเวอร์อุปสรรคพวกนี้ก็จะหมดไป ใน API ของวินโดวส์ได้มีฟังก์ชัน TranslateMessage ซึ่งเรามักจะบรรจุไว้ในแมสเซสลูป โดยอยู่เหนือฟังก์ชัน DispatchMessage การทำงานของฟังก์ชันนี้จะเป็นดังนี้ สมมติว่ามีการกดคีย์บอร์ด แมสเซจ WM_KEYDOWN และ WM_KEYUP จึงเกิดขึ้น การทำงานจะเริ่มจากฟังก์ชัน GetMessage ดึงแมสเซจจากหัวคิวมาบรรจุลงใน Structure ของ MSG ต่อมาฟังก์ชัน TranslateMessage จะเช็คว่าแมสเซจใน MSG เป็น WM_KEYDOWN หรือ WM_SYSKEYDOWN หรือเปล่า ถ้าไม่ใช่จะจบการทำงานทันที ถ้าพบก็จะส่งค่าใน wParam(Virtual-key Code) ไปให้คีย์บอร์ดไดรเวอร์เพื่อนำค่านี้ไปแปลงเป็นรหัส ASCII แล้วส่งกลับมา (ถ้าแปลงเป็นรหัสแอสกีไม่ได้เช่น คีย์ F1 จะจบการทำงานทันที) หลังจากนั้นฟังก์ชัน TranslateMessage จะสร้างแมสเซจ WM_CHAR(หรือ WM_SYSCHAR) แล้วนำไปบรรจุไว้ที่หัวคิวของ Application queue โดยเลื่อนแมสเซจอื่น ในคิวร่นถอยหลังไปจึงเป็นอันเสร็จสิ้นการทำงานของฟังก์ชัน TranslateMessage จากที่กล่าวมานี้จึงขอสรุปและเพิ่มเติมเป็นหัวข้อย่อยดังนี้

- ฟังก์ชัน TranslateMessage จะทำงานเมื่อพบว่าแมสเซจใน MSG เป็น WM_KEYDOWN หรือ WM_SYSKEYDOWN เท่านั้น

- ในกรณีที่ wParam(Virtual-key Code) ของ WM_KEYDOWN(หรือ WM_SYSKEYDOWN) สามารถแปลงเป็นรหัสแอสกีได้จะมีการสร้าง WM_CHAR หรือ WM_SYSCHAR โดยขึ้นอยู่กับแมสเซจใน MSG ว่าเป็น WM_KEYDOWN หรือ WM_SYSKEYDOWN

- แมสเซจที่สร้างขึ้นมานี้เราเรียกว่า "Character Message" และถูกบรรจุที่หัวของแอปพลิเคชัน ดังนั้น Character Message จึงอยู่ตรงกลางระหว่าง WM_KEYDOWN(หรือ WM_SYSKEYDOWN) กับ WM_KEYUP(หรือ WM_SYSKEYUP)

- ค่าใน wParam ของ WM_CHAR(หรือ WM_SYSCHAR) ก็คือรหัส ASCII นั่นเอง

- ค่าใน lParam ของ WM_CHAR(หรือ WM_SYSCHAR) จะ copy มาจากค่าใน lParam ของ WM_KEYDOWN(หรือ WM_SYSKEYDOWN)

Resource

รีซอร์สหรือเครื่องประดับสำหรับแอปพลิเคชันประกอบไปด้วย dialog boxes, menus, strings, accelerators, icons, fonts, bitmaps และ cursors ในบรรดารีซอร์สทั้งหมดนี้เราสามารถแบ่งออกเป็น ASCII Resource กับ Non-ASCII Resource

ในการสร้างแอสกีรีซอร์สนั้นเราต้องออกแบบรีซอร์สที่จะสร้างในรูปแบบที่วินโดวส์กำหนดไว้โดยพิมพ์เป็น Text file ธรรมดาลงในไฟล์ที่มีนามสกุลเป็น .RC ซึ่งต่อไปเราจะเรียกไฟล์นี้ว่า "resource file" สำหรับนอน-แอสกีรีซอร์สซึ่งได้แก่ icons, fonts, bitmaps และ cursors ต้องสร้างขึ้นด้วย Tools Program ต่างๆเช่น Image Editor หรือ Font Editor ในชุด SDK 3.1 เมื่อสร้างเสร็จแล้วจึงนำชื่อไฟล์ที่สร้างขึ้นไปใส่ลงในรีซอร์สไฟล์(RC)

เมื่อสร้างและออกแบบรีซอร์สไฟล์เรียบร้อยแล้วก็ลงมือประกอบรีซอร์สเข้าไปรวมกับแอปพลิเคชันโดยเริ่มจากการใช้ Resource Compiler(RC.EXE) เพื่อคอมไพลรีซอร์สไฟล์(.RC)ให้เป็นไบนารีรีซอร์สไฟล์(.RES) เมื่อได้ไบนารีรีซอร์สไฟล์แล้วให้ใช้รีซอร์สคอมไพล์เลอร์อีกครั้งหนึ่งเพื่อรวมไบนารีรีซอร์สไฟล์(RES)เข้ากับเอกซีคิวไฟล์(EXE)การที่ Resource ถูกสร้างและคอมไพล์แยกต่างหากก่อนที่จะมารวมกับส่วนของโค้ดนับว่าเป็นข้อดี กล่าวคือ โปรแกรมเมอร์ไม่ต้องรับภาระการออกแบบด้าน Graphic ให้สวยงามรวมทั้งสามารถเปลี่ยนแปลงแก้ไขได้ง่ายอีกด้วย

Multiple Document Interface

MDI เป็นยูสเซอร์อินเทอร์เฟสอย่างหนึ่งที่ประกอบด้วย วินโดว์หลัก กับ วินโดว์เอกสาร ซึ่งเราสามารถเปิดวินโดว์เอกสารได้มากกว่าหนึ่งเอกสารพร้อมๆกัน โครงสร้างของ Multiple Document Interface ประกอบด้วย Main application window ที่ชื่อ "MDI Frame window" บน Client Area ของ Frame window เราจะสร้าง Child window ที่ชื่อว่า "MDI Client window" เพื่อใช้เป็นที่บรรจุวินโดว์เอกสาร(หน้าต่างเอกสาร)ลงไป ทุกครั้งที่มีการเปิดหน้าต่างเอกสาร ก็จะมีการสร้าง Child window ที่ชื่อ"MDI Child window" ขึ้นมาซึ่งเป็น Child window ของ MDI Client window อีกที ดังแสดงตามรูป

1) สามารถ Minimize Document เพื่อซ่อนหน้าต่างเอกสารโดยแสดงเป็นภาพไอคอนเล็กๆบนส่วนของ Client window

2) MDI จะมี System keyboard accelerator เพื่อควบคุมการทำงาน เช่น

- Ctrl + F6 ใช้เปลี่ยนไปยังหน้าต่างเอกสารถัดไป
- Ctrl + F4 ใช้ปิดหน้าต่างเอกสาร
- Alt + '-' ใช้เปิด System Menu ของหน้าต่างเอกสารที่กำลัง Active อยู่

3) หน้าต่างเอกสารจะไม่มี Menu bar เมนูของ Main application window จะทำหน้าที่เป็นเมนูของหน้าต่างเอกสารที่กำลัง Active อยู่ เราสามารถเปลี่ยนรูปแบบของเมนูในแต่ละหน้าต่างเอกสารก็ได้

4) บนเมนูควรจะมี Window pop-up menu เพื่อบรรจุเมนูไอเท็ม Tile, Cascade, Icon rearrange รวมทั้งรายชื่อของหน้าต่างเอกสารที่กำลังเปิดอยู่(คั่นด้วย separator line)

- Tile จะ rearrange เฉพาะหน้าต่างเอกสารที่ไม่ได้ Minimize โดยหน้าต่างเอกสารที่กำลัง Active จะอยู่ตำแหน่งบน-ซ้ายของ Client window
- Cascade จะ rearrange เฉพาะหน้าต่างเอกสารที่ไม่ได้ Minimize เช่นเดียวกับ Tile แต่จะ Display เป็นลักษณะของ Stack ที่ซ้อนทับกันโดยหน้าต่างเอกสารที่กำลัง Active จะอยู่ตำแหน่งบนสุดของ Stack
- Icon rearrange จะ rearrange เฉพาะหน้าต่างเอกสารที่กำลัง Minimize ให้จัดเรียงที่ด้านล่าง-ซ้ายของClient window

5) เมื่อผู้ใช้ Maximize หน้าต่างเอกสาร Title bar ของหน้าต่างเอกสารจะไปปรากฏบน Title bar ของ Main application window นอกจากนี้ยังนำ System Menu ของหน้าต่างเอกสารไปแทรกที่ด้านซ้ายสุดของ Menu bar และแทรก Restore Button ที่ขวาสุดของ Menu bar

6) หน้าต่างเอกสารที่กำลัง Active อยู่จะมี Highlight ปรากฏบน Title bar ของหน้าต่างเอกสารนั้น

คุณสมบัติของโปรแกรม

1. สามารถป้อนข้อความแบบ Multiple proportional font คือมีได้หลายฟอนท์ แต่ละตัวอักษรมีความกว้างไม่เท่ากัน

2 Auto break line โปรแกรมสามารถตัดคำที่เกินขอบที่ตั้งไว้โดยอัตโนมัติ

3 สามารถสร้างเอกสารที่มีหลายพารากราฟ โดยที่แต่ละพารากราฟมี margin ไม่เท่ากันได้

2. Auto break line โปรแกรมสามารถตัดคำที่เกินขอบที่ตั้งไว้โดยอัตโนมัติ

3. สามารถสร้างเอกสารที่มีหลายพารากราฟ โดยที่แต่ละพารากราฟมี margin ไม่เท่ากันได้

4. โปรแกรมสามารถ scroll หน้าต่างเพื่อทำการดูข้อความส่วนที่เกินหน้าต่างได้ทั้งในแนวดิ่ง และแนวนอน (Horizontal and Vertical scroll)

5. ขนาดความสูงในแต่ละบรรทัดจะมีขนาดที่ไม่เท่ากัน ขึ้นอยู่กับความสูงของตัวอักษรที่สูงที่สุดในบรรทัดนั้น โปรแกรมจะจัดตำแหน่งของบรรทัดโดยอัตโนมัติเมื่อมีการเปลี่ยนขนาดของตัวอักษรในบรรทัด

6. ขนาดของ caret จะเปลี่ยนไปตามความสูงของแต่ละบรรทัด

7 สามารถแสดงภาษาไทยได้ โดยใช้ฟอนท์ BrowalliaUPC ซึ่งเป็น True Type ฟอนท์ใน Windows Thai edition

8. สามารถรันได้ทุก Platform ที่ระบบวินโดว์สสามารถรันได้ โดยโปรแกรมต้องการระบบที่มีหน่วยความจำอย่างน้อย 640 KB และพื้นที่ว่างบนดิสก์สำหรับทำ swap file

9 ขนาดของข้อมูลมีขนาดใหญ่ได้ไม่จำกัด ขึ้นอยู่กับพื้นที่ที่เหลือในดิสก์ โปรแกรมต้องการขนาดพื้นที่หน่วยความจำคงที่

ข้อจำกัดของโปรแกรม

1 ในแต่ละบรรทัดสามารถป้อนตัวอักษรได้ไม่เกิน 512 ตัวอักษร

2. ในแต่ละบรรทัดสามารถเปลี่ยนฟอนท์ได้มากที่สุด 16 ฟอนท์

3. โปรแกรมต้องการพื้นที่หน่วยความจำ 256 KB สำหรับเก็บข้อมูล ข้อมูลส่วนที่เกินจะถูก swap ลง disk

การใช้งานโปรแกรม

1 <u>การเปลี่ยนฟอนท์</u>

เลือก Menu font ในเมนู Character

2 <u>การเปลี่ยน indent</u>

เลื่อน caret ไปยังพารากราฟที่ต้องการเปลี่ยน เลือก dialog จากเมนู แล้วตั้งค่า Indent ใหม่โดยสามารถตั้งได้ไม่เกิน 6 นิ้ว

3. <u>การขึ้นพารากราฟใหม่</u>

ถ้าเรากด Enter จะถือว่าเป็นการขึ้นพารากราฟใหม่ โดยจะมี Indent เป็นค่า default = 3

DIALOG.C

```
#include <windows.h>
#include "cons.h"
#include "exfunc.h"
#include <stdlib.h>
#include <string.h>

/* get string from dialog and convert to integer */
/* return : nonzero for change otherwise zero */
int GetRightIndent(HWND    hdlg)
{
        char    i;
        int     temp;
        char    ans[8], buffer[3];

        GetDlgItemText(hdlg, 102, ans, 8);

        i = 0;
        while (ans[i] != '.' && ans[i] && i < 2)
        {
                buffer[i] = ans[i];
                i++;
        }

        buffer[i++] = 0;
        if (atoi(buffer) > MAXPAPERWIDTH)
        {
                MessageBox(hdlg,
                        "Right Indent exceed avaliable values",
                        NULL,   MB_OK);
                return 0;
        }
```

```
        temp = atoi(buffer)*logicalinch;

        if (i < (char)strlen(ans))
        {
                strcpy(ans, &ans[i]);
                temp += (atoi(ans)*logicalinch)/100;
        }

        if (temp == descript.rightindent)
                return 0;
        else
        {
                descript.rightindent = temp;
                return 1;
        }
}
```

DISPLAY.C

```
#include <windows.h>
#include "const.h"
#include "exfunc.h"
#include <memory.h>

/* export variable */
int         begincol;  /* for scroll horizontal */
int         beginline; /* for scroll vertical */
int         clientwidth;
int         clientheight;

/* description :- set positon of caret    */
/* input :- hwnd -> handle of window */
/* return :- none.   */

void    GetCaret(HWND    hwnd)
{
        CreateCaret(hwnd, 0, 0, dispTable[currentline].lineBase);
        SetCaretPos (xPos, yPos);
        ShowCaret(hwnd);
}

/* description :- update client dimension when size of*/
/*      window changed.              */
/* input :- hwnd -> handle of window */
/*         dimension -> width in loword          */
/*         height in hiword.             */
/* return   none    */

void ClientSizeChange(HWND          hwnd,
                      LPARAM        dimension)
```

```
{
        int     xdiff, ydiff;
        int     i, temp;

        xdiff = LOWORD(dimension)-clientwidth;
        ydiff = HIWORD(dimension)-clientheight;

        clientwidth = LOWORD(dimension);
        clientheight = HIWORD(dimension);

        if (xPos > clientwidth)
        {
                begincol -= xPos-clientwidth+XSCROLLUNIT;
                xPos = clientwidth-XSCROLLUNIT;

                InvalidateRect(hwnd, NULL, TRUE);
                SetCaretPos(xPos, yPos);
        }
        else
        {
                if (xdiff > 0)
                {
                        /* width is expanded */
                        if (begincol+xdiff < descript.leftindent)
                        {
                                begincol += xdiff;
                                xPos += xdiff;
                        }
                        else
                        {
                                xPos += descript.leftindent-begincol;
                                begincol = descript.leftindent;
                        }
```

```
                    InvalidateRect(hwnd, NULL, TRUE);
                    SetCaretPos(xPos, yPos);
            }
        }

        i = beginline;
        temp = 0;
        while (temp+dispTable[i].lineHeight < clientheight &&
                    dispTable[i].paraIndex)
            temp += dispTable[i++].lineHeight;

        if (yPos > clientheight)
        {
            i = currentline - i;
            beginline += i+1,

            i = beginline;
            yPos = 0;
            while (i < currentline)
                yPos += dispTable[i++].lineHeight;

            InvalidateRect(hwnd, NULL, TRUE);
            SetCaretPos(xPos, yPos);
        }
        else
        {
            if (ydiff > 0)
            {
                /* height is expanded */
                if (i >= beginline)
                    beginline = 0;
                else
```

```
                beginline -= i+1;
                i = beginline;
                yPos = 0;
                while (i < currentline)
                yPos += dispTable[i++].lineHeight;
            InvalidateRect(hwnd, NULL, TRUE);
            SetCaretPos(xPos, yPos);
        }
}

/* description :- update invalid rectangle */
/* input :- hwnd -> handle of window. */
/* return . none */

void UpdateClient(HWND    hwnd)
{
        PAINTSTRUCT         ps;
        int                 height=0;
        int                 line, start, temp, size, i;
        HDC                 hdc;
        DWORD               ans;
        BYTE                row;
        BYTE                FAR *data;
        HFONT               hold;
        struct script_tag FAR *scriptptr;

    hdc = BeginPaint(hwnd, &ps);
        line = beginline;

        while (height+dispTable[line].lineHeight < ps.rcPaint.top
                && dispTable[line].paraIndex)
```

```
            height += dispTable[line++].lineHeight;


        start = dispTable[line].lineOffset;


        /* find actual offset in same block as currentline */
        if (dispTable[line].blockindex == dispTable[currentline].blockindex)
                if (start > dispTable[currentline].lineOffset)
                {
                start = dispTable[currentline].lineOffset+lstrlen(linebuffer.data);
                temp = currentline+1;
                while (temp < line)
                        start += dispTable[temp++].sizeInLine;
                }


        scriptptr = textptr[dispTable[line].blockindex].script;
        ans = FindScript(scriptptr, start);
        row = (BYTE)HIWORD(ans);
        hold = SelectObject(hdc, FontHandle(scriptptr[row].fontindex));


        /* loop show line in updated region */
        temp = scriptptr[row].sizeGroup-LOWORD(ans);
        while (height < ps.rcPaint.bottom && dispTable[line].paraIndex)
{
                MoveTo(hdc, begincol, height+dispTable[line].lineBase);
                start = 0;


                if (line == currentline)
                {
                        data=linebuffer.data;
                        size = lstrlen(linebuffer.data);
                }
                else
                {
```

```
            data=&textptr[dispTable[line].blockindex]
                        .textData[dispTable[line].lineOffset],
            size = dispTable[line].sizeInLine;
        }

        /* loop show multiple fonts in one line */
        while (1)
        {
          if (start+temp < size)
                ExtTextOut(hdc, 0, 0, 0, NULL, &data[start], temp, NULL);
          else
          {
                if (start+temp == size)
                {
                        ExtTextOut(hdc, 0, 0, 0, NULL, &data[start], temp, NULL);
                        i = scriptptr[++row].fontIndex,
                        if (i)
                        {
                                SelectObject(hdc, FontHandle(i));
                                temp = scriptptr[row].sizeGroup;
                        }
                }
                else
                {
                        ExtTextOut(hdc, 0, 0, 0, NULL, &data[start], size-start, NULL);
                        temp -= size-start;
                }
                break;
        }

        start += temp,
        temp = scriptptr[++row].sizeGroup,
        SelectObject(hdc, FontHandle(scriptptr[row].fontIndex)),
```

```
            }
            if ((dispTable[line].blockindex != dispTable[line+1].blockindex)
                    && dispTable[line+1].paraIndex)
            {
              while (!paraTable[dispTable[line+1].paraIndex].blockindex
                    && dispTable[line+1].paraIndex)
                    height += dispTable[line++].lineHeight;

              if (dispTable[line+1].paraIndex)
              {
                    row = 0;
                    start = 0;
                    scriptptr = textptr[dispTable[line+1].blockindex].script;
                    temp = scriptptr->sizeGroup;
                    SelectObject(hdc, FontHandle[scriptptr[row].fontIndex]);
              }
              else
                    break;
              }
              height += dispTable[line++].lineHeight;
        }

        SelectObject(hdc, hold);

    EndPaint(hwnd, &ps);
}

/* description :- covert offset in bytes into row and */
/*          offset in row in script table              */
/* input :- ptr -> pointer to script table */
/*offset -> offset in bytes in script table */
/* return :- row in hiword and offset in loword  */
```

```
DWORD FindScript(struct script_tag FAR        *ptr,  int offset)
{
        int       count=0, ans;
        BYTE      i=0;

        while (1)
        {
                if (count+ptr[i].sizeGroup < offset)
                {
                        count += ptr[i].sizeGroup;
                        i++;
                }
                else
                {
                        if (count+ptr[i].sizeGroup == offset)
                        {
                                if (ptr[i+1].fontindex)
                                {
                                        i++;
                                        ans = 0;
                                }
                                else
                                        ans = ptr[i].sizeGroup;
                        }
                        else
                                ans = offset-count;

                        break;
                }
        }
        return MAKELONG(ans, i);
}
```

```
/* description :- calculate width of multiple fonts */
/*          string.                 */
/* input :- data -> pointer to string*/
/*          size -> size of string          */
/*offset -> offset of string in script table*/
/*script -> pointer to script table of input*/
/*          string.                 */
/* return   width of string    */

int WhatTextExtent(BYTE     FAR *data, int     size, int          offset,     struct script_tag FAR *script)
{
        int         count, width, temp;
        BYTE        i;
        DWORD ans;
        HFONT       hold;

        ans = FindScript(script, offset);
        i = (BYTE)HIWORD(ans);
        width = script[i].sizeGroup-LOWORD(ans);
        count = 0;
        temp = 0;
        hold = SelectObject(hdc, FontHandle(script[i].fontindex));

        while (1)
        {
                if (count+width >= size)
                {
                        temp += LOWORD(GetTextExtent(hdc, &data[count], size-count));
                        break;
                }
                else
                {
                        temp += LOWORD(GetTextExtent(hdc, &data[count], width));
```

```
                count += width;
                width = script[++i].sizeGroup;
            }

            SelectObject(hdc, FontHandle(script[i].fontIndex));
        }

        SelectObject(hdc, hold);
        return  temp;
}

/* description :- break exceed word to next line.*/
/* input .- hwnd -> handle of window  */
/*        line -> broken line          */
/*mode -> 0 for brak until next line's width*/
/*          not exceed margin.         */
/*  1 for break until end of paragraph*/
/* return .- none   */

void BreakLine(HWND     hwnd,int        line,int        mode)
{
        int     size, offscript, temp, pos, i, count;
        int     bottom;
        TEXTBLOCK FAR   *ptr;
        BYTE    FAR     *data;
        RECT            rc,
        DISPTABLE FAR   *table;

        GetClientRect(hwnd, &rc);
        count = beginline,
        rc.top = 0,
```

```
while (count < line)
        rc.top += dispTable[count++].lineHeight;

rc.bottom = rc.top;
bottom = rc.bottom+dispTable[++count].lineHeight;;
do
{
        table = &dispTable[line];
        ptr = &textptr[table->blockindex];

        /* must check if break line is in text block */
        if (line+1 ==       MAXWINDOWLINE)
                line = ScrollDispTable[1];

        if (line==currentline)
        {
                size = lstrlen(linebuffer.data);
                data = linebuffer.data;
        }
        else
        {
                size = table->sizeInLine;
                data = &ptr->textData[table->lineOffset];
        }

        /* find word to break line */
        offscript = table->lineOffset;
        if (dispTable[currentline].blockindex == table->blockindex)
                if (dispTable[currentline].lineOffset < table->lineOffset)
                        offscript = table->lineOffset
                        + lstrlen(linebuffer.data)
                        -(dispTable[currentline+1].lineOffset
                        -dispTable[currentline].lineOffset),
```

```
temp = size;
do {
        pos = WordBreak(1, data, temp);
        count = WhatTextExtent(data, pos, offscript, ptr->script);
        temp = pos;
    } while (count > descript.rightindent);

table->lineWidth = count;
data = &data[pos];
size -= pos; /* size of broken string */

/* move data */
if (table->paraIndex != (table+1)->paraIndex)
{
        InsertNewLine(line, table->paraIndex);
        rc.bottom = clientheight;
}

ptr = &textptr[(table+1)->blockindex];
if ((table+1)->blockindex == table->blockindex)
{
    /* data in same block */
    if (line == currentline)
    {
        /* from buffer to block */
        (table+1)->sizeInLine += size;
        if ((table+1)->lineOffset>=table->lineOffset+size)
        {
            /* have enough space between line and line+1 */
            count = (table+1)->lineOffset-size,
            MemoryMove(&ptr->textData[count],
                                    data,
```

```
                                                    size),
                    (table+1)->lineOffset = count;
                }
                else
                {
                    /* must move old data */
                    temp = (table+1)->lineOffset;
                    count= temp-table->lineOffset;
                    count = size-count;
                    if (ptr->size+count <= BLOCKSIZE)
                    {
                        MemoryMove(&ptr->textData[temp+count],
                                   &ptr->textData[temp],
                                   ptr->size-temp);
                        ptr->size += count;

                        MemoryMove(&ptr->textData[table->lineOffset],
                                   data,
                                   size);
                        (table+1)->lineOffset=table->lineOffset;

                        /* update offset in same block */
                        temp = 2;
                        while (table->blockindex == (table+temp)->blockindex
                               && (table+temp)->blockindex)
                            (table+temp++)->lineOffset += count;
                    }
                    else
                    {
                        i = (table+1)->lineOffset;

                        /* update lineoffset */
                        pos = 1;
```

```
temp = 0;
while (table->blockindex == (table+pos)->blockindex)
{
        (table+pos)->lineOffset = temp;
        temp += (table+pos++)->sizeInLine;
}

count = ptr->size-i+size;
ptr->size = i;
if (textptr[(table+pos)->blockindex].size+count <=
        BLOCKSIZE
        && (table+pos)->paraIndex == table-
>paraIndex)
{
        /* following block have enough data */
        (table+1)->blockindex = (table+pos)-
>blockindex;
        ptr = &textptr[(table+1)->blockindex];
        MemoryMove(&ptr->textData[count],
                ptr-
>textData,
                ptr->size);

        /* update lineoffset */
        temp = pos;
        while ((table+pos)->blockindex ==
(table+temp)->blockindex)
                (table+temp++)->lineOffset +=
count;

        ptr->size += count;
}
else
```

```
                                        {
                                            /* allocate new block */
                                            (table+1)->blockindex =
NewTextBlock(table->blockindex,
                                                    table->paraIndex);
                                            ptr = &textptr[(table+1)->blockindex];
                                            ptr->size = count;
                                        }
                                        MemoryMove(&ptr->textData[size],
                                                &textptr[table-
>blockindex].textData[i],
                                                count-size);
                                        MemoryMove(ptr->textData,
                                                data,
                                                size);
                                        MoveScript(ptr->script,
                                                0,
                                                textptr[table-
>blockindex].script,
                                                table-
>lineOffset+lstrlen(linebuffer.data)-size,
                                                count),
                                        while (pos > 1)
                                            (table+(--pos))->blockindex = (table+1)-
>blockindex,
                                    }
                                }
                                data[0] = 0,
                            }
                            else
```

```
            {
                /* from same block don't move data */
                (table+1)->lineOffset -= size;
                table->sizeInLine -= size;
                (table+1)->sizeInLine += size;
            }
        }
        else
        {
            /* data in another block line+1 is first line*/
            if (ptr->size+size <= BLOCKSIZE)
            {
                /* can move to current block */
                MemoryMove(&ptr->textData[size],
                                    ptr->textData,
                                    ptr->size);
                ptr->size += size;

                MemoryMove(ptr->textData,
                                    data,
                                    size);
                MoveScript(ptr->script,
                                    0,
                                    textptr[table->blockindex].script,
                                    offscript+pos,
                                    size),

                /* if (line != currentline)
                        textptr[table->blockindex] size -= size; */

                /* update offset in same block */
                temp=2,
                while ((table+temp)->blockindex == (table+1)->blockindex)
```

```
                (table+temp++)->lineOffset += size;
        }
        else
        {
            /* must allocate new block */
            (table+1)->blockindex = NewTextBlock(table->blockindex,
                                                 table->paraIndex),
            temp = (table+1)->sizeInLine;

            MemoryMove(&textptr[(table+1)->blockindex].textData[size],
                                                 ptr->textData,
                                                 temp);
            MoveScript(textptr[(table+1)->blockindex].script,
                                                 0,
                                                 ptr->script,
                                                 0,
                                                 temp);
            MemoryMove(ptr->textData,
                                                 &ptr->textData[temp],
                                                 ptr->size-temp);
            ptr->size -= temp;

            i = 2;
            while ((table+2)->blockindex == (table+i)->blockindex)
                (table+i++)->lineOffset -= temp;

            ptr = &textptr[(table+1)->blockindex],
            ptr->size = temp+size;

            MemoryMove(ptr->textData,
                                                 data,
                                                 size),
```

```
                MoveScript(ptr->script,
                                                0,
                                                textptr[table->blockindex].script,
                                                offscript+pos,
                                                size);

                        (table+1)->lineOffset = 0;
                }

                table->sizeInLine -= size;
                (table+1)->sizeInLine += size;
                if (line == currentline)
                        linebuffer.data[lstrlen(linebuffer.data)-size] = 0;
                else
                        textptr[table->blockindex].size -= size;
        }

/* update line width of line and line+1 */
        if (line != currentline)
        {
        ptr = &textptr[table->blockindex];
        offscript = table->lineOffset;
        if (dispTable[currentline].blockindex == table->blockindex)
        {
                        temp = dispTable[currentline+1].lineOffset
                                        - dispTable[currentline].lineOffset,
                        temp = lstrlen(linebuffer.data)-temp;
                        offscript += temp,
        }
        table->lineWidth = WhatTextExtent(&ptr->textData[table->lineOffset],

                table->sizeInLine,
```

```
                offscript,

                ptr->script);

                if (UpdateLineHeight(line, offscript))
                        rc.bottom = clientheight;
        }
        else
        {
                if (UpdateLineHeight(line, table->lineOffset))
                        rc.bottom = clientheight;

        if (dispTable[currentline].blockindex == (table+1)->blockindex)
        {
                temp = dispTable[currentline+1].lineOffset
                                - dispTable[currentline].lineOffset;
                temp = lstrlen(linebuffer.data)-temp;
        }
        }
        ptr = &textptr[(table+1)->blockindex];
        offscript = (table+1)->lineOffset;
        if (dispTable[currentline].blockindex == (table+1)->blockindex)
                offscript += temp;

        (table+1)->lineWidth = WhatTextExtent(&ptr->textData

                        [(table+1)->lineOffset],

                (table+1)->sizeInLine,

                offscript,
```

```
                ptr->script);
            bottom += (table+1)->lineHeight;

            if (UpdateLineHeight(line+1, offscript))
                rc.bottom = clientheight;

            if ((table+1)->lineWidth > descript.rightindent)
                line++;
            else
            {
                if (mode)
                {
                    while (dispTable[line+1].lineWidth <= descript.rightindent
                            && dispTable[line].paraIndex ==
dispTable[line+1].paraIndex)
                        line++;

                    if (dispTable[line].paraIndex == dispTable[line+1].paraIndex)
                        line++;
                    else
                        break;
                }
                else
                    break;
            }
        } while (1);

        /* set invalidate rectangle */

        rc.bottom = max(rc.bottom, bottom);
        InvalidateRect(hwnd, &rc, TRUE);
}
```

```
/* main.c                                                              */
/* written by : koson thambundit                                       */
/* created  5/3/2536                                                   */
/* last modified :- 20/3/2536                                          */
/* purpose :- main routine in program; initialize and                  */
/*                    collect input from user and contribute           */
/*                    to proper routine to handle it.                  */

#include <windows.h>
#include "const.h"
#include "exfunc.h"

#include <stdlib.h>
#include <string.h>
#include "resource.h"

/* export variable */
HDC             hdc;

/* internal variable */
static HANDLE   hInst;

/* call back function */
long FAR PASCAL MainWndProc(HWND, UINT, WPARAM, LPARAM);
BOOL FAR PASCAL  DlgProc(HWND, UINT, WPARAM, LPARAM);

/* description :- main routine , initiallize and                       */
/*                    collect message from message queue               */
```

```
/*                                  and send to window procedure to handle*/
/* input :- hInstance -> ID. number of instance of new*/
/*                                  process.
                */
/*                  hPrevInstance :- ID. number of previouse  */
/*                                  process that run same     */
/*                                  program.
                */
/*                      lpszCmdLine :- command line of program          */
/*                  cmdShow :- status of initial window                 */
/* return :- zero.                                                      */

int PASCAL WinMain(HANDLE  hInstance,
                   HANDLE  hPrevInstance,
                   LPSTR   lpszCmdLine,
                   int     cmdShow)
{
     HWND     hwnd;
     MSG      msg;
     WNDCLASS    wndclass;

     if (!hPrevInstance)
          return 0L;

     hInst = hInstance;

     wndclass.lpszClassName = "DreamWord";
     wndclass.hInstance = hInstance;
     wndclass.lpfnWndProc = MainWndProc;
     wndclass.hCursor = LoadCursor(NULL, IDC_IBEAM);
     wndclass.hIcon = NULL;
     wndclass.lpszMenuName = MAKEINTRESOURCE(1);
     wndclass.hbrBackground = GetStockObject(WHITE_BRUSH);
```

```
        wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
        wndclass.cbClsExtra = 0;
        wndclass.cbWndExtra = 0;

        RegisterClass(&wndclass);

        hwnd = CreateWindow( "DreamWord",
                             "WordProcessor",
                             WS_OVERLAPPEDWINDOW,
                             CW_USEDEFAULT,
                             0,
                             CW_USEDEFAULT,
                             0,
                             NULL,
                             NULL,
                             hInstance,
                             NULL);

        ShowWindow(hwnd, cmdShow);

/* message loop */
        while(GetMessage(&msg, 0, 0, 0))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }

        return 0;
}

/* description :- routine for receive message sent          */
```

```
/*                                  from message queue.
                */
/* input :- hwnd -> handle of window                                    */
/*                          wMsg -> recevied message
                */
/*                          wParam -> word paramater of message         */
/*                          lParam -> long paramater of message         */
/* return :- zero                                                       */

long FAR PASCAL MainWndProc(HWND    hwnd,
                                    UINT    wMsg,
                                    WPARAM      wParam,
                                    LPARAM      lParam)
{
        RECT rc;

   switch (wMsg)
   {
          case WM_CREATE:
            hdc = GetDC(hwnd); /* get own device context */
      SetTextAlign(hdc, TA_UPDATECPITA_BASELINE);
            SetBkMode(hdc, TRANSPARENT);

            if (!TableInitial())
            {
               MessageBox(NULL, "Not enough memory!", NULL, MB_OK);
               PostQuitMessage(0);
            }
      break;

        case WM_SIZE:
            ClientSizeChange(hwnd, lParam);
            break;
```

```
case WM_CHAR:
    InsertBuffer(hwnd, (BYTE)wParam);
    break;

case WM_KEYDOWN:
    switch(wParam)
    {
        case VK_ESCAPE:
            change_thai();
            break;

        case VK_LEFT:
        case VK_RIGHT:
        case VK_UP:
        case VK_DOWN:
        case VK_RETURN:
            MoveCaret(hwnd, wParam);
            break;

        case VK_DELETE:
        case VK_BACK:
            DeleteCharacter(hwnd, wParam);
            break;

        case VK_F1:
            break; /* for debug purpose */
    }
    break;

case WM_PAINT
    UpdateClient(hwnd);
    break,
```

```
        case WM_SETFOCUS:
            GetCaret(hwnd);
            break;

        case WM_KILLFOCUS:
            DestroyCaret();
            break;

        case WM_COMMAND:
            switch (wParam)
            {
            case   IDM_FONT:
                /* choose font */
                    if (ChangeFont(hwnd))
                    {
                        GetClientRect(hwnd ,&rc);
                        rc.top = yPos;
                        InvalidateRect(hwnd, &rc, TRUE);

                        GetCaret(hwnd),
                    } ;
                    break;

                case IDM_INDENT:
                {
                    FARPROC lpfn;
                    LONG        pos;
                    int     index, line, temp;

                    HideCaret(hwnd);

/* change indent */
```

```
temp = descript.rightindent;
lpfn = MakeProcInstance((FARPROC)DlgProc, hInst);
if (DialogBox(hInst, MAKEINTRESOURCE(100), hwnd, lpfn))
{
    /* update entire window */
    if (!dispTable[currentline].blockindex)
    {
        ShowCaret(hwnd);
        break;
    }
    index = dispTable[currentline].paraIndex;

    /* save current position */
    pos = GetCurrentPos(index);

    /* find first line of paragraph */
    line = -1;
    while (dispTable[++line].paraIndex != index);

    Buffer2Block(0); /* update buffer */

    currentline = line;
    MemoryMove(&linebuffer.data[0],
textptr[dispTable[line].blockindex].textData,
                                        dispTable[line].sizeInLine);

    linebuffer.script = textptr[dispTable[line].blockindex]
                                                        .script;
    linebuffer.data[dispTable[line].sizeInLine] = 0;

    if (temp > descript.rightindent)
        BreakLine(hwnd, line, 1);
```

```
                    else
                        CheckFirstword(hwnd, line+1, 1);

                    /* restore position */
                    RestoreCurrentPos(hwnd, pos, line);
                    InvalidateRect(hwnd, NULL, TRUE);
                }

                FreeProcInstance(lpfn);

                CreateCaret(hwnd, NULL, 0, dispTable[currentline].lineBase);
                SetCaretPos(xPos, yPos);
                ShowCaret(hwnd);
            }
            break;
        }
        break;

    case WM_DESTROY:
        FreeTable();
        PostQuitMessage(0);
        break;

    default:
        return(DefWindowProc(hwnd, wMsg, wParam, lParam));
    }
    return 0L;
}

/* description :- dialog handle routine                    */
/* input :-       hdlg -> dialog window handle             */
/*                msg -> message sent to dialog box        */
```

```
/*                      wParam -> word paramater of message                    */
/*                      lParam -> long paramater of message                    */
/* return :- TRUE for new value is received                                    */
/*                       else FLASE                                            */
                        */

BOOL FAR PASCAL DlgProc(HWND      hdlg,
                                  UINT          msg,
                                  WPARAM        wParam,
                                  LPARAM lParam)
{
        char    buffer[3],
        char    ans[8];
        int  i;

        switch (msg)
        {
          case WM_INITDIALOG:
            i = descript.rightindent/logicalinch;
            itoa(i, buffer, 10);
            strcpy(ans, buffer);

            strcat(ans, ".");

            i = descript.rightindent%logicalinch,
            itoa(i, buffer, 10),
            buffer[2] = 0;

            strcat(ans, buffer);

            SetDlgItemText(hdlg, 102, ans),
            SendDlgItemMessage(hdlg, 102, EM_SETSEL, 0, MAKELONG(0, -1)),
            SetFocus(GetDlgItem(hdlg, 102)),
```

```
            break;

        case WM_COMMAND:
            switch (wParam)
            {
                case IDOK :
                    if (GetRightIndent(hdlg))
                        EndDialog(hdlg, TRUE);
                    else
                        EndDialog(hdlg, FALSE);

                    return TRUE;

                case IDCANCEL
                    EndDialog(hdlg, FALSE);
                    return TRUE;
            }
            break;
    }

    return FALSE;
}
```

```
/* mem.c                                                              */
/* wriiten by :- koson thambundit                                     */
/* created : 4/3/2536                                                 */
/* last modified  20/3/36                                             */
/* purpose : to manage data on memory and disk                        */

#include <windows.h>
#include "const.h"
#include "exfunc.h"

#include <windowsx.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <io.h>
#include <errno.h>

/* internal function */
static int MakeSpace(int);


/* export variable */
TEXTBLOCK       FAR *textptr;
LISTHEADER      FIFOlist;/* list in swap map */


/* internal variable */
static HFILE                    swapfile;
static char                     *swapname;
```

```
/* description :- allocate and initial textblock data */
/*                                          structure
                                */
/* input :- none
                                */
/* return :- block index in array of textblock.                    */

int MemoryInitial(void)
{
        int                     count;
        OFSTRUCT        of;
        char                    *env;

        textptr=(TEXTBLOCK FAR *)GlobalAllocPtr(GHND,

                sizeof(TEXTBLOCK)*(DWORD)BLOCKINMEM);
        if (textptr == NULL)
                return 0;

        swapname = (char *)LocalAlloc(LPTR, 64);
        if (swapname == NULL)
                return 0;

        if ((env = getenv("TEMP"))==NULL)
                strcpy(swapname, "swapfile.swp");
        else
        {
                strcpy(swapname, env);
                if (swapname[lstrlen(swapname)-1] == '\\')
                        strcat(swapname, "swapfile.swp");
                else
```

```
                    strcat(swapname, "\\swapfile.swp");
        }

        swapfile = OpenFile(swapname, &of, OF_CREATE);
        if (swapfile==HFILE_ERROR)
                return 0;

        for (count=1, count<BLOCKINMEM; count++)
                textptr[count].size = -1,

        return 1, /* succeed initialize */
}

/* description :- free memory space to system                    */
/* input :- none                                                 */
                */
/* return :- none
                */

void FreeMemory(void)
{
        OFSTRUCT        of;

        GlobalFreePtr(textptr);
        _lclose(swapfile);
        OpenFile(swapname, &of, OF_DELETE),

        LocalFree((HLOCAL)swapname);
}

/* descriptiion :- free textblock and delete it from          */
```

```
/*                                                    list of praragraph
                */
/* input :- current -> block index of deleted block          */
/*                                    para -> paragraph index of deleted block      */
/* return :- none
                */

void DeleteTextBlock(int      current,
                                                          int       para)
{
        /* in display table it's must be updated before */
        int       prev, next;

        prev = textptr[current].prevBlock;
        next = textptr[current].nextBlock;

        if (next)
                textptr[next].prevBlock = prev;

        if (prev)
                textptr[prev].nextBlock = next;
        else
        {
                while (current != paraTable[++prev].blockindex); /* it must find someone */
                paraTable[prev-1].blockindex = next;
        }

        _fmemset(&textptr[current], 0, sizeof(TEXTBLOCK));
        textptr[current].size = -1;
}

/* description - allocate new textblock from memory      */
```

```
/*                                                    space and insert into list of      */
/*              paragraph.                    */
/* input :- current -> block before new block else                    */
/*                                                                    -1 for first block in paragraph*/
/*                                    paraindex -> paragraph index of new block */
/* return :- block index of new block.                                */

int NewTextBlock(int          current,
                                                    int       paraindex)
/* this function allocate      new textblock and follow current in list */
/* return new entry */
{
        int       count=0, temp;

        /* find free block */
        do
        {
                if (textptr[count].size == -1)
                        break;
                else
                        count++;
        } while (count < BLOCKINMEM);

        if (count == BLOCKINMEM)
        /* allocate full segment */
                count = MakeSpace(paraindex);

        if (current != -1)
        /* if current = -1 not insert into list */
        {
                temp = textptr[current].nextBlock;
                textptr[count].prevBlock = current;
                textptr[count].nextBlock = temp;
```

```
            textptr[current].nextBlock = count;
            if (temp)
              textptr[temp].prevBlock = count;
        }

        return count; /* new entry */
}

/* description :- move data in dest to source for count*/
/* input :- dest -> pointer of destination location.        */
/*                 source -> pointer of source location.    */
/*                 count -> bytes be moved
        */
/* return :- none
                */

void    MemoryMove(BYTE  FAR    *dest,
                                BYTE  FAR     *source,
                                UINT          count)
{
        WORD    extra, data;
        WORD    destoff, souroff;

        extra = HIWORD(dest);
        data = HIWORD(source);
        destoff = LOWORD(dest),
        souroff = LOWORD(source);

        _asm {   pushf
                        push    ds
                        push    es
```

```
                push    si
                push    di

                cld
                mov             si, souroff
                mov             di, destoff
        }

if (extra == data)
        if (LOWORD(source) < LOWORD(dest))
                _asm {  add     si, count
                                dec     si
                                add     di, count
                                dec di
                                std
                        }
_asm {
                mov             ax, extra
                mov             es, ax
                mov             ax, data
                mov             ds, ax
                mov             cx, count

                rep     movs

                pop             di
                pop             si
                pop             es
                pop             ds
                popf
        }
}
```

```
/* description :- swap textblock in swap file to main */
/*                                                  memory
                                                              */
/* input :- fileoffset -> offset from begin of swap   */
/*                                                              file that this swap stay.    */
/*                                  paraindex -> paragraph index of swapped   */
/*                                                              block.
                                                              */
/* return . index of swapped textblock                                          */

int SwapTextBlock(LONG     fileoffset,
                                                    int       paraindex)
{
        TEXTBLOCK               tempblock;
        TEXTBLOCK FAR *ptr;
        int                     blockindex;

        _lseek(swapfile, fileoffset, SEEK_SET);
        _lread(swapfile, &tempblock, sizeof(TEXTBLOCK));

        /* update FIFO list */
        blockindex = FIFOlist.first;
        ptr = &textptr[blockindex];
        FIFOlist.first = ptr->list;

        /* update chain of textblock in same paragraph */
        if (!ptr->prevBlock)
        {
                paraTable[paraindex].blockindex = -1;
                paraTable[paraindex].fileoffset = fileoffset;
        }
        else
```

```
{
        if (ptr->prevBlock == -1)
        {
          if (ptr->prevOffset != -1)
          {
                        _lseek(swapfile, ptr->prevOffset+2, SEEK_SET);
                        _lwrite(swapfile, &fileoffset, sizeof(LONG));
                }
        }
        else
        {
                textptr[ptr->prevBlock].nextBlock = -1;
                textptr[ptr->prevBlock].nextOffset = fileoffset;
        }
}

if (!ptr->nextBlock)
        if (ptr->nextBlock == -1)
        {
          if (ptr->nextOffset != -1)
          {
                        _lseek(swapfile, ptr->nextOffset, SEEK_SET);
                        _lwrite(swapfile, &fileoffset, sizeof(LONG));
          }
        }
        else
        {
          textptr[ptr->nextBlock].prevBlock = -1;
          textptr[ptr->nextBlock].prevOffset = fileoffset;
        }

/* update data in memory to swapfile */
_lseek(swapfile, fileoffset, SEEK_SET);
```

```
        _lwrite(swapfile, ptr, sizeof(TEXTBLOCK));

        MemoryMove((BYTE FAR *)ptr,
                                (BYTE FAR *)&tempblock,
                                sizeof(TEXTBLOCK));

        return blockindex;
}

/* description :- swap first textblock in LRU list          */
/*                                          into disk
                                ..*/
/* input :- paragraph index                                 */
/* return : entry swapped
        */
/*.update · displaytable and paragraph table incase of*/
/*                          first block
                            */

static int MakeSpace(int      paraindex)
{
        int                         blockindex;
        TEXTBLOCK FAR *ptr;
        LONG                        fileoffset,

        blockindex = FIFOlist.first;
        ptr = &textptr[blockindex],
        FIFOlist first = ptr->list;

        fileoffset = _lseek(swapfile, 0, SEEK_END),
        _lwrite(swapfile, ptr, sizeof(TEXTBLOCK));
```

```
/* update chain of textblock in same paragraph */
if (!ptr->prevBlock)
{
        paraTable[paraindex].blockindex = -1;
        paraTable[paraindex].fileoffset = fileoffset;
}
else
{
        if (ptr->prevBlock == -1)
        {
          if (ptr->prevOffset != -1)
          {
                        _lseek(swapfile, ptr->prevOffset+2, SEEK_SET);
                        _lwrite(swapfile, &fileoffset, sizeof(LONG));
                }
        }
        else
        {
                textptr[ptr->prevBlock].nextBlock = -1;
                textptr[ptr->prevBlock].nextOffset = fileoffset;
        }
}

if (!ptr->nextBlock)
        if (ptr->nextBlock == -1)
        {
          if (ptr->nextOffset != -1)
          {
                        _lseek(swapfile, ptr->nextOffset, SEEK_SET);
                        _lwrite(swapfile, &fileoffset, sizeof(LONG));
          }
        }
        else
```

```
        {
          textptr[ptr->nextBlock].prevBlock = -1;
          textptr[ptr->nextBlock].prevOffset = fileoffset;
        }

    return blockindex;
}
```