บทที่ 7
การจัดการหน่วยความจำหลัก
(Memory Management)

การจัดการหน่วยประมวลผลกลางให้มีประสิทธิภาพต้องมีให้กระบวนการ (Process) หลายตัว ได้มีโอกาสแบ่งปันกันทำงานซึ่งทำให้ ประสิทธิผลการใช้งานหน่วยประมวลผลกลางดีขึ้น และอัตราการตอบสนองต่อผู้ใช้เร็วขึ้น เพื่อให้ได้ประสิทธิภาพอย่างสมบูรณ์ จึงจำเป็นต้องให้มีหลายกระบวนการอยู่ในหน่วยความจำหลักพร้อม ๆ กัน โดยแบ่งกระบวนการเป็นส่วน ๆ

ในบทนี้ จะแสดงวิธีการจัดการหน่วยความจำหลักแบบต่าง ๆ เริ่มตั้งแต่แบบง่ายที่สุดจนถึงแบบแบ่งเป็นหน้า (Paging) หรือเป็นตอน (Segmentation) หรือแบบผสมหน้าและตอน (Page Segmentation) แต่ละวิธี มีข้อดี-ข้อเสียแตกต่างกัน การเลือกวิธีที่เหมาะสมกับระบบหนึ่ง ๆ นั้นขึ้นอยู่กับปัจจัยหลายประการ โดยเฉพาะอย่างยิ่ง เรื่องฮาร์ดแวร์ของระบบ เพราะแต่ละวิธีจำเป็นต้องมีฮาร์ดแวร์ช่วยเฉพาะอย่าง

วิธีการจัดการหน่วยความจำหลักทุกวิธีในบทนี้ จะมีข้อจำกัดเหมือนกัน คือโปรแกรมทั้งหมดของกระบวนการหนึ่งจะต้องอยู่ในหน่วยความจำหลักก่อนที่กระบวนการนั้น จะเริ่มทำงาน ซึ่งหมายความว่า ขนาดของกระบวนการจะต้องไม่ใหญ่กว่าขนาดของหน่วยความจำหลัก

ความเป็นมา

ดังจะเห็นได้ว่าหน่วยความจำหลักเป็นศูนย์กลางของการทำงานต่าง ๆ ในระบบปัจจุบันหน่วยความจำหลักคือ แถวลำดับ หรือฮาร์ดแวร์ขนาดใหญ่ ที่มีประกอบด้วยที่เก็บข้อมูลย่อย ๆ เป็นไบท์ (Byte) หรือ คำ (Word) โดยแต่ละไบท์จะมีเลขที่บอกตำแหน่ง (Address) ของตัวเอง การใช้งานหน่วยความจำหลัก ทำได้โดยการอ่าน หรือเขียนข้อมูลที่ตำแหน่งต่าง ๆ เหล่านี้ โดยปกติแล้วหน่วยประมวลผลกลางจะเป็นผู้ใช้งาน

คำสั่งในฮาร์ดแวร์ทั่วไปจะทำงานตามลำดับ โดยเริ่มจากการอ่านคำสั่งจากหน่วยความจำหลักแล้วแปลความหมายจากคำสั่งนั้น ซึ่งอาจมีการอ่านตัวแปรอื่นเพิ่มเติมจากหน่วยความจำหลักเพื่อมาคำนวณ หรือกระทำการอ่านหลังจากคำนวณแล้ว อาจมีการเขียนผลลัพธ์ที่ได้ออกไปยังหน่วยความจำหลัก สังเกตว่าหน่วยความจำหลักจะเห็นแต่ตัวข้อมูลที่เก็บอยู่เท่านั้น ไม่ทราบที่มาหรือความสัมพันธ์ต่าง ๆ (เช่น เป็นตัวชี้คำสั่ง, ตัวดัชนี, การอ้างถึงตำแหน่งโดยอ้อม หรือเป็นตำแหน่งค่าคงที่ เป็นต้น) หรือทราบว่าคำเหล่านั้นใช้ทำอะไรบ้าง (เป็นข้อมูลดิบหรือส่วนของ

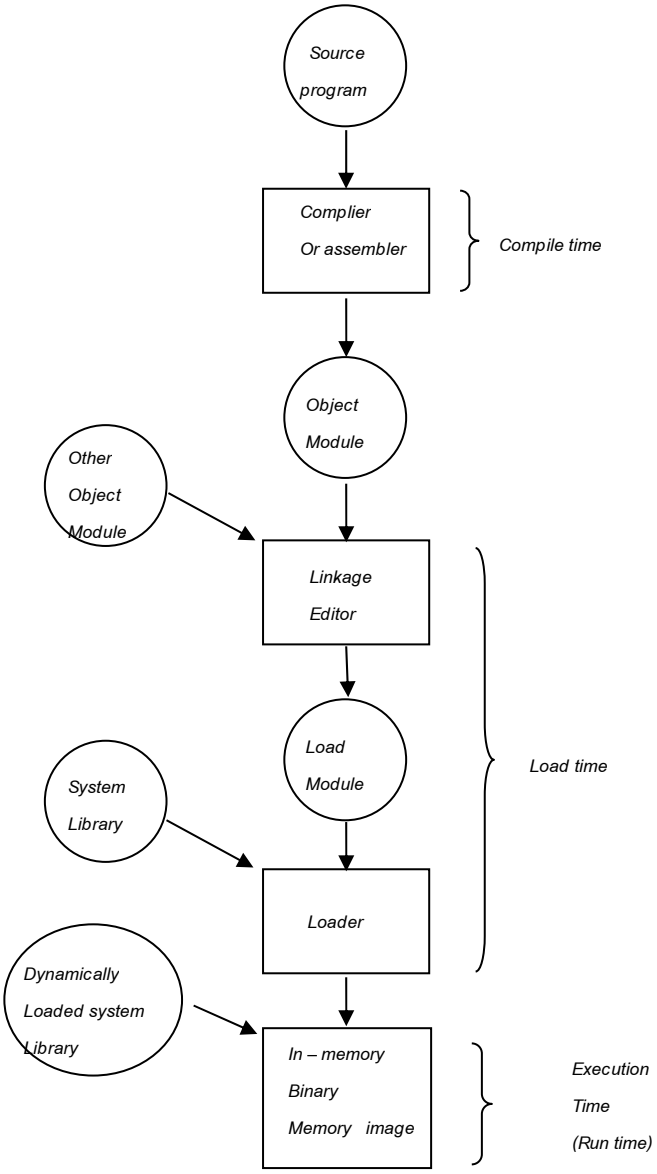
โปรแกรม) เราไม่สนใจว่าโปรแกรมคำนวณค่าตำแหน่ง ในหน่วยความจำหลักเหล่านี้มาได้อย่างไร เราสนใจแต่ตำแหน่งต่าง ๆ ในหน่วยความจำที่กระบวนการต้องการใช้งานเท่านั้น

การกำหนดตำแหน่ง

ในการทำงานของระบบคอมพิวเตอร์จะต้องมีการนำกระบวนการลงสู่หน่วยความจำหลักก่อนที่จะสามารถให้กระบวนการนั้นทำงานได้ ปกติแล้วกระบวนการหรือโปรแกรมต่าง ๆ ที่เก็บอยู่ในหน่วยความจำช่วย (Secondary Storage) จะเก็บอยู่ในรูปของแฟ้มข้อมูลเลขฐานสอง (Binary Executable File) ที่สามารถใช้งานได้เลย ในหน่วยความจำช่วย (Secondary Storage) อาจมีกระบวนการหรือโปรแกรมหลายตัว ซึ่งรอคอยที่จะเข้าทำงานในหน่วยความจำหลัก เราเรียกกลุ่มกระบวนการนี้ว่าแถวคอยขาเข้า (Input Queue)

โดยทั่วไประบบจะเลือกกระบวนการหนึ่ง จากแถวคอยขาเข้านี้มาทำงานในหน่วยความจำหลัก ซึ่งมักต้องมีการย้ายตำแหน่ง (Relocate) หรือเชื่อมต่อกับโปรแกรมระบบอื่น ๆ ที่เกี่ยวข้องก่อนที่จะทำงานได้ เมื่อกระบวนการทำงาน กระบวนการจะอ่านหรือเขียนข้อมูลและโปรแกรมโดยตรงจากหน่วยความจำหลัก เมื่อกระบวนการสิ้นสุดหรือทำงานเสร็จพื้นที่ในหน่วยความจำที่กระบวนการครอบครองอยู่จะว่างลงกระบวนการต่อไป จะสามารถเข้ามาใช้งานพื้นที่หน่วยความจำนี้ได้ กระบวนการของผู้ใช้อาจทำงานอยู่ในส่วนใดของหน่วยความจำหลักก็ได้ ตำแหน่งเริ่มต้นของหน่วยความจำหลัก คือ 0000 ส่วนตำแหน่งเริ่มต้นของพื้นที่ของกระบวนการผู้ใช้อาจเป็นค่าใดก็ได้ ซึ่งจะมีผลต่อการทำงานของโปรแกรมผู้ใช้ (User Program) มาก โดยปกติโปรแกรมของผู้ใช้ต้องผ่านขั้นตอนหลายขั้น ดังรูปที่ 7.1 (ในบางกรณีบางขั้นอาจไม่มีก็ได้) ตำแหน่งหน่วยความจำ (Address) ภายในโปรแกรมของผู้ใช้อาจแสดงได้หลายแบบแตกต่างกันตามขั้นตอนต่าง ๆ ในโปรแกรมต้นฉบับ (Source Program) ตำแหน่งเหล่านี้มักเขียนแทนด้วยตัวแปร เช่น a, b ในการทำงานตัวแปลภาษา (Compiler) จะแปลตัวแปรเหล่านี้ ให้เป็นตำแหน่งแบบสัมพัทธ์ (Relative Address) เช่น มีค่าเป็นระยะห่างจากตำแหน่งเริ่มต้นของโปรแกรม 14 ไบท์ ตัวเชื่อมต่อโปรแกรม หรือตัวนำโปรแกรมลงหน่วยความจำหลัก จะแปลค่าตำแหน่งแบบสัมพัทธ์นี้ไปเป็นตำแหน่งสัมบูรณ์ (Absolute Address) หรือตำแหน่งจริงทางกายภาพ (Physical Address) เช่น ตำแหน่ง 74014 การกำหนดค่าตำแหน่งในแต่ละขั้นตอน จะเปลี่ยนค่าตำแหน่งจากรูปแบบหนึ่ง ไปยังอีกรูปแบบหนึ่ง

โดยปกติการกำหนดค่าตำแหน่งในหน่วยความจำ (Binding Address) ของโปรแกรมและข้อมูลจะเกิดขึ้นในขั้นตอนต่าง ๆ ดังนี้



รูปที่ 7.1 ขั้นตอนการเข้าทำงานของผู้ใช้ (ที่มาSilberschatz,Gavin;1994)

ช่วงการแปล (Compile Time) ถ้าเรารู้ว่ากระบวนการหรือโปรแกรมจะทำงานที่ตำแหน่งใด ในหน่วยความจำหลัก ตัวแปลภาษาก็สามารถแปลโปรแกรม โดยใช้ตำแหน่งสัมบูรณ์ (Absolute Address) หรือตำแหน่งจริง (Physical Address) ได้เลย ตัวอย่างเช่น ถ้ากำหนดให้กระบวนการของผู้ใช้ เริ่มต้นที่ตำแหน่ง R ตัวแปรภาษา ก็สามารถแปลตำแหน่งต่าง ๆ ได้โดยเริ่มจากคำ R นี้ไป ถ้ามีการเปลี่ยนตำแหน่งเริ่มต้น ก็แปลโปรแกรมกันใหม่อีกครั้ง แฟ้มข้อมูลที่มีนามสกุล .com ของระบบ MS-DOS เป็นแบบตำแหน่งสัมบูรณ์ ค่าตำแหน่งในโปรแกรมจะถูกกำหนดตั้งแต่ช่วงการแปลโปรแกรม

ช่วงการนำโปรแกรมลงสู่หน่วยความจำหลัก (Load Time) ถ้าในขณะที่แปลโปรแกรมเราจะไม่รู้ว่าโปรแกรมจะทำงานที่ตำแหน่งใดในหน่วยความจำหลัก ตัวแปลโปรแกรมก็จะต้องแปลตำแหน่งเป็นแบบย้าย ๆ (Relative Address) ดังนั้น การกำหนดตำแหน่งจริงจะเกิดขึ้นขณะที่นำโปรแกรมลงสู่หน่วยความจำหลัก ถ้ามีการเปลี่ยนตำแหน่งเริ่มต้นใหม่ เราก็เพียงนำโปรแกรมลงสู่หน่วยความจำหลักใหม่อีกครั้งหนึ่ง

ช่วงการทำงาน (Execution Time) กระบวนการอาจถูกย้ายไปมาได้ในระหว่างการทำงาน การกำหนดค่าตำแหน่งจริง (Real Address) จะกระทำก่อนไม่ได้เลย ต้องทำในขณะที่ทำงาน (Execute) จริงเท่านั้น จะต้องมีอุปกรณ์พิเศษ (ฮาร์ดแวร์) ช่วย จึงจะสามารถใช้วิธีนี้ได้

ก่อนที่จะดำเนินงานต้องนำโปรแกรมไปใส่ไว้ในหน่วยความจำก่อน เพื่อให้สามารถใช้หน่วยความจำได้อย่างมีประสิทธิภาพสูงสุด เราอาจนำโปรแกรมลงสู่หน่วยความได้หลายวิธีคือ การนำโปรแกรมลงสู่หน่วยความจำแบบสัมพัทธ์ (Dynamic Loading) การเชื่อมต่อโปรแกรมแบบสัมพัทธ์ (Dynamic Linking) และการแบ่งส่วน (Overlays) แต่ละแบบมีวิธีการดังต่อไปนี้

การนำโปรแกรมลงสู่หน่วยความจำแบบสัมพัทธ์

เพื่อให้สามารถใช้หน่วยความจำได้อย่างมีประสิทธิภาพสูงสุด เราอาจใช้วิธีการนำโปรแกรมลงสู่หน่วยความจำแบบสัมพัทธ์ วิธีนี้จะไม่นำโปรแกรมย่อย (Subprogram) ลงสู่หน่วยความจำพร้อมกับโปรแกรมหลัก (Main Program) แต่จะนำลงเมื่อมีการเรียกใช้เท่านั้น โปรแกรมย่อย (Subprogram) ทั้งหมดจะอยู่ในหน่วยความจำช่วย (Secondary Storage) ในรูปแบบที่ย้ายตำแหน่งได้ เมื่อโปรแกรมหลัก (Main Program) ต้องการเรียกโปรแกรมย่อยใด ก็จะตรวจดูว่าโปรแกรมย่อยนั้นอยู่ในหน่วยความจำแล้วหรือยัง ถ้ายังก็จะนำโปรแกรมย่อยนั้นลงสู่หน่วยความจำหลักโดยย้ายตำแหน่งไปไว้ในที่ที่เหมาะสม และย้ายการควบคุมไปไว้ที่โปรแกรมย่อยนั้นเพื่อทำงานต่อไป

ประโยชน์ของวิธีนี้ คือ โปรแกรมย่อยที่ไม่ได้ใช้ก็จะไม่ถูกนำลงสู่หน่วยความจำหลัก ให้เสียเนื้อที่เปล่า ๆ วิธีนี้เหมาะกับโปรแกรมย่อยที่ใช้สำหรับเหตุการณ์ซึ่งเกิดขึ้นไม่บ่อย เช่น โปรแกรมจัดการข้อผิดพลาดต่าง ๆ เป็นต้น วิธีนี้จะทำให้โปรแกรมหลัก (Main Program) ที่ใช้งานจริงอาจมีขนาดเล็กกว่า โปรแกรมทั้งหมดมาก วิธีนี้ไม่จำเป็นต้องพึ่งคำสั่งพิเศษใด ๆ จากระบบปฏิบัติการแต่ผู้ใช้ต้องเป็นผู้จัดการบริหารโปรแกรมนี้เอง

การเชื่อมต่อโปรแกรมแบบสัมพัทธ์

ในรูปที่ 7.1 จะเห็นว่ามีโปรแกรมของระบบเชื่อมต่อแบบสัมพัทธ์ ระบบปฏิบัติการส่วนใหญ่จะมีการเชื่อมต่อโปรแกรมแบบคงที่เท่านั้น โปรแกรมย่อยต่าง ๆ ของระบบ ที่จัดไว้ให้ภาษาต่าง ๆ จะใช้งานเหมือนหน่วยความจำช่วยอื่น ๆ คือ ต้องนำไปเชื่อมต่อกับโปรแกรมหลักของผู้ใช้

รวมเป็นโปรแกรมเลขฐานสอง (Binary Program) ที่พร้อมจะใช้งาน ก่อนที่จะไปทำงานจริงส่วน วิธีการเชื่อมต่อแบบสัมพัทธ์นี้จะคล้ายกับ การนำโปรแกรมลงสู่หน่วยความจำแบบสัมพัทธ์ โดยหน่วงเวลาการเชื่อมต่อโปรแกรม แทนที่จะหน่วงเวลาการนำโปรแกรมลงสู่หน่วยความจำ วิธีนี้มักใช้กับโปรแกรมย่อยของระบบที่เป็นโปรแกรมสนับสนุน (System Libraries) ในภาษาต่าง ๆ โดยปกติ โปรแกรมของผู้ใช้ทุกโปรแกรมต้องเชื่อมต่อ และรวมเอาโปรแกรมย่อยของระบบที่ต้องการใช้ไว้ในโปรแกรมหลักของผู้ใช้เองซึ่งทำให้เสียเนื้อที่ในหน่วยความจำช่วยและในหน่วยความจำหลักมาก (เนื่องจากทุกโปรแกรมต่างมีส่วนของโปรแกรมย่อยของระบบซ้ำซ้อนกัน) ส่วนในวิธีการเชื่อมต่อแบบสัมพัทธ์ ขณะเชื่อมต่อโปรแกรมย่อยจะมีการสร้างชุดคำสั่งเรียก (Stub) ซึ่งมีขนาดเล็กมาก เพื่อใช้เรียกโปรแกรมย่อย (Subprogram) ที่ต้องการขณะทำงาน โปรแกรมย่อยของระบบมักจะอยู่ประจำในหน่วยความจำหลัก ชุดคำสั่งเรียก (Stub) นี้จะทำงานโดยการเปลี่ยนโปรแกรมตัวเอง เป็นตำแหน่งของโปรแกรมย่อยในระบบและเรียกโปรแกรมย่อยนั้น ดังนั้นเมื่อมีการใช้ชุดคำสั่งเรียกนี้อีกครั้งโปรแกรมย่อยของระบบ ก็จะถูกเรียกทำงานโดยตรง โดยไม่เสียเวลาทำงานในชุดคำสั่งเรียกนี้อีกต่อไป วิธีนี้ยังประหยัดเนื้อที่มากอีกด้วย เพราะโปรแกรมของผู้ใช้ต่างใช้โปรแกรมย่อยของระบบอันเดียวกันไม่เกิดการซ้ำซ้อน

นอกจากนั้น วิธีนี้ยังสนับสนุน การแก้ไขปรับปรุงโปรแกรมย่อยของระบบอีกด้วย เมื่อมีการแก้ไขปรับปรุงโปรแกรมย่อยของระบบ เช่น การกำจัดข้อผิดพลาดเล็กน้อย โปรแกรมผู้ใช้ทั้งหลายก็จะใช้โปรแกรมย่อยใหม่ได้ทันที ถ้าเป็นการเชื่อมต่อแบบคงที่โปรแกรมผู้ใช้ต้องทำการเชื่อมต่อกับโปรแกรมย่อยของระบบใหม่ทั้งหมด เราอาจป้องกันการใช้โปรแกรมย่อยรุ่นใหม่ ซึ่งอาจเข้ากันไม่ได้ในระบบเชื่อมต่อแบบสัมพัทธ์ได้ โดยใส่ข้อมูลของโปรแกรมย่อยรุ่นลงในโปรแกรมด้วย เมื่อมีการแก้ไขปรับปรุงโปรแกรมย่อยของระบบถ้ายังเข้ากับรุ่นเดิมได้ก็ใช้หมายเลขรุ่นเดิม ถ้าเป็นการปรับปรุงให้เข้ากับรุ่นเดิมไม่ได้ก็ใช้หมายเลขรุ่นใหม่ เราจึงจำเป็นต้องมีโปรแกรมย่อยของระบบรุ่นต่าง ๆ ในหน่วยความจำหลัก เพื่อให้โปรแกรมที่เชื่อมต่อแบบสัมพัทธ์ก่อนมีการแก้ไข สามารถทำงานได้ถูกต้อง ระบบนี้เรียกว่าโปรแกรมย่อยร่วม (Shared Libraries)

การแบ่งส่วน

เนื่องจาก ต้องมีการนำโปรแกรมทั้งหมดของกระบวนการหนึ่ง ลงในหน่วยความจำก่อนจะเริ่มทำงาน ดังนั้นขนาดของกระบวนการจะถูกจำกัดด้วยขนาดของหน่วยความจำหลัก เราจึงต้องมีการแบ่งส่วนกระบวนการออกเป็นส่วน ๆ เพื่อให้สามารถทำงานในหน่วยความจำหลักได้ การแบ่งส่วนนี้ทำโดยแบ่งส่วนของโปรแกรมและข้อมูลที่ต้องการใช้ในช่วงเวลาแรกให้เข้าทำงานในหน่วยความจำหลัก แล้วนำโปรแกรมช่วงต่อไปลงทับโปรแกรมเดิม หลังจากที่โปรแกรมช่วงแรกทำงานเสร็จแล้ว ตัวอย่างเช่น ตัวแปลภาษาแอสแซมบลี้ แบบแปล 2 รอบ ในรอบแรกจะมีการสร้าง

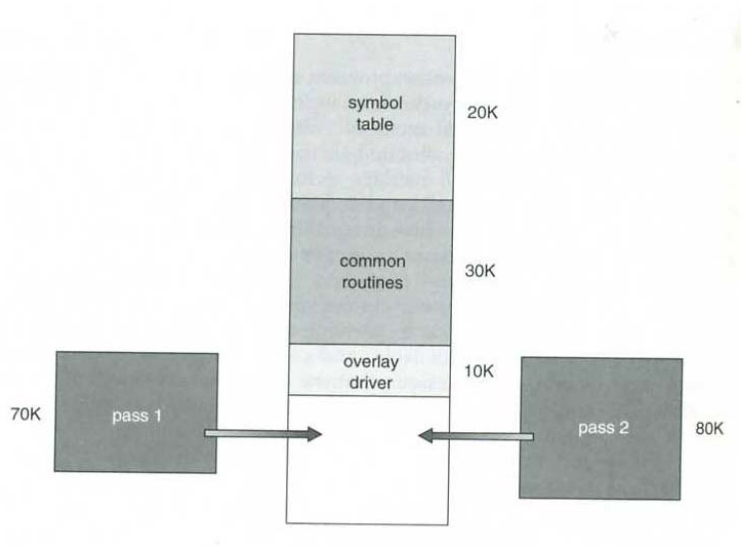
ตารางสัญลักษณ์และรอบที่ 2 จะเป็นการแปลภาษาเครื่อง เราอาจทำการแบ่งส่วนตัวแปลภาษานี้ออกเป็น 2 ส่วน คือ ส่วนของการแปลรอบที่หนึ่ง และส่วนของการแปลรอบที่สอง สมมุติให้โปรแกรมตัวแปลภาษามีขนาด ดังนี้

1. ส่วนการแปลรอบที่ 1	70 K ไบท์
2. ส่วนการแปลรอบที่ 2	80 K ไบท์
3. ตารางสัญลักษณ์	20 K ไบท์
4. โปรแกรมย่อยร่วม	30 K ไบท์
	200 K ไบท์

ถ้าเราต้องการนำโปรแกรมทั้งหมดลงสู่หน่วยความจำหลัก เราต้องมีหน่วยความจำหลักขนาดอย่างน้อย 200 K ไบท์ ถ้าเรามีเพียง 150 K ไบท์ (ที่เหลือจากส่วนของระบบปฏิบัติการ) ก็จะทำงานไม่ได้ เราจึงต้องใช้วิธีแบ่งส่วนโดยกำหนดส่วนแรก A ประกอบด้วยตารางสัญลักษณ์โปรแกรมย่อยร่วมและส่วนการแปลรอบที่หนึ่ง ส่วนที่สอง B ประกอบด้วยตารางสัญลักษณ์โปรแกรมย่อยร่วมและส่วนการแปลรอบที่สอง

เราเพิ่มตัวควบคุมการแบ่งส่วนขนาด 10 K และรวมกับส่วนแรก A นำลงในหน่วยความจำหลัก หลังจากนำการแปลรอบที่หนึ่งเสร็จแล้ว ตัวควบคุมการแบ่งส่วนจะอ่านส่วน B ลงสู่พื้นที่ของส่วน A และส่งการควบคุมต่อไปยังการแปลรอบที่สอง จะเห็นได้ว่าส่วนแบ่ง A มีขนาด 120 K และส่วนแบ่ง B มีขนาด 130 K ดังนั้น เราสามารถทำการแปลภาษาแอสแซมบลี้ โดยใช้เนื้อที่หน่วยความจำหลัก ที่มีเพียง 150 K แต่อาจช้าลงเล็กน้อยเพราะต้องมีการอ่านส่วน B เข้ามาระหว่างการทำงานด้วย

โปรแกรมของส่วน A และ B จะเก็บอยู่ในหน่วยความจำสำรองในแบบตำแหน่งสัมบูรณ์ การทำโปรแกรมแบบแบ่งส่วนนี้ต้องมีวิธีการกำหนดตำแหน่งและเชื่อมต่อเฉพาะอย่างแต่ไม่จำเป็นต้องใช้คำสั่งพิเศษใด ๆ จากระบบปฏิบัติการ เหมือนกับการนำโปรแกรมลงสู่หน่วยความจำแบบสัมพัทธ์เราอาจแบ่งโปรแกรมผู้ใช้เป็นแฟ้มข้อมูลหลายแฟ้มแยกกัน และอ่านเข้ามาทีละแฟ้มระบบปฏิบัติการเพียงแต่เห็นว่ามีการอ่านข้อมูลเพิ่มขึ้นกว่าแบบไม่ได้แบ่งส่วนเท่านั้น ผู้เขียนโปรแกรมต้องออกแบบการแบ่งส่วนเองทั้งหมด ซึ่งเป็นงานที่ค่อนข้างยาก เพราะผู้เขียนต้องมีความรู้เรื่องโครงสร้างของโปรแกรมการทำงาน ตลอดจนโครงสร้างข้อมูลต่าง ๆ ที่ใช้ในโปรแกรมเป็นอย่างดี เนื่องจากโปรแกรมมีขนาดใหญ่ ถ้าโปรแกรมขนาดเล็กก็คงไม่ต้องแบ่งส่วน

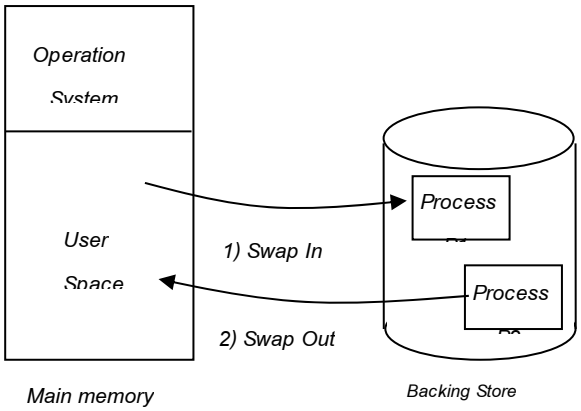


รูปที่ 7.2 การแบ่งส่วนสำหรับตัวแปลภาษาแอสแซมบลี้แบบแปล 2 รอบ

จึงยิ่งเป็นการยากที่จะเข้าใจการทำงานทั้งหมด วิธีการแบ่งส่วนนี้จึงใช้เฉพาะในเครื่องคอมพิวเตอร์ขนาดเล็กหรือระบบที่มีหน่วยความจำหลักจำกัดจริง ๆ แต่ไม่มีฮาร์ดแวร์ช่วย (ในการจัดการขั้นสูงแบบอื่น ๆ) เท่านั้น การจัดแบบอัตโนมัติเป็นสิ่งจำเป็นอย่างยิ่ง สำหรับโปรแกรมขนาดใหญ่

การสับเปลี่ยน

กระบวนการที่กำลังทำงานต้องอยู่ในหน่วยความจำหลักเสมอแต่อาจถูกย้ายไปอยู่ในหน่วยเก็บโปรแกรมชั่วคราว (Backing Store) ได้แล้วนำกลับมาใหม่เพื่อทำงานต่อในระบบการ



รูปที่ 7.3 การสับเปลี่ยน 2 กระบวนการบนจานบันทึก

ทำงานแบบหลายโปรแกรมที่ใช้การจัดตารางการทำงานแบบเวียนเทียน (Round Robin) เมื่อส่วนแบ่งเวลา (Time Quantum) สิ้นสุด ตัวจัดการหน่วยความจำจะย้ายกระบวนการปัจจุบันออกไปและเอากระบวนการใหม่เข้าทำงานแทน ดังรูปที่ 7.3

ในขณะเดียวกันตัวจัดตารางการทำงานของหน่วยประมวลผลกลาง ก็จะจัดเวลาให้กระบวนการใหม่ได้ทำงานบ้าง จะเห็นว่ามีการสับเปลี่ยนกันระหว่างกระบวนการเก่าและกระบวนการใหม่จากหน่วยความจำหลักสู่หน่วยเก็บโปรแกรมชั่วคราว ในทางทฤษฎีตัวจัดการหน่วยความจำต้องสามารถสับเปลี่ยนกระบวนการได้เร็วมาก จนกระทั่ง ตัวจัดตารางหน่วยประมวลผล (CPU Scheduling) เห็นว่ามีกระบวนการอยู่ในหน่วยความจำตลอดเวลา ส่วนแบ่งเวลาต้องมีขนาดใหญ่พอที่จะคุ้มค่ากับเวลาที่เสียไปในการสับเปลี่ยนกระบวนการ (Swap Process) หรืองานในแต่ละครั้ง

การสับเปลี่ยนงาน (Swap Process) อาจเกิดในระบบที่ใช้ระดับความสำคัญเป็นเกณฑ์ (Priority Based) ในการจัดตารางการทำงาน เมื่อมีงานที่มีระดับความสำคัญสูงเข้ามาในระบบตัวจัดการหน่วยความจำอาจสับเปลี่ยนให้งานที่มีระดับความสำคัญต่ำ ออกไปพักชั่วคราวแล้วให้งานระดับความสำคัญสูงเข้าทำงานจนเสร็จก่อน จึงสับงานระดับความสำคัญต่ำกลับมาทำงานต่อไป บางครั้งเรียกวิธีนี้ว่า หมุนออกและหมุนเข้า (Roll-out Roll-in)

โดยปกติงานที่ถูกสับเปลี่ยนออกไปจะถูกสับเปลี่ยนกลับมา ณ ตำแหน่งเดิม แต่ทั้งนี้ขึ้นอยู่กับวิธีการกำหนดตำแหน่ง (Addressing Binding) ที่ใช้ ถ้าการกำหนดตำแหน่งเกิดขึ้นในช่วงการแปลหรือช่วงการนำโปรแกรมลงสู่หน่วยความจำหลักเราก็จะไม่สามารถย้ายงานไป ณ ตำแหน่งอื่นได้ แต่ถ้าการกำหนดตำแหน่งเกิดขึ้นในช่วงการทำงาน เราอาจสับเปลี่ยนงานกลับเข้ามา ณ ตำแหน่งใดก็ได้

การสับเปลี่ยนงานต้องมีหน่วยเก็บโปรแกรมชั่วคราว ซึ่งปกติใช้หน่วยความจำช่วย (Secondary Storage) ความเร็วสูงที่มีขนาดใหญ่พอที่จะเก็บงานทั้งหมดไว้ได้ในคราวเดียวกัน และต้องสามารถอ่าน-เขียน ณ ตำแหน่งใดก็ได้โดยตรง (Random Access) ระบบจะมีข้อมูลของแถวพร้อม (Ready Queue) ของงานทั้งหมดที่อยู่ในหน่วยเก็บโปรแกรมชั่วคราวนี้ และข้อมูลนี้ต้องอยู่ในหน่วยความจำหลัก เมื่อตัวจัดตารางหน่วยประมวลผล (CPU Scheduling) เลือกงานที่จะทำต่อไปได้ก็จะร้องขอให้ตัวย้ายงาน (Dispatcher) นำกระบวนการเข้าทำงาน ตัวย้ายงานจะตรวจดูว่ากระบวนการที่ต้องการอยู่ในหน่วยความจำหลักหรือไม่ ถ้าไม่อยู่แลหน่วยความจำหลักเต็ม ตัวย้ายงานก็จะย้ายเอากระบวนการบางตัวในหน่วยความจำหลักออกไป และสับเอากระบวนการ ที่ต้องการเข้ามาแทนที่

จะเห็นได้ว่า เวลาที่ใช้ในการสับเปลี่ยนงานนี้ (Context-switch Time) ค่อนข้างมาก สมมติว่ากระบวนการผู้ใช้มีขนาด 100 K ไบท์ หน่วยเก็บโปรแกรมชั่วคราว มีอัตราการถ่ายโอนขอมูล 1 M ไบท์ต่อวินาที ดังนั้นเวลาที่ใช้การถ่ายโอนกระบวนการของผู้ใช้จะเท่ากับ 100 K / 1 M = 100 มิลลิวินาที และถ้าอัตราเฉลี่ยในการที่หัวอ่านเคลื่อนไปยังตำแหน่งที่ต้องการ (Latency Time) เป็น 8 มิลลิวินาที เวลาในการสับเปลี่ยนจะเป็น 100+8= 108 มิลลิวินาที การสับเปลี่ยนงานมักต้องสับเปลี่ยนงานเก่าออกและสับเปลี่ยนงานใหม่เข้าแทนที่ ดังนั้นเวลารวมในการสับเปลี่ยนงานครั้งหนึ่ง ๆ จะเท่ากับ 216 มิลลิวินาที

จะสังเกตเห็นว่า เวลาที่ใช้ในการสับเปลี่ยนงานส่วนใหญ่ คือ เวลาการถ่ายโอนข้อมูล เวลาใช้จะเป็นสัดส่วนโดยตรงกับขนาดของหน่วยความจำที่สับเปลี่ยน ถ้าระบบมีหน่วยความจำหลัก 1000 K ไบท์ ใช้พื้นที่โดยเป็นระบบปฏิบัติการ 100 K ไบท์ กระบวนการผู้ใช้จะมีขนาดใหญ่ที่สุดไม่เกิน 900 K ไบท์ กระบวนการผู้ใช้ขนาดเล็ก ๆ เช่น 100 K ไบท์จะเสียเวลาถ่ายโอนข้อมูล 108 มิลลิวินาทีเทียบกับกระบวนการ ที่มีขนาด 900 K ไบท์ ใช้เวลา 908 มิลลิวินาที ดังนั้นเราอาจประหยัดเวลาได้มาก ถ้ารู้ขนาดที่แน่นอนที่กระบวนการที่ต้องใช้หน่วยความจำหลัก การเรียกใช้d กระบวนการเราก็จะสับเปลี่ยนกระบวนการเข้าหรือออกเฉพาะส่วนที่เป็นจริงเท่านั้น กระบวนการผู้ใช้จึงควรให้ข้อมูล การใช้หน่วยความจำหลักกับระบบปฏิบัติการอย่างถูกต้อง และตลอดเวลาการทำงานโดยเฉพาะการจัดการหน่วยความจำแบบสัมพัทธ์ (Dynamic) กระบวนการผู้ใช้จะต้องใช้คำสั่งเรียกระบบร้องขอพื้นที่และส่งคืนพื้นที่ เพื่อที่จะบอกระบบปฏิบัติการให้ทราบถึงปริมาณการใช้หน่วยความจำที่เปลี่ยนไป

นอกจากนี้ในการสับเปลี่ยนงานเราต้องแน่ใจว่ากระบวนการ ไม่ได้มีอะไรติดค้างอยู่โดยเฉพาะอย่างยิ่งการรอคอยอุปกรณ์รับส่งข้อมูล ถ้ากระบวนการหนึ่งต้องรออุปกรณ์รับส่งข้อมูล เราอาจต้องการสับเปลี่ยนกระบวนการนี้ออกไปคอยในหน่วยเก็บโปรแกรมชั่วคราว แต่ขณะเดียวกันอุปกรณ์รับส่งข้อมูลอาจต้องใช้ที่พักข้อมูล ในเนื้อที่ของกระบวนการผู้ใช้ เพื่อรับหรือส่งข้อมูลให้แก่ผู้ใช้ ถ้ากระบวนการเดิมถูกสับเปลี่ยนออกไปและมีกระบวนการใหม่เข้ามาแทนที่อุปกรณ์รับส่งข้อมูลอาจส่งข้อมูลไปผิดกระบวนการได้ ปัญหานี้อาจแก้ไขได้ 2 วิธี คือ

1. ห้ามสับเปลี่ยนงานที่รอคอยการรับส่งข้อมูล (Process Wait I/O)

2. การรีบส่งข้อมูล ต้องใช้ที่พักข้อมูลภายในเนื้อที่ของระบบปฏิบัติการเท่านั้นและระบบจะส่งต่อให้กระบวนการ เมื่อกระบวนการกำลังทำงานอยู่ในหน่วยความจำหลักเท่านั้น

การสับเปลี่ยนกระบวนการ นี้ มีในระบบยูนิกซ์ในยุคแรก ๆ ถ้าระบบทำงานปกติจะไม่มีการสับเปลี่ยนเกิดขึ้น ระบบจะเริ่มมีการสับเปลี่ยนเมื่อมีกระบวนการ ในระบบมากเกินไปจนเกิด

ปัญหา และเมื่อระบบกลับสู่ภาวะปกติการสับเปลี่ยนงานก็จะหยุดไปด้วยเช่นกัน ปัจจุบันมีระบบปฏิบัติการน้อยมากที่ใช้วิธีนี้

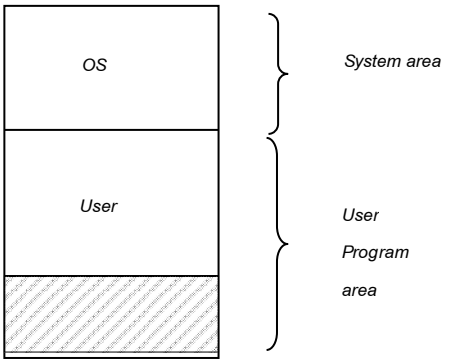
การจัดสรรเนื้อที่แบบเดี่ยว

วิธีการจัดการหน่วยความจำหลักที่ง่ายที่สุดคือไม่ต้องทำอะไรเลยผู้ใช้จัดการหน่วยความจำหลักเอง โดยสามารถใช้ส่วนใด ๆ ในหน่วยความจำก็ได้

วิธีนี้มีข้อดีคือ ผู้ใช้มีความยืดหยุ่นสูง สามารถจัดการหน่วยความจำหลักได้ตามใจ ค่าใช้จ่ายในการจัดการก็ไม่มี ไม่ต้องมีอุปกรณ์พิเศษช่วย ต้องมีโปรแกรมระบบช่วยจัดการ

แต่ข้อเสีย หรือข้อจำกัดของวิธีนี้คือ ระบบไม่ได้ช่วยอะไรผู้ใช้เลย ผู้ใช้สามารถควบคุมคอมพิวเตอร์ได้ทั้งหมด แต่ระบบปฏิบัติการไม่สามารถควบคุมสัญญาณจัดจังหวะได้ ไม่มีกลไกในการจัดการคำสั่งเรียกระบบ หรือข้อผิดพลาดที่อาจเกิดขึ้น ไม่มีเนื้อที่ในการทำงานแบบหลายโปรแกรม วิธีนี้จะใช้กับระบบที่เป็นการทำงานเฉพาะอย่าง โดยผู้ใช้ต้องการความยืดหยุ่นสูงที่สุดและตั้งใจที่จะเขียนโปรแกรมสนับสนุนต่าง ๆ เองทั้งหมด

วิธีที่สูงขึ้นกว่า วิธีแรกอีกขั้นหนึ่ง คือ การแบ่งหน่วยความจำหลักออกเป็น 2 ส่วน ส่วนหนึ่งสำหรับผู้ใช้อีกส่วนหนึ่งสำหรับโปรแกรมของระบบปฏิบัติการเอง ซึ่งอาจจะวางตัวอยู่ในตำแหน่งล่างสุด (0000) หรือ ตำแหน่งบนสุด (FFFF) ของหน่วยความจำหลัก แต่โดยปกติแล้วตารางสัญญาณการขัดจังหวะ (Interrupt Vector) จะอยู่ในตำแหน่งล่างสุด ของหน่วยความจำ ดังนั้นจึงเป็นการสะดวกและปลอดภัยกว่า ที่จะให้ระบบปฏิบัติการอยู่ในตำแหน่งล่างนี้ด้วย (ดูรูปที่ 7.4) ในหัวข้อนี้จึงจะอธิบายโดยเลือกให้ระบบปฏิบัติการอยู่ในตำแหน่งล่าง (การเลือกเป็นตำแหน่งล่างการเลือกเป็นตำแหน่งบน ก็สามารถทำได้คล้าย ๆ กัน)



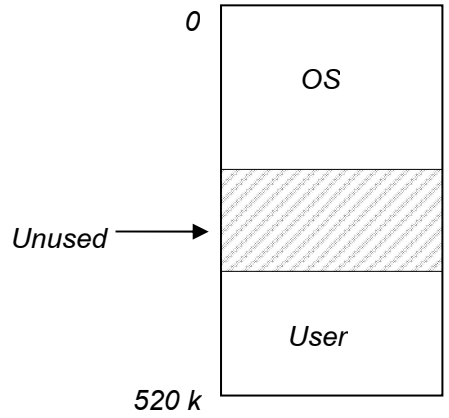
รูปที่ 7.4 แสดงการจัดสรรเนื้อที่แบบเดี่ยว

ถ้าระบบปฏิบัติการอยู่ในหน่วยความจำต่ำ (Low Memory) และผู้ใช้อยู่ในหน่วยความจำสูง (High Memory) ในรูปวาดกลับหัวกลับหางตามความนิยมคือ เอาตำแหน่งล่าง (0) ไว้บนและเอา

ตำแหน่งบนไว้ล่าง ระบบต้องมีการป้องกันโปรแกรมและข้อมูลของระบบจากผู้ใช้ที่อาจเข้ามาแก้ไข ในส่วนของระบบโดยใช้อุปกรณ์พิเศษช่วย ซึ่งอาจเป็นรีจีสเตอร์ฐาน (Base Register) หรือรีจิสเตอร์ขอบเขต (Limit Register) แต่เนื่องจากวิธีนี้เรามีผู้ใช้เพียงคนเดียวจึงไม่จำเป็นต้องใช้รีจีสเตอร์ขอบเขต

ปัญหาอีกข้อหนึ่งคือ การนำโปรแกรมผู้ใช้ลงในหน่วยความจำเพราะเนื้อที่ของผู้ใช้ไม่ได้เริ่มต้นที่ตำแหน่ง 0000 แต่เริ่มจากตำแหน่งตามค่าในรีจิสเตอร์ฐานถ้าเราทราบค่ารีจิสเตอร์นี้ในช่วงการแปลโปรแกรม เราก็สามารถแปลค่าตำแหน่งเป็นแบบสัมบูรณ์ได้โดยเริ่มจากค่านี้เป็นต้นไป ถ้าค่ารีจีสเตอร์ฐานนี้เปลี่ยนไปเราก็ต้องแปลโปรแกรมใหม่ หรืออีกวิธีหนึ่งคือ แปลค่าตำแหน่ง (Address) เป็นแบบย้ายได้ (Relative) และให้ตัวนำโปรแกรมลงสู่หน่วยความจำ เป็นตัวกำหนดค่าตำแหน่งจริงอีกที ถ้าค่ารีจีสเตอร์ฐานเปลี่ยนไปเราเพียงแต่นำโปรแกรมลงหน่วยความจำใหม่อีกครั้งไม่ต้องแปลใหม่

วิธีที่กล่าวมานี้ค่ารีจีสเตอร์ฐานจะต้องคงที่ ในระหว่างที่กระบวนการ ทำงานเพราะถ้ามีการเปลี่ยนแปลงแล้วค่าตำแหน่งต่าง ๆ ในโปรแกรมอาจใช้ไม่ได้ ดังนั้นเราจะเปลี่ยนรีจิสเตอร์ฐานนี้ได้ก็ต่อเมื่อ ไม่มีกระบวนการ ใด ๆ กำลังทำงานอยู่ ในบางกรณีอาจต้องเปลี่ยนแปลงขนาดของระบบปฏิบัติการระหว่างที่ทำงานอยู่ (ค่ารีจีสเตอร์ฐานก็ต้องเปลี่ยนตามไปด้วย) เช่น ระบบปฏิบัติการ มีโปรแกรมของตัวควบคุมอุปกรณ์และทีพักข้อมูล ซึ่งไม่ค่อยได้ใช้ จึงอาจแยกส่วนนี้ออกไปเก็บไว้ในหน่วยความจำช่วยได้ โดยไม่ต้องนำลงหน่วยความจำหลักพร้อมโปรแกรมหลักของระบบปฏิบัติการเพื่อให้มีพื้นที่สำหรับผู้ใช้มากที่สุด ส่วนที่แยกไว้นี้เราเรียกว่าส่วนผันแปร (Transient) ซึ่งจะถูกนำเข้ามาในหน่วยความจำเมื่อต้องการเท่านั้น ทำให้ขนาดของระบบมีการเปลี่ยนแปลงขณะทำงาน

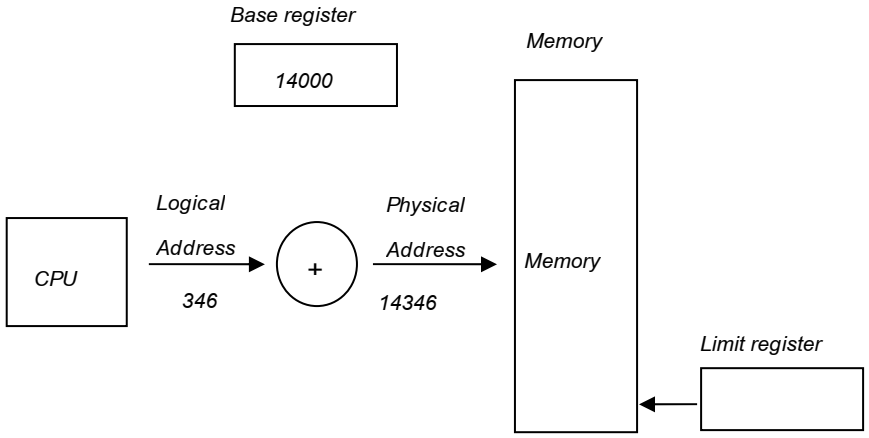


รูปที่ 7.5 การนำกระบวนการของผู้ใช้ลงในหน่วยความจำหลักข้างสูง

มีวิธีแก้ปัญหานี้ 2 วิธี คือ วิธีแรกนำโปรแกรมของผู้ใช้ลงในหน่วยความเริ่มจากข้างสูงมาหาข้างต่ำจนสุดที่ค่าในรีจิสเตอร์ฐานดังรูปที่ 7.5 เป็นการวางกลับกันกับรูปที่ 7.4

โดยวิธีนี้เนื้อที่ว่างตรงกลางระหว่างโปรแกรมผู้ใช้และระบบปฏิบัติการสามารถใช้ได้ทั้งเมื่อระบบต้องการใช้เนื้อที่เพิ่ม หรือผู้ใช้ต้องการเนื้อที่เพิ่ม

อีกวิธีหนึ่ง คือ หน่วงเวลากำหนดตำแหน่งจริง จนกระทั่งถึงเวลาทำงานจริง แต่วิธีนี้ต้องใช้อุปกรณ์ช่วยหรือฮาร์ดแวร์เสริม ดังรูปที่ 7.6 รีจีสเตอร์ฐานกลายเป็นรีจิสเตอร์สำหรับการย้ายตำแหน่ง การอ้างอิงตำแหน่งในหน่วยความจำทุกครั้งต้องนำค่าอ้างอิงมาบวกกับค่ารีจีสเตอร์ฐานเสียก่อนที่ให้ได้ค่าตำแหน่งจริง เช่น ค่ารีจีสเตอร์ฐานเท่ากับ 14000 ผู้ใช้ต้องการอ่านค่าจากตำแหน่ง 346 ก็จะไปอ่านจากตำแหน่งจริงที่ 14000 + 346 = 14346 เป็นต้น



รูปที่ 7.6 การย้ายตำแหน่งแบบสัมพันธ์ โดยการใช้รีจิสเตอร์ช่วย

จะสังเกตได้ว่า โปรแกรมของผู้ใช้ ไม่ทราบค่าตำแหน่งที่แท้จริงทางกายภาพ โปรแกรมอาจสร้างตัวชี้ ชี้ไปยังตำแหน่ง 346 และทำการคำนวณเก็บค่า อ่านค่า หรือเปรียบเทียบค่า ในตำแหน่งอื่น ๆ กับค่าในตำแหน่ง 346 นี้ โปรแกรมของผู้ใช้ทำงานด้วย ตำแหน่งทางตรรกะ หรือตำแหน่งเทียม ฮาร์ดแวร์ของเครื่อง เป็นผู้จัดการการจับคู่ตำแหน่งทางตรรกะนี้กับตำแหน่งจริงเมื่อมีการอ้างอิงตำแหน่งในหน่วยความจำ การกำหนดตำแหน่งขณะทำงาน หรือแบบสัมพัทธ์

เมื่อมีการเปลี่ยนแปลงค่าตำแหน่งเริ่มต้น ระบบก็เพียงแต่เปลี่ยนค่าในรีจีสเตอร์ฐานไปเป็นค่าเริ่มต้นตัวใหม่ และย้ายโปรแกรมผู้ใช้ทั้งหมดจากตำแหน่งเดิมไปสู่ตำแหน่งใหม่ การทำเช่นนี้อาจต้องเสียเวลา ในการคัดลอกโปรแกรมจากที่หนึ่งไปยังอีกที่หนึ่ง แต่ทำให้สามารถเปลี่ยนแปลงค่าตำแหน่งเริ่มต้นของโปรแกรมได้ตลอดเวลา

จะสังเกตเห็นว่า ในระบบนี้มีการกำหนดตำแหน่ง 2 แบบ คือ ตำแหน่งทางตรรกะ (Logical Address) ค่า 0 ถึง max และตำแหน่งจริงทางกายภาพ ค่าจาก R ถึง R+max โดย R เป็นค่าตำแหน่งเริ่มต้น โปรแกรมของผู้ใช้จะใช้ตำแหน่งทางตรรกะเท่านั้น และคิดว่ากระบวนการทำงานอยู่ในตำแหน่ง 0 ถึง max ส่วนระบบปฏิบัติการจะรู้มากกว่าและสามารถอ้างอิงตำแหน่งจริงโดยตรง ได้ในช่วงผู้ควบคุม (Monitor Mode) ข้อมูลต่าง ๆ ที่โปรแกรมผู้ใช้ส่งให้ระบบปฏิบัติการ เช่น ที่พักข้อมูลตัวแปรในการเรียกระบบ จะต้องมีการย้ายตำแหน่ง หรือคำนวณหาค่าตำแหน่งจริงให้ถูกต้อง โดยเฉพาะอย่างยิ่งการเรียนใช้อุปกรณ์รับส่งข้อมูล เพราะโปรแกรมผู้ใช้ให้ค่าตำแหน่งทางตรรกะมา ระบบต้องคู่หาตำแหน่งจริงก่อนนำไปใช้งาน

แนวคิดเรื่องการใช้ตำแหน่งทางตรรกะในการอ้างอิงแทนตำแหน่งจริง นี้เป็นหัวใจของการจัดการหน่วยความจำหลัก

การจัดสรรเนื้อที่แบบหลายส่วน

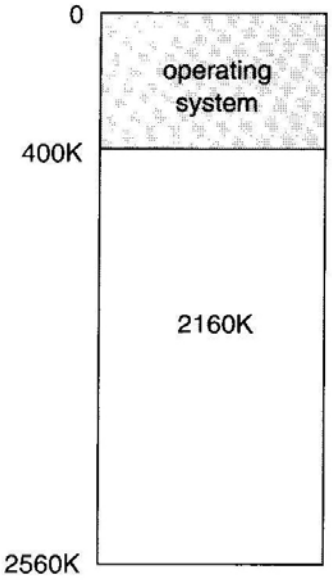
ในระบบการทำงานแบบหลายโปรแกรมนั้นจะมีกระบวนการต่าง ๆ มากมายเข้ามาทำงานในระบบ (โดยถูกบรรจุในหน่วยความจำหลัก) แต่ละกระบวนการจะผลัดกันใช้หน่วยประมวลผลกลาง ตามวิธีการจัดตารางการทำงานของระบบ ปัญหาหลักในการจัดการหน่วยความจำหลัก คือ การจัดสรรพื้นที่ให้กระบวนการต่าง ๆ ที่รออยู่ในแถวคอยขาเข้า ได้เข้าทำงานในหน่วยความจำหลัก

วิธีที่ง่ายที่สุด วิธีหนึ่ง คือ การแบ่งหน่วยความจำหลักออกเป็นส่วนย่อย ๆ ที่มีขนาดคงที่ โดยแต่ละส่วนย่อยนั้นจะเก็บกระบวนได้เพียง 1 กระบวนการ ด้วยเหตุนี้ทำให้จำนวนของกระบวนการ ที่ทำงานในระบบนี้ถูกจำกัดโดยจำนวนพื้นที่ย่อย เมื่อกระบวนการทำงานเสร็จแล้ว กระบวนการคืนพื้นที่ส่วนย่อย ที่ครอบครองอยู่นี้แก่ระบบ ระบบก็จะนำไปจัดสรรให้กระบวนการอื่นที่รออยู่ในแถวคอยขาเข้าต่อไป วิธีการนี้ถูกนำมาใช้ครั้งแรกในระบบปฏิบัติการ os/360 ของ IBM เรียกว่า MFT แต่ในปัจจุบันวิธีนี้ถูกยกเลิกไปแล้ว สำหรับวิธีการต่าง ๆ ที่จะกล่าวต่อไปนี้เป็นเพียงการแสดงให้เห็นภาพรวม ของวิธีการแบ่งหน่วยความจำหลัก อกเป็นขนาดคงที่ที่ใช้ในระบบการทำงานแบบกลุ่ม เป็นหลัก

แนวคิดเบื้องต้น

ในระบบปฏิบัติการจะมีตารางที่เก็บข้อมูลว่า พื้นที่ในหน่วยความจำหลักส่วนใดถูกครอบครอง และส่วนใดยังว่างอยู่ เริ่มต้นพื้นที่ในหน่วยความจำจะว่างทั้งหมด เมื่อมีกระบวนการเข้ามาในระบบ ระบบจะจัดการหาพื้นที่ที่ใหญ่พอสำหรับกระบวนการนั้น และจัดแบ่งพื้นที่ให้กระบวนการ เพียงเท่าที่ต้องการ ส่วนพื้นที่ที่เหลืออยู่ก็จะเก็บไว้ใช้ในการจัดสรรครั้งต่อไป

ตัวอย่างเช่น ระบบมีหน่วยความจำหลักขนาด 2560 K มีโปรแกรมของระบบใช้พื้นที่อยู่ 400 K ส่วนว่างที่เหลืออีก 2160 K สำหรับผู้ใช้สมมุติให้ในแถวคอยขาเข้ามีกระบวนการ ดังรูปที่ 7.7 ระบบใช้วิธีจัดตารางการทำงานระยะยาว (Long-Term Scheduling) แบบมาก่อน-ได้ก่อน (First-in First-out) ดังนั้นเราสามารถ



job queue

process	memory	time
P_1	600K	10
P_2	1000K	5
P_3	300K	20
P_4	700K	8
P_5	500K	15

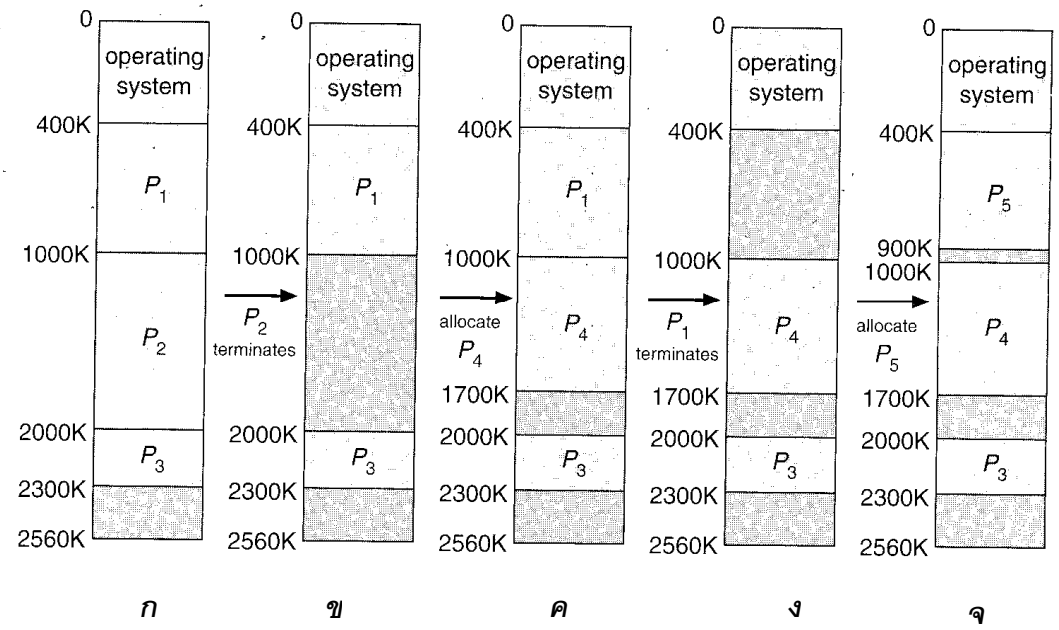
รูปที่ 7.7 ตัวอย่างการจัดตารางการทำงาน

จัดสรรพื้นที่ให้แก่ กระบวนการ P1 P2 P3 และ P4 ได้ทันที ดังแผนภาพของหน่วยความจำในรูปที่ 7.8 (ก) จะเห็นว่า มีพื้นที่ว่างขนาด 260 K ซึ่งไม่เพียงพอกับขนาดของกระบวนการ P4 ที่รออยู่ในแถวคอยขาเข้า ถ้าระบบใช้วิธีจัดตารางการทำงานระยะสั้นแบบเวียนเทียน โดยส่วนแบ่งเวลามีค่าเท่ากับ 1 หน่วยเวลา ณ เวลาที่ 14 หน่วยเวลา กระบวนการ P2 จะทำงานเสร็จพื้นที่ ที่ P2 ครอบครองอยู่จะว่างลง กระบวนการ P4 จึงสามารถทำงานได้ (ดังแสดงในรูปที่7.8 (ค)) และเมื่อกระบวนการ P1 ทำงานเสร็จ ณ หน่วยเวลาที่ 28 (ดังแสดงในรูปที่ 7.8 (ง)) กระบวนการ P5 ก็สามารถเข้ามาใช้พื้นที่ว่างลงต่อไปได้

จากตัวอย่างข้างต้น จะพบว่ามีกลุ่มของพื้นที่ว่างในหน่วยความจำ ซึ่งแต่ละพื้นที่จะมีขนาดไม่เท่ากัน กระจายไปทั่วทั้งหน่วยความจำ ถ้ามีกระบวนการหนึ่งเข้ามาในระบบ ระบบก็จะตรวจหาพื้นที่ว่างที่มีขนาดเพียงพอกับกระบวนการนั้น ถ้าพื้นที่ว่างมีขนาดใหญ่มากเกินไป พื้นที่ว่างนั้นก็จะถูกแบ่งออกเป็น 2 ส่วน โดยส่วนหนึ่ง เป็นพื้นที่สำหรับกระบวนการนั้น และอีกส่วนหนึ่ง จะเป็นพื้นที่ว่าที่เหลือ ซึ่งพื้นที่ว่างนี้จะถูกนำไปรวมไว้ในกลุ่มพื้นที่ว่างของหน่วยความจำ เมื่อ

กระบวนการใดทำงานเสร็จแล้ว กระบวนการนั้นจะปล่อยพื้นที่ที่ถือครองอยู่กลับสู่ระบบ ถ้าพื้นที่ที่คืนมาต่อเนื่องกับพื้นที่ว่างเดิมระบบก็จะรวมเป็นพื้นที่ว่างผืนเดียวกัน และกลับไปตรวจดูว่าพื้นที่ว่างใหม่มีขนาดเพียงพอ กับกระบวนการ ที่รอคอยอยู่หรือไม่

ปัญหาข้างต้น เป็นปัญหาในระบบที่มีการจัดสรรพื้นที่แบบยืดหยุ่น (Dynamic Storage Allocation) ระบบจะพยายามจัดหาพื้นที่ว่างตามที่ผู้ใช้ร้องขอ (ขนาด n ไบท์) โดยตรวจดูจากกลุ่มพื้นที่ว่างในหน่วยความจำหลักมีวิธีการจัดการพื้นที่ว่างนี้หลายวิธีแต่ละวิธี พยายามให้



รูปที่ 7.8 การจัดสรรหน่วยความจำหลักและการจัดตารางการทำงานระยะยาว

การจัดสรรพื้นที่มีประโยชน์สูงสุด วิธีที่นิยม ได้แก่ วิธีแรกเหมาะสม (First-Fit) วิธีเหมาะสมที่สุด (Best-Fit) และวิธีแย่สุด (Worst-Fit)

วิธีแรกเหมาะสม (First-Fit) คือ เลือกพื้นที่แรกที่พบว่ามีขนาดใหญ่กว่า หรือเท่ากับพื้นที่ที่ต้องการ โดยการค้นหาพื้นที่ว่างนี้ อาจเริ่มค้นหาตั้งแต่จุดเริ่มต้นของกลุ่มพื้นที่ว่าง หรือเริ่มค้นต่อจากจุดที่หยุดลงจากการค้นหาครั้งก่อน ก็ได้

วิธีเหมาะสมที่สุด (Best-Fit) คือ เลือกพื้นที่ที่มีขนาดใกล้เคียงกับขนาดพื้นที่ที่ต้องการมากที่สุด (ทำให้เกิดช่องว่างใหม่เล็กที่สุด การค้นหาพื้นที่ว่างแบบนี้ จะต้องตรวจดูพื้นที่ที่ว่างทั้งหมดในระบบ ถ้าได้มีการเรียงพื้นที่ว่างไว้ ตามลำดับขนาดแล้ว การค้นหาก็จะเร็วขึ้น วิธีนี้จะทำให้พื้นที่ว่างที่เกิดขึ้นใหม่มีขนาดเล็กที่สุด

วิธีแย่สุด (Worst-Fit) คือ เลือกพื้นที่ที่มีขนาดใหญ่กว่าพื้นที่ที่ต้องการมากที่สุด (ทำให้เกิดช่องว่างใหม่ใหญ่ที่สุด) กาค้นหาก็เหมือนกับแบบเหมาะสมที่สุดคือ จะต้องตรวจดูพื้นที่ที่ว่างทั้งหมดในระบบนอกจาจะมีการเรียงไว้ วิธีนี้มีแนวคิดว่าการเลือกจะทำให้เกิดพื้นที่ว่างใหม่ ที่มีขนาดใหญ่ที่สุด อาจนำไปใช้ต่อไปได้มากกว่าวิธีเหมาะสมที่สุด ซึ่งทำให้พื้นที่ว่างที่เกิดขึ้นใหม่มีขนาดเล็กที่สุด

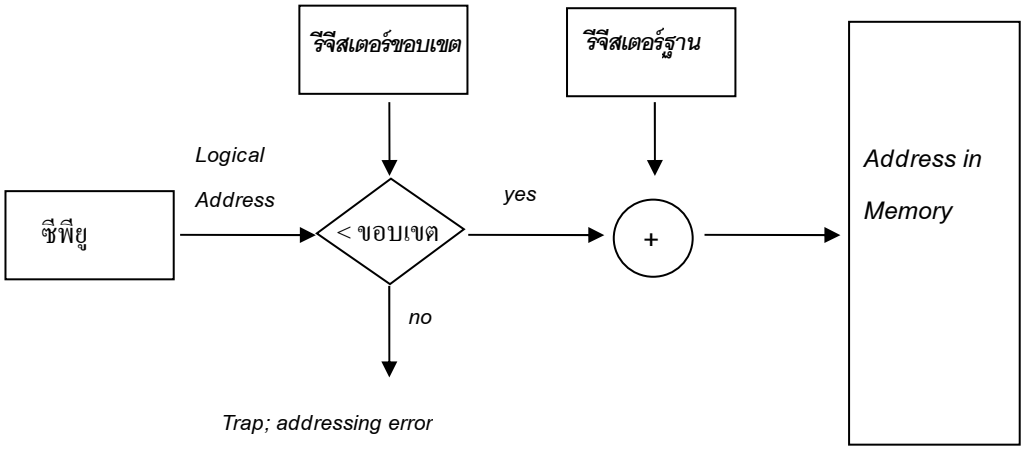
มีการสร้างแบบจำลองระบบเพื่อทดสอบวิธีทั้ง 3 ที่กล่าวมาแล้ว ผลปรากฏว่าวิธีแรกเหมาะและวิธีเหมาะสมที่สุดดีกว่าวิธีแย่สุด ในแง่ของเวลาที่ลดลงและประสิทธิผลในการใช้ที่เก็บข้อมูล แต่ก็มิได้หมายความว่าทั้งวิธีแรกเหมาะ และวิธีเหมาะสมที่สุด เป็นวิธีการที่มีประสิทธิภาพดีที่สุด โดยทั่วไปแล้ววิธีแรกเหมาะใช้เวลาน้อยกว่าแบบอื่น ๆ

การใช้ขั้นตอนวิธี ในการจัดสรรพื้นที่หน่วยความจำทั้ง 3 วิธีนี้จะมีปัญหาที่เกิดตามมา คือ พื้นที่หน่วยความจำ ซึ่งว่างเป็นช่วง ๆ มีขนาดเล็กสำหรับงานที่รอคอยอยู่ หรือที่เรียกว่า การสูญเปล่าพื้นที่ย่อยภายนอก (External Fragmentation) นั่นหมายความว่าในขณะที่กระบวนการถูกบรรจุลงหน่วยความจำ และถูกนำออกไปจากระบบ หน่วยความจำจะถูกแบ่งเป็นส่วนย่อยเล็ก ๆ หลายส่วน ทำให้ระบบไม่สามารถบรรจุกระบวนการใหม่ได้อีกทั้ง ๆ ที่พื้นที่ว่างรวมของระบบ (ในหน่วยความจำ)มีขนาดใหญ่พอ เนื่องจากพื้นที่ที่ว่างเหล่านั้นมิได้อยู่ติดต่อกันเป็นผืนเดียว แต่กลับอยู่กระจัดกระจายไปทั่ว ๆ ในหน่วยความจำ

การสูญเปล่าพื้นที่ย่อยภายนอก (External Fragmentation) นี้ จะเป็นปัญหามากหรือน้อยนั้นอยู่กับขนาดของพื้นที่หน่วยความจำทั้งหมดกับขนาดของกระบวนการที่เข้ามาในระบบ จากการศึกษาทางสถิติโดยใช้วิธีแรกเหมาะ พบว่าแม้ว่าจะมีการปรับปรุงขั้นตอนวิธีบางอย่างช่วยแล้ว ถ้าจัดสรรพื้นที่ไป n หน่วยแล้ว มักจะเกิดพื้นที่ว่าง 0.5 n หน่วย เนื่องจากถูกแบ่งเป็นส่วนย่อย ๆ หรือ คิดเป็นประมาณ หนึ่งในสามของหน่วยความจำหลักที่สูญเปล่าไป ซึ่งมีชื่อเรียกว่า “กฎร้อยละห้าสิบ” (50 percent law)

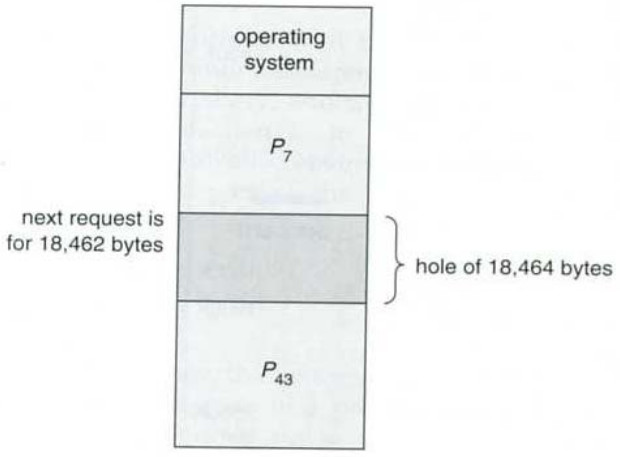
เมื่อระบบได้จัดสรรพื้นที่ว่างในหน่วยความจำให้กับกระบวนการ แล้วกระบวนการนั้นก็จะถูกบรรจุลงสู่ตำแหน่งที่จัดไว้ให้และเริ่มทำงาน เนื่องจากมีหลายกระบวนการอยู่ในหน่วยความจำพร้อม ๆ กัน เราจึงต้องมีการป้องกันกระบวนการต่าง ๆ ไม่ให้ออกนอกเขตของตน โดยใช้รีจิสเตอร์ฐาน (Base Register) และรีจีสเตอร์ขอบเขต (Limit Register) ซึ่งเป็นการกำหนดตำแหน่งแบบสัมพัทธ์ สามารถย้ายตำแหน่งของโปรแกรมได้ ในขณะทำงาน ตำแหน่งทางตรรกะจะต้องมีค่าน้อยกว่าค่าในรีจีสเตอร์ขอบเขต (Limit Register) ก่อนที่จะนำไปบอกกับค่าในรีจีสเตอร์ฐาน เพื่อหาค่าตำแหน่งจริงต่อไป ดังรูปที่ 7.9

เมื่อตัวจัดตารางการทำงานของหน่วยประมวลผล จัดให้กระบวนการหนึ่งทำงาน ตัวควบคุมก็จะเขียนค่าฐานและขอบเขต สำหรับกระบวนการนี้ลงในรีจีสเตอร์ เนื่องจากการอ่านหรือเขียนข้อมูลในตำแหน่งต่าง ๆ ต้องผ่านการตรวจสอบจากฮาร์ดแวร์ โดยใช้ค่าในรีจีสเตอร์ฐานและรีจีสเตอร์ขอบเขต เราจึงสามารถป้องกันกระบวนการออกนอกขอบเขต ของตนเองได้



รูปที่ 7.9 แสดงการใช้รีจิสเตอร์ฐานและรีจิสเตอร์ขอบเขตจัดการหน่วยความจำ

รูปที่ 7.10 แสดงปัญหาอีกข้อหนึ่ง จะเห็นว่า ถ้าเรามีพื้นที่ว่าง ขนาด 18,464 ไบท์ และกระบวนการ ใหม่ต้องการ 18,462 ไบท์ แล้วเราจัดสรรเนื้อที่ให้พอดี 18,462 ไบท์ ก็จะเหลือพื้นที่



รูปที่ 7.10 ตัวอย่างการจัดสรรหน่วยความจำหลัก

ว่าง 2 ไบท์ การเก็บข้อมูลพื้นที่ว่าง 2 ไบท์นี้ ต้องเสียค่าใช้จ่ายสูงกว่าเนื้อที่ 2 ไบท์มาก ดังนั้นเรามักจัดสรรพื้นที่ว่างที่เหลือเล็กน้อยนี้ ให้เป็นส่วนหนึ่งของพื้นที่ที่ร้องขอไปเลย ซึ่งพื้นที่ส่วนนี้ก็จะสูญเปล่าไป เช่นกัน เราเรียนกว่า “การสูญเปล่าพื้นที่ย่อยภายใน” (Internal Fragmentation)

การจัดการทำงานระยะยาว

เมื่อกระบวนการต่าง ๆ เข้าสู่ระบบกระบวนการ เหล่านั้น จะถูกนำไปเก็บในแถวคอยขาเข้า (Input Queue) จากนั้น ตัวจัดตารางการทำงานระยะยาว (Long-term Scheduler) ก็จะเป็นผู้คำนวณหรือตรวจนับปริมาณพื้นที่ ในหน่วยความจำที่แต่ละกระบวนการ ต้องการ และดูว่า ในหน่วยความจำจะมีพื้นที่ว่างขนาดเท่าไร เพื่อที่จะเลือกว่ากระบวนการ ใดจะเป็นผู้ที่ได้รับเลือกให้เข้าใช้พื้นที่ว่างนั้น และหลังจากที่กระบวนการที่ได้รับเลือก ถูกบรรจุลงในหน่วยความจำของระบบ อาจจะมีการย้ายตำแหน่งเกิดขึ้นถ้าจำเป็น กระบวนการนั้นจะต้องแย่งใช้หน่วยประมวลผล กับกระบวนการ อื่นที่มีอยู่แล้วในระบบ เมื่อกระบวนการทำงานเสร็จแล้ว ก็จะคืนหน่วยความจำที่ถือครองอยู่แก่ระบบ ซึ่งระบบก็นำไปจัดสรรให้กระบวนการอื่น ๆ ที่รออยู่ในแถวคอยขาเข้าต่อไป

ระบบจะมีข้อมูลเกี่ยวกับขนาดของพื้นที่ว่างแต่ละช่วงในหน่วยความจำ และข้อมูลทางด้านความต้องการพื้นที่หน่วยความจำของแต่ละกระบวนการที่อยู่ในแถวคอยขาเข้า ดังนั้นตัวจัดตารางระยะยาว สามารถเรียงลำดับกระบวนการในแถวคอยขาเข้าให้สอดคล้องกับขั้นตอนวิธี ในการจัดตารางการทำงานได้ โดยพื้นที่ในหน่วยความจำจะถูกจัดสรรให้แก่กระบวนการต่าง ๆ ไปเรื่อย ๆ จนกระทั่งไม่สามารถจัดสรรต่อไปได้ เนื่องจากขนาดของพื้นที่ว่างในหน่วยความจำหลัก มีขนาดเล็กกว่าขนาดของกระบวนการที่อยู่หัวแถวคอย เมื่อเป็นเช่นนี้ตัวจัดตาราง จะรอจนกว่ามีบางกระบวนการในระบบปล่อยพื้นที่ที่ถือครองอยู่คืนแก่ระบบ หรือระบบอาจจะข้ามไปเลือกกระบวนการที่มีขนาดเล็กกว่าแต่ไม่ได้อยู่ที่หัวแถวคอย เพื่อที่จะสามารถใช้พื้นที่ว่างเท่าที่มีอยู่ตอนนั้นได้ การที่ระบบข้ามไปเลือกกระบวนการหลัง ๆ ให้เข้ามาทำงานนั้น ระบบต้องเลือกกระบวนการที่มีระดับความสำคัญต่ำลงกว่าหัวแถว แต่มีขนาดเหมาะสมเข้ามา ปัญหานี้ทำให้ตัวจัดตารางการทำงานมีทางเลือกระหว่างการข้ามหรือไม่ข้ามไปเลือกกระบวนการที่มีระดับความสำคัญต่ำกว่า

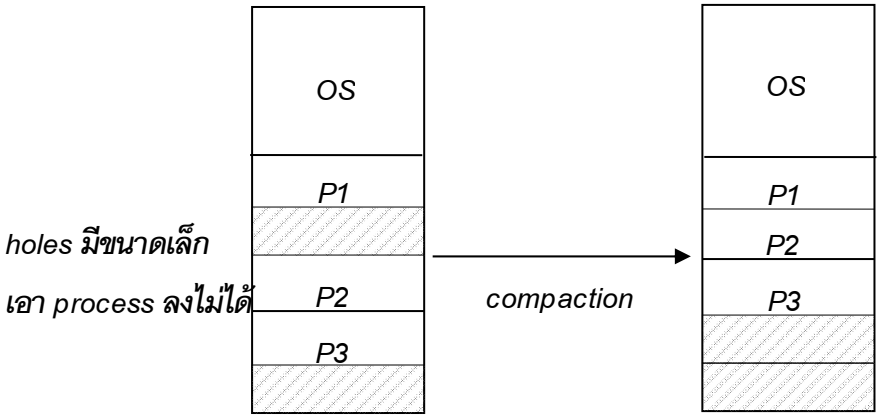
วิธีการทำงานดังกล่าวนี้ จะไม่ทำให้เกิดปัญหาการสูญเปล่าของพื้นที่ย่อยภายใน (Internal Fragmentation) เนื่องจากระบบจัดสรรพื้นที่ให้เท่าที่กระบวนการต้องการใช้ แต่จะเกิดการสูญเปล่าพื้นที่ย่อยภายนอกสถานะแรก คือรูปที่ 7.8 (a) จะพบว่ามีพื้นที่ว่างขนาด 260K ในระบบ ซึ่งพื้นที่มีขนาดเล็กเกินกว่าที่จะบรรจุกระบวนการ P4 หรือ P5 ได้ แต่อย่างไรก็ตามเมื่อระบบเข้าสู่สถานะที่ 7.8 (c) จะมีพื้นที่ย่อยสูญเปล่าภายนอกถึง 560K (=360 + 260) ซึ่งมีขนาดเพียงพอสำหรับ

กระบวนการ P5 แต่ระบบไม่สามารถนำกระบวนการ P5 บรรจุลงได้ เนื่องจากพื้นที่ขนาด 560K ดังกล่าว มิได้เป็นพื้นที่ว่างต่อเนื่องกัน แต่กลับเป็นพื้นที่ว่าง 2 พื้นที่ แยกกันคือ 300 K และ 260 K ซึ่งแต่ละพื้นที่มีขนาดเล็กกว่ากระบวนการ P5

การสูญเปล่าพื้นที่ย่อยภายนอก (External Fragmentation) หรือการที่พื้นที่ถูกแบ่งเป็นส่วนย่อย ๆ นี้ อาจเป็นปัญหาใหญ่ของระบบได้ ในกรณีที่แย่ที่สุดคือ มีพื้นที่ว่างซึ่งสูญเปล่าระหว่างกระบวนการที่ติดกันทุก ๆ คู่ก็ได้ ถ้าเราสามารถทำให้พื้นที่ว่างเปล่านี้ต่อเนื่องกัน ก็อาจใช้ใส่กระบวนการได้อีกมาก ขั้นตอนวิธีแบบแรกเหมาะ หรือแบบเหมาะสมที่สุดมีผลต่อพื้นที่ย่อยที่สูญเสียไปนี้มาก (วิธีแรกเหมาะให้ผลดีกว่าในบางระบบ วิธีเหมาะสมที่สุดก็เช่นกัน คือ ให้ผลดีกว่าในบางระบบ) อีกปัจจัยหนึ่งก็คือ เมื่อมีพื้นที่ว่างเราจะจัดสรรพื้นที่ใหม่ให้ติดด้านบน หรือติดด้านล่างของพื้นที่ว่างดี ไม่ว่าจะใช้ขั้นตอนวิธีแบบใด ปัญหาการสูญเปล่าพื้นที่ย่อยภายนอก ก็ไม่อาจหมดไปได้

การบีบอัด (Compaction)

วิธีการหนึ่งที่สามารถใช้ในการแก้ปัญหาการสูญเสียพื้นที่ย่อยภายนอกได้คือ การบีบอัด (Compaction) การบีบอัดเป็นการสับเปลี่ยนหรือโยกย้ายพื้นที่ว่างในระบบ ให้มาอยู่รวมกันเป็นพื้นที่ผืนเดียวต่อเนื่องกัน ตัวอย่างเช่น จากแผนภาพหน่วยความจำในรูปที่ 7.8 ระบบสามารถทำการบีบอัด หน่วยความจำของระบบ ให้เป็นไปดังรูปที่ 7.11 ได้จะเห็นว่า พื้นที่ว่างเล็ก ๆ 3 ผืน (คือ 100K, 300K และ 260K) ได้ถูกย้ายมารวมให้เป็นพื้นที่ผืนเดียวที่มีขนาด 660K



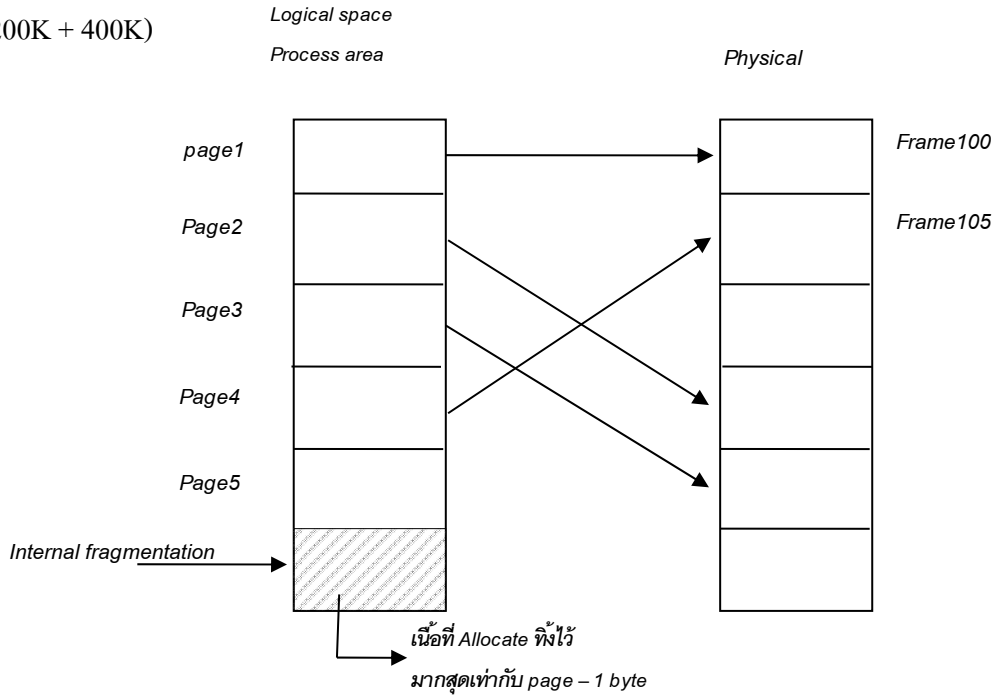
รูปที่ 7.11 การบีบอัดเนื้อที่หน่วยความจำ

การบีบอัดหน่วยความจำของระบบ ไม่อาจทำได้กับทุกระบบ สังเกตรูปที่ 7.11 ที่มีการเคลื่อนย้ายกระบวนการ P4 และ P3 กระบวนการทั้ง 2 ต้องสามารถทำงานต่อได้ เมื่อเข้ามาอยู่ในพื้นที่ โดยตำแหน่งอ้างอิงภายในทั้งหมดจะต้องถูกปรับปรุง กรณีที่การย้ายตำแหน่งเป็นแบบคงที่

และเกิดขึ้นในขณะบรรจุกระบวนการ (Load Time) เราจะไม่สามารถบีบอัดหน่วยความจำได้ เราสามารถบีบอัดหน่วยความจำ เฉพาะกรณีที่การย้ายตำแหน่งเป็นแบบสัมพัทธ์ (Dynamic) และเกิดขึ้นขณะทำงาน (Execution Time) เท่านั้น

ถ้าการย้ายตำแหน่งเป็นแบบสัมพัทธ์ เราสามารถย้ายตำแหน่งของกระบวนการได้โดยเพียงย้ายโปรแกรมและข้อมูลของกระบวนการไปยังตำแหน่งใหม่ และแก้ไขค่ารีจีสเตอร์ฐานของกระบวนการไปเป็นค่าใหม่

เมื่อการบีบอัดหน่วยความจำหลักสามารถทำได้สิ่งที่เราจะต้องพิจารณาต่อมา คือ ค่าใช้จ่ายที่เกิดขึ้น โดยขั้นตอนวิธีในการบีบอัดหน่วยความจำหลัก ที่ง่ายที่สุดคือ จัดการเคลื่อนย้ายกระบวนการ ทั้งหมดในระบบ ให้ไปอยู่สุดขอบใดขอบหนึ่งไปทางเดียวกันทั้งหมด ของหน่วยความจำหลัก ซึ่งจะเป็นผลให้เกิดพื้นที่ว่าง ขนาดใหญ่ ซึ่งเกิดจากพื้นที่ว่างย่อย ๆ รวมกัน ของหน่วยความจำหลัก จะเป็นผลให้เกิดพื้นที่ว่างขนาดใหญ่ที่เกิดจากพื้นที่ว่างย่อย ๆ รวมกัน ในทิศทางตรงกันข้าม วิธีการบีบอัดหน่วยความจำ ดังกล่าวนี้มีค่าใช้จ่ายสูงมาก พิจารณาการจัดสรรพื้นที่หน่วยความจำ ในรูปที่ 7.12 จะพบว่า ถ้าเราทำการบีบอัดหน่วยความจำ ตามขั้นตอนวิธีอย่างง่าย ดังกล่าวข้างต้น เราต้องเคลื่อนย้ายทั้งกระบวนการ P3 และ P4 ซึ่งมีขนาดโดยรวมเท่ากับ 600K (200K + 400K)



รูปที่ 7.12 การบีบอัดเนื้อที่หน่วยความจำแบบอื่น

แต่ในสถานการณ์นี้ เราอาจจะเคลื่อนย้ายเพียงกระบวนการ P4 ไปไว้บนกระบวนการ P3 ซึ่งเป็นการย้ายพื้นที่ขนาดเพียง 300K จะเห็นได้ว่า ในกรณีสุดท้ายนี้จะทำให้เกิดพื้นที่ว่างขนาดใหญ่ตรงกลางของพื้นหน่วยความจำหลักทั้งหมด แทนที่จะเป็นท้าย นอกจากนี้ถ้าเราเพิ่มการสังเกตโดยเพิ่มขั้นตอนวิธีให้ซับซ้อนขึ้น จะเห็นว่าถ้าในแถวคอยขาเข้ามีกระบวนการเหลืออยู่อีกเพียง 1 กระบวนการที่ต้องการเข้ามาทำงานในระบบ ซึ่งมีขนาดเพียง 450K เราก็อาจย้ายเพียงกระบวนการ P2 มีขนาด 100K ไปไว้ที่ใดที่หนึ่งในหน่วยความจำเช่น ใต้กระบวนการ P4 ซึ่งถึงแม้ว่าการเคลื่อนย้ายดังกล่าว จะไม่ทำให้เกิดพื้นที่ว่างขนาดใหญ่ที่เป็นผืนเดียวกัน แต่ก็สามารถตอบสนองความต้องการ ด้านพื้นที่ของกระบวนการได้ เพราะว่าได้พื้นที่ว่างที่มีขนาดใหญ่เพียงพอที่จะบรรจุกระบวนการได้ทันท่วงที จะเห็นว่าการหาวิธีการที่ดีที่สุดในการบีบอัดหน่วยความจำหลักเป็นสิ่งที่ยากมาก

นอกจากนั้น ยังสามารถรวมวิธีการสลับที่ (Swapping) กับการบีบอัดหน่วยความจำได้โดยการนำกระบวนการออกไปเก็บพักไว้ในหน่วยเก็บข้อมูลสำรองก่อน แล้วค่อยนำกลับเข้ามาในหน่วยความจำหลักอีกครั้งเมื่อถึงเวลาทำงานจริง การทำเช่นนี้จะทำให้เกิดพื้นที่ว่างซึ่งระบบสามารถนำกระบวนการอื่นเข้ามาใช้ได้ การนำกระบวนการ เดิม ที่ถูกย้ายไว้ในหน่วยความจำสำรอง กลับเข้ามาในระบบอีกครั้งนั้น อาจจะมีปัญหาหลายอย่างเกิดตามมา เช่น ถ้ากระบวนการนั้นใช้การย้ายตำแหน่งแบบคงที่ เราก็จำเป็นต้องย้ายกระบวนการ นั้น กลับเข้ามาที่ตำแหน่งเดิมในหน่วยความจำ ซึ่งอาจเป็นผลให้ต้องย้ายกระบวนการ อื่นที่ใช้พื้นที่ดังกล่าวอยู่ออกไปจากหน่วยความจำไปเก็บในหน่วยจำสำรอง หรือ ถ้ากระบวนการ นั้นใช้การย้ายตำแหน่งแบบสัมพัทธ์ กระบวนการ นั้นจะสามารถบรรจุลงในพื้นที่ตำแหน่งใหม่ได้ แต่ระบบก็ต้องจัดหาพื้นที่ว่างใหม่ให้โดยอาจต้อง ทำการบีบอัดหน่วยความจำด้วย ถ้าจำเป็น จากนั้นจึงจะนำกระบวนการ เดิมที่ถูกย้ายออกไป กลับเข้ามาใหม่ได้

วิธีการบีบอัดหน่วยความจำอีกวิธีหนึ่งคือ การย้ายกระบวนการออกไปไว้ในที่พักข้อมูลชั่วคราว แล้วย้ายกลับมาไว้ในหน่วยความจำที่ใหม่ที่ต้องการ ถ้าในระบบมีกลไกในการสับเปลี่ยนกระบวนการอยู่แล้วก็จะยิ่งสะดวกและประหยัดการเขียนโปรแกรมการบีบอัดได้มาก

การใช้รีจิสเตอร์ฐานหลายตัว (Multiple Base Registers)

ปัญหาหลัก ของการแบ่งพื้นที่เป็นขนาดไม่เท่ากันก็คือ การสูญเสียพื้นที่ย่อยภายนอกอีกทางเลือกหนึ่งในการแก้ปัญหานี้คือ การแบ่งเนื้อที่ของกระบวนการออกเป็นหลาย ๆ ส่วน โดยแต่ละส่วน อาจมีขนาดเล็กกว่าช่องว่างที่สูญเสียไปในวิธีแรก ดังนั้นเราสามารถจัดกระบวนการลงหน่วยความจำได้ง่ายขึ้น การใช้วิธีนี้ต้องมีรีจิสเตอร์ฐานหลายตัวช่วยรวมทั้งมีกลไกในการแปลงตำแหน่งทางตรรกะ มาเป็นตำแหน่งจริงที่เหมาะสมกับรีจิสเตอร์ฐานหลายตัวด้วย

วิธีแรก คือ การแบ่งหน่วยความจำหลักออกเป็น 2 ส่วนแยกจากกัน ระบบใช้รีจิสเตอร์ 2 ชุดแต่ละชุด มีรีจิสเตอร์ฐานและรีจิสเตอร์ขอบเขตการแบ่งหน่วยความจำหลักทำโดยให้บิตที่สูงสุด 1 บิต กำหนดค่าเป็น 0 สำหรับหน่วยความจำส่วนล่าง และเป็น 1 สำหรับหน่วยความจำส่วนบน โดยปกติ ตัวแปลภาษาชั้นสูง และตัวแปลภาษาแอสแซมบลี้ จะเก็บข้อมูลประเภทใช้อ่านอย่างเดียว (Read Only) เช่น ค่าคงที่ต่าง ๆ และคำสั่ง ไว้ในหน่วยความจำส่วนบน และเก็บพวกตัวแปรต่าง ๆ ไว้ในหน่วยความจำส่วนล่าง เราอาจใช้บิตป้องกัน (Protection Bits) ควบคู่ไปกับรีจิสเตอร์ทั้ง 4 ตัว เพื่อป้องกันการเขียนทับข้อมูลในหน่วยความจำส่วนบนได้การแบ่งแบบนี้ช่วยให้กระบวนการ ผู้ใช้สามารถใช้ส่วนที่เป็นโปรแกรมร่วมกันได้ โดยมีส่วนของข้อมูลแยกเป็นแต่ละกระบวนการ อยู่ในหน่วยความจำส่วนล่าง

วิธีที่สอง คือ แยกโปรแกรมหนึ่ง ๆ ออกเป็น 2 ส่วน คือส่วนของโปรแกรมและส่วนของข้อมูลขณะทำงาน หน่วยประมวลผลจะรู้เองว่า ตอนนี้เป็นการอ่านคำสั่งหรืออ่านข้อมูล ก็จะเลือกใช้รีจิสเตอร์ฐานและขอบเขต คู่ที่ใช้สำหรับส่วนนั้น ๆ ดังนั้นส่วนที่เป็นโปรแกรมจึงสามารถใช้ร่วมกันได้หลายผู้ใช้

จะเห็นได้ว่า ทั้งสองวิธีที่กล่าวมาแล้ว เราแยกส่วนโปรแกรมและข้อมูลออกจากัน ทำให้สามารถใช้ส่วนที่เป็นโปรแกรมร่วมกันได้ เป็นการประหยัดเนื้อที่ในหน่วยความจำทำให้ไม่ให้มีการซ้ำซ้อน และยังช่วยลดการสูญเปล่าพื้นที่ย่อยภายนอก ได้ด้วย

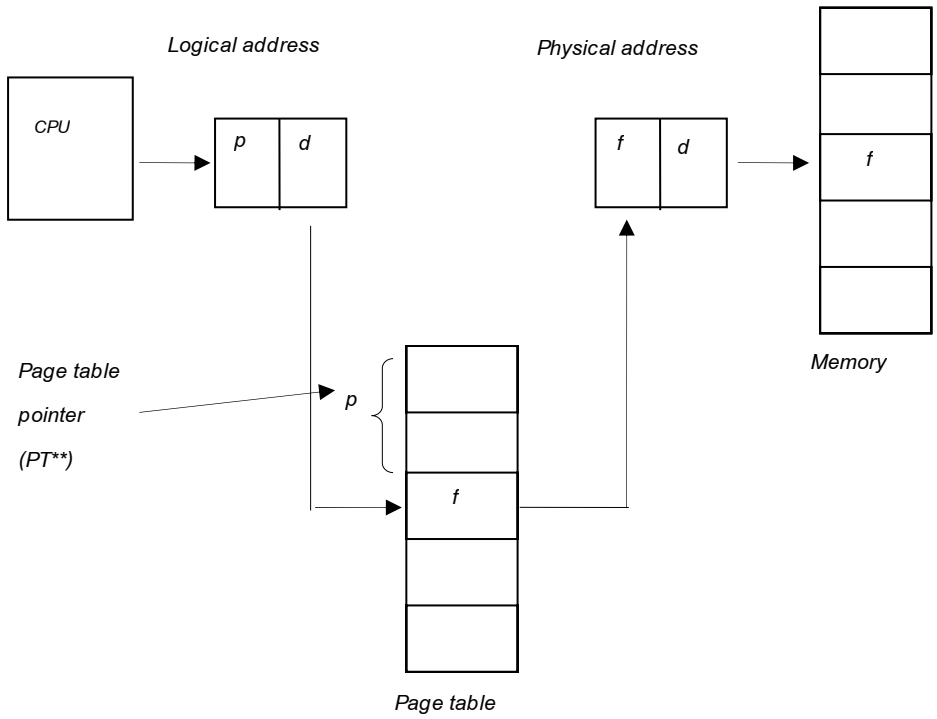
การแบ่งเป็นหน้า (Paging)

การแบ่งหน่วยความจำ โดยมีหลายขนาด มักทำให้เกิดการสูญเปล่าเนื้อที่ย่อยภายนอก (External Fragmentation) ซึ่งเกิดจากการที่พื้นที่ที่เหลืออยู่กระจัดกระจายกันในหน่วยความจำ เนื่องจากกระบวนการหนึ่ง ๆ ต้องการพื้นที่ต่อเนื่องกัน ตลอดทั้งกระบวนการไม่สามารถอยู่กระจัดกระจายได้ เราจึงต้องแก้ไขโดยการบีบอัดหน่วยความจำเพื่อย้ายเนื้อที่ว่างมาเรียงต่อกันให้สามารถใช้งานได้ ยังมีทางแก้อีกวิธีหนึ่งคือ การแบ่งเป็นหน้า (Paging) ซึ่งทำให้กระบวนการสามารถอยู่ในหน่วยความจำได้โดยไม่ต้องเรียงต่อเนื่องกันทั้งกระบวนการ เราจึงสามารถใช้เนื้อที่ว่างย่อย ๆ ที่อยู่กระจัดกระจายได้โดยไม่สูญเปล่า หรือไม่ต้องมีการบีบอัดก่อน ทั้งยังตัดปัญหาในการจัดการเนื้อที่ที่มีขนาดต่าง ๆ กันได้เพราะการแบ่งเป็นหน้า จะแบ่งเนื้อที่ในหน่วยความจำและในกระบวนการออกเป็นหน้า ๆ หน้าละเท่า ๆ กันโดยมีขนาดแน่นอนและเท่ากันทั้งหมด การสับเปลี่ยนกระบวนการเข้าออกจากหน่วยความจำไปสู่ที่พักข้อมูลชั่วคราว (Backing Store) จะทำได้ง่ายขึ้น เนื่องจากทุกหน้ามีขนาดเท่ากันหมด ปัญหาการสูญเสียพื้นที่ภานอกจึงหมดไป การเข้าถึงข้อมูลอาจช้าลงบ้าง การบีบอัดข้อมูลไม่มีความจำเป็นอีกต่อไป วิธีการแบ่งเป็นหน้านี้จึงเป็นที่นิยม ในระบบปฏิบัติการหลายระบบ เพราะมีข้อดีหลายอย่างเหนือกว่าวิธีในหัวข้อต้น ๆ

อุปกรณ์ช่วย (Hardware Support)

หน่วยความจำจริงจะถูกบ่งเป็นส่วน ๆ แต่ละส่วน มีขนาดเท่า ๆ กัน เรียกว่า หน้า (Frame) หรือ หน้าจริง และหน่วยความจำทางตรรกะของโปรแกรมก็จะถูกแบ่งเป็นส่วนๆ เช่นกันโดยมีขนาดเท่ากับส่วนของหน่วยความจำจริง เรียกว่าหน้า (Page) เมื่อต้องการให้กระบวนการหนึ่งทำงาน ก็เพียงแต่นำกระบวนการนั้น จากที่พักโปรแกรมชั่วคราวบรรจุลงในหน่วยความจริงโดยแยกเป็นหน้า ๆ ไม่ต้องต่อเนื่องกัน ที่พักโปรแกรมชั่วคราวก็ถูกแบ่งเป็นส่วน ๆ เช่นกัน ใช้ขนาดเดียวกับขนาดหน้าของหน่วยความจำจริง และหน้าทางตรรกะ

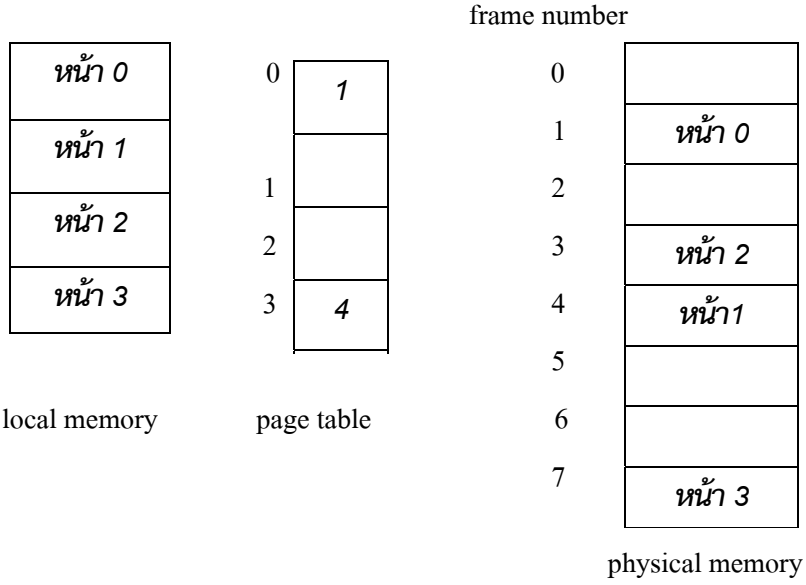
ระบบแบ่งเป็นหน้านี้ ต้องมีฮาร์ดแวร์ช่วย ดังแสดงในรูปที่ 7.13



รูปที่ 7.13 ฮาร์ดแวร์ช่วยในการแบ่งหน้า

ตำแหน่งทางตรรกะที่กระบวนการใช้จะถูกแบ่งเป็น 2 ส่วนหนึ่ง คือ หมายเลขหน้า (Page Number) p อีกส่วนหนึ่งเป็นระยะจากขอบหน้า (Page Offset) d หมายเลขหน้าใช้เป็นตัวชี้ไปยังตารางเลขหน้า (Page Table) ในตารางเลขหน้านี้ จะมีค่าตำแหน่งจุดเริ่มต้น (Base Address) f ของหน้าจริง ในหน่วยความจำหลัก ค่าตำแหน่งหน้าจริงนี้ รวมกับค่าระยะจากขอบหน้า (f +d) จะเป็น

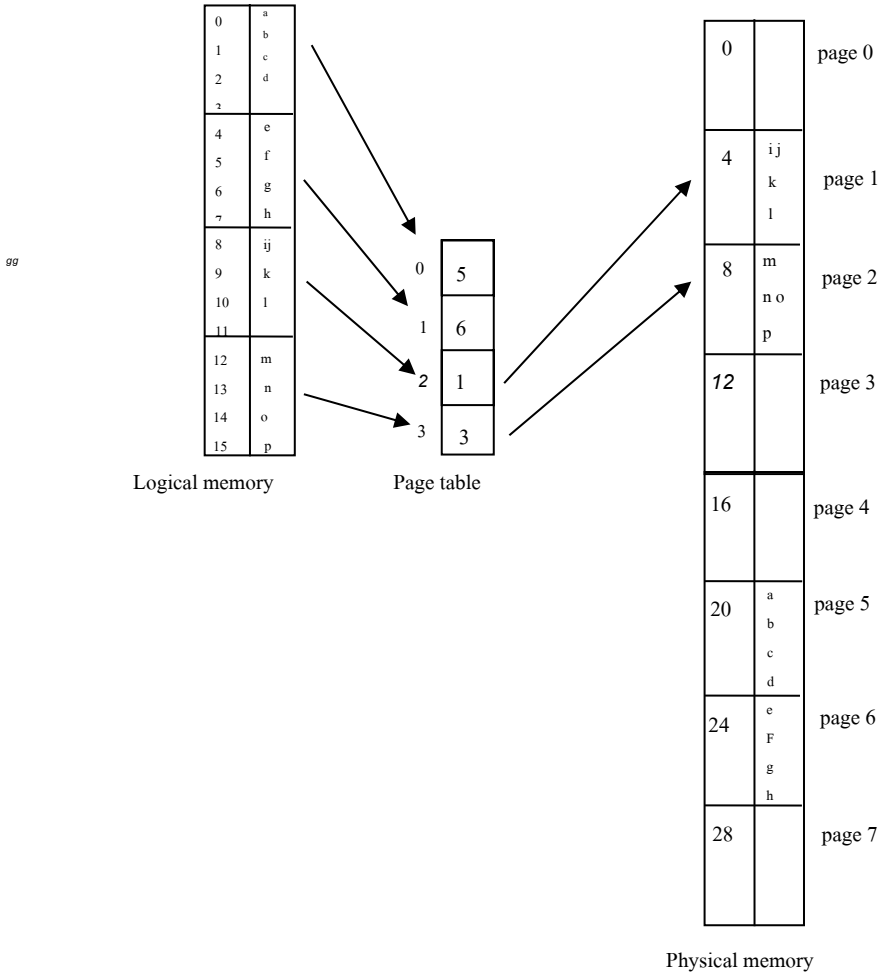
ตำแหน่งจริง ในหน่วยความจำหลัก รูปที่ 7.14 แสดงแผนภาพของหน่วยความจำทางตรรกะกับหน่วยความจำจริง



รูปที่ 7.14 การจับคู่การแบ่งหน้าทางตรรกะกับการแบ่งหน้าจริง

ขนาดของหน้า มักจะถูกกำหนดโดยฮาร์ดแวร์ หรือสถาปัตยกรรมของเครื่องคอมพิวเตอร์นั้น ๆ โดยปกติจะมีขนาดเป็นเลขยกกำลังของ 2 และนิยมใช้อยู่ระหว่าง 512 คำ ถึง 2048 คำ การที่เลือกขนาดของหน้าเป็นกำลังของ 2 เพื่อให้สะดวกในการแปลงตำแหน่งทางตรรกะ เป็นตำแหน่งจริง เช่น ขนาดของหน้าเป็น 2n คำ หรือไบท์ ตำแหน่งทางตรรกะ ส่วนล่าง n บิตจะเป็นค่าระยะจากขอบหน้า และบิตส่วนบนที่เหลือก็จะเป็นหมายเลขหน้า

เพื่อความเข้าใจ เราสมมุติตัวอย่างง่าย ๆ โดยกำหนดให้ ขนาดของหน้าเท่ากับ 4 คำ ดูรูปที่ 7.15 ประกอบ หน่วยความจำจริงมีขนาด 32 คำ รวม 8 หน้า และสมมุติให้หน่วยความจำทางตรรกะในมุนมองของกระบวนการผู้ใช้ มี 4 หน้าเท่ากับ 16 คำ ตำแหน่งทางตรรกะ 0 จะแสดงค่าหมายเลขหน้า 0 และระยะจากขอบ 0 จากหมายเลขหน้า 0 เทียบในตารางเลขหน้าจะเห็นว่าตรงกับหน้าจริงหมายเลข 5 ดังนั้น ตำแหน่งทางตรรกะ 0 แปลงเป็นตำแหน่งจริง = (5 * 4) + 0 = 20 ตำแหน่งทางตรรกะ 3 คือหน้า 0 ระยะห่าง 3 แปลงเป็นตำแหน่งจริงหน้า 6 ดูจากตารางเลขหน้า ได้ตำแหน่งจริง = (6 * 4) + 0 = 24 และตำแหน่งทางตรรกะ 13 แปลงเป็นตำแหน่งจริง 9



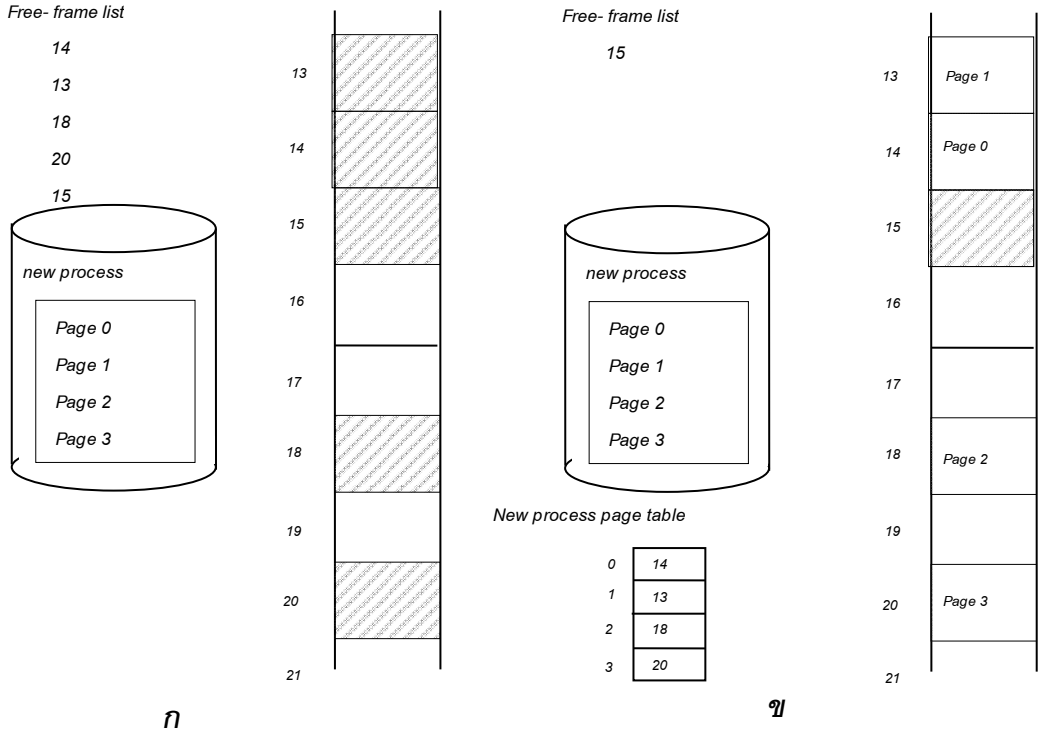
รูปที่ 7.15 ตัวอย่างการแบ่งเป็นหน้า สำหรับหน่วยความจำขนาด 32 ตำแหน่ง

จะสังเกตเห็นว่า การแบ่งเป็นหน้าเป็นระบบย้ายตำแหน่งสัมพัทธ์ (Dynamic Relocation) โดยที่ตำแหน่งทางตรรกะทุกตำแหน่ง อ้างอิงผ่านฮาร์ดแวร์ของระบบ (ตารางเลขหน้า และวิธีการแปลง) ไปสู่ตำแหน่งจริง ระบบแบ่งเป็นหน้านี้เปรียบเสมือนมีตารางของรีจิสเตอร์ฐาน (Base Register) หลายตัวโดยแต่ละตัว ใช้สำหรับหน้าจริงแต่ละหน้า

ตัวจัดตารางการทำงานระยะยาว (Long-Term Scheduling)

วิธีการจัดการหน่วยความจำมีผลสำคัญต่อ ตัวจัดตารางการทำงานระยะยาวเมื่อมีกระบวนการ ใหม่เข้ามาในระบบ ตัวจัดตารางระยะยาว จะดูที่ขนาดของกระบวนการ ซึ่งบอกเป็นจำนวนหน้าแล้วดูเนื้อที่ว่าง ในหน่วยความจำหลักว่ามีจำนวนหน้าว่างพอเพียงหรือไม่ ซึ่งระบบอาจเก็บเป็นรายชื่อหมายเลขหน้าว่าง ถ้ากระบวนการ มีขนาด n หน้า ตัวจัดตารางระยะยาว ก็ต้องจัดเนื้อ

ที่ว่าง ให้ได้ n หน้า เช่นกัน ถ้ามีเนื้อที่พอ ก็จะนำส่วนของกระบวนหน้าแรก ลงในเนื้อที่ว่าง ที่แรก และใส่หมายเลขหน้าจริง ลงในตารางเลขหน้าเช่นกัน ทำเช่นนี้ไปเรื่อย ๆ จนครบ n หน้า (ดูรูปที่ 7.16 ประกอบ)



รูปที่ 7.16 หน้าจริงที่ว่างอยู่ (ก) ก่อนการจัดสรร และ (ข) หลังการจัดสรร

ในระบบการแบ่งเป็นหน้าที่ จะไม่มี การสูญเปล่าพื้นที่ย่อยภายนอก เพราะเนื้อที่ในหน่วยความจำ ที่ยังว่างอยู่แม้จะไม่ต่อเนื่องกัน สามารถจะจัดให้กระบวนการ ที่ต้องการได้เสมอ แต่อาจมีการสูญเปล่าพื้นที่ย่อยภายในได้ เพราะการจัดสรรเนื้อที่จริง จัดให้ทีละหน้า เป็นจำนวนเต็ม ถ้ากระบวนการ ต้องการใช้เนื้อที่ ไม่เป็นจำนวนเต็มหน้า หน้าสุดท้ายย่อมเหลือเศษ มีพื้นที่บางส่วนว่างเปล่า แต่ถูกจัดสรร ให้กระบวนการ ด้วย ไม่เต็มหน้า เช่น ขนาดของหน้าเป็น 2048 ไบท์ กระบวนการ ต้องการ 72,766 ไบท์ คิดเป็นพื้นที่ 35 หน้า กับอีก 1086 ไบท์ ดังนั้นเราต้องจัดสรรเนื้อที่จริงให้ 36 หน้า มีการสูญเปล่าพื้นที่ย่อยภายในหน้า = 2048/ 1086 = 962 ไบท์ ในกรณีที่เลวร้ายที่สุดกระบวนการ หนึ่งอาจต้องการพื้นที่ n หน้า กับอีก 1 ไบท์ ซองทำให้เกิดการสูญเปล่าพื้นที่ย่อยภายใน ไปเกือบ 1 หน้าเต็ม เพราะต้องจัดสรรเนื้อที่ให้ n + 1 หน้า ถ้าขนาดของกระบวนการ ไม่ขึ้นกับขนาดของหน้าแล้ว การสูญเสียพื้นที่ย่อยภายในเฉลี่ย จะเท่ากับ ครึ่งหน้า แต่

ละกระบวนการ เมื่อเป็นเช่นนี้ เราจึงควรเลือก ให้หน้ามีขนาดเล็กดีกว่า แต่การที่กระบวนการ หนึ่งมีหลายหน้าเพราะหน้ามีขนาดเล็ก ทำให้ต้องใช้พื้นที่การเก็บตารางเลขหน้า มากขึ้นเมื่อหน้ามีขนาดใหญ่ พื้นที่ในการเก็บตารางเลขหน้า ก็จะลดลง

ระบบปฏิบัติการแต่ละระบบมีวิธีการเก็บ ตารางเลขน้าไม่เหมือนกัน แต่ส่วนใหญ่จะเก็บตารางเลขหน้า 1 ตาราง สำหรับ 1 กระบวนการ และเก็บตัวชี้ไปยังตารางเลขหน้า ไว้ในรีจิสเตอร์ซึ่งค่าของรีจิสเตอร์นี้ จะถูกเก็บอยู่ในตารางข้อมูลเฉพาะของกระบวนการ (Process Control Block : FCB) ดังนั้นเมื่อตัวย้ายการควบคุมหน่วยประมวลผลกลาง (Dispatcher) จะย้ายการควบคุมไปยังกระบวนการการหนึ่ง ก็จะต้องอ่านค่ารีจิสเตอร์ที่เก็บตัวชี้ไปยังตารางเลขหน้าของกระบวนการ นั้นเข้ามาด้วย

การสร้างตารางเลขหน้า

การสร้างตารางเลขหน้าจำเป็นต้องมีฮาร์ดแวร์ช่วย วิธีที่ง่ายที่สุดคือ ใช้รีจิสเตอร์เฉพาะงานจำนวนหนึ่งให้เท่ากับขนาดของตาราง ซึ่งควรสร้างมาให้มีความเร็วสูงพิเศษในการแปลงตำแหน่งทางตรรกะให้เป็นตำแหน่งจริง เพราะการอ้างอิงตำแหน่งในหน่วยความจำทุกครั้งต้องผ่านตารางเลขหน้านี้ ก่อนเสมอ ความเร็วในการแปลงจึงเป็นสิ่งสำคัญต่อประสิทธิภาพของระบบมาก ตัวย้ายการควบคุม มีหน้าที่บรรจุค่ารีจิสเตอร์พิเศษเหล่านี้ เหมือนกับรีจิสเตอร์อื่น ๆ ที่กระบวนการ ต้องใช้คำสั่งในการบรรจุค่า หรือแก้ไขค่าในรีจิสเตอร์พิเศษนี้ ต้องเป็นคำสั่งสงวน (Reserve Word) ซึ่งใช้ได้เฉพาะระบบปฏิบัติการเท่านั้น

การใช้รีจิสเตอร์เก็บตารางเลขหน้า เหมาะสมกับระบบที่มีตารางเลขหน้าขนาดเล็ก เช่น 256 ช่อง แต่สำหรับระบบที่มีตารางเลขหน้าขนาดใหญ่มาก เช่น 1 ล้านช่อง การเก็บเลขหน้าในรีจิสเตอร์พิเศษ จะไม่คุ้มค่าอีกต่อไป จำเป็นต้องเก็บตารางเลขหน้าไว้ในหน่วยความจำหลักแทน และใช้รีจิสเตอร์ เก็บตัวชี้ไปยังตารางเลขหน้าแทน (Page-Table Base Register : PTBR) การเปลี่ยนตารางเลขหน้าใหม่ ก็เพียงเปลี่ยนค่าในรีจิสเตอร์ที่เป็นค่าตัวชี้นี้เพียง 1 ตัว จะช่วยลดเวลาที่ใช้ในการสับเปลี่ยนงานได้มาก

การเก็บตารางเลขหน้าในหน่วยความจำหลักนี้ ทำให้การอ้างอิงตำแหน่งในหน่วยความจำต้องทำเป็น 2 ขั้นตอน คือ ขั้นแรก เมื่อเราต้องการอ้างอิงตำแหน่งทางตรรกะ 1 เราต้องอ่านค่าในตารางเลขหน้าก่อน โดยดูตำแหน่งตารางเลขหน้า จาก PTBR เพื่อดูว่าตรงกับหน้าจริงหมายเลขใด ขั้นตอนนี่จะต้องอ่านหน่วยความจำหลัก 1 ครั้ง ขั้นที่สอง ใช้หมายเลขหน้าจริง ที่ได้รวมกับค่าระยะจากขอบหน้าใน I เป็น ค่าตำแหน่งจริง ที่ใช้ในการอ่านคำที่ต้องการจากหน่วยความจำหลัก อีกครั้งหนึ่ง รวม 2 ขั้นตอน ต้องอ่านหน่วยความจำหลัก 2 ครั้ง ต่อการอ้างอิงตำแหน่งทางตรรกะ 1 ครั้ง

ดังนั้น ความเร็วในการอ้างอิงหน่วยความจำ จะลดลงครึ่งหนึ่ง ซึ่งไม่อาจยอมรับได้เพราะช้าลงถึง 50%

ปัญหานี้นิยมแก้ไขโดยใช้หน่วยความจำพิเศษ ที่เรียกว่า รีจิสเตอร์เสริม (Associative Registers หรือ Translation Lookaside Buffers : TLB) ซึ่งเป็น ฮาร์ดแวร์ประเภทหน่วยความจำที่มีความเร็วสูงมาก รีจิสเตอร์เสริมและตัวจะมี 2 ส่วน ส่วนที่นึ่งเป็นดัชนี อีกส่วนหนึ่งเป็นข้อมูลเมื่อเราส่งค่า เข้าไปในกลุ่มของรีจิสเตอร์เสริม รีจิสเตอร์เสริมทุกตัว จะเทียบค่านี้กับดัชนีของตนพร้อมกัน ถ้าตรงกับดัชนีของรีจิสเตอร์ใด รีจิสเตอร์นั้นก็จะส่งผลลัพธ์ เป็นข้อมูลของตน ออกมาทันที การค้นหาเปรียบเทียบค่าดัชนี โดยใช้รีจิสเตอร์เสริมนี้ เร็วมาก รีจิสเตอร์เหล่านี้ จะมีราคาแพงมาก

รีจิสเตอร์เสริมเหล่านี้จะใช้เก็บบางส่วนของตารางเลขหน้า เมื่อมีการอ้างอิงตำแห่งทางตรรกะ ระบบก็จะตรวจดูหมายเลขหน้าจากรีจิสเตอร์เสริมเหล่านี้ก่อนเพื่อหาค่าหมายเลขหน้าจริง ถ้าพบหมายเลขที่ต้องการ ในริจิสเตอร์เสริมเหล่านี้ ก็สามารถใช้อ้างอิง ตำแหน่งจริงได้ทันที เวลาที่ใช้ในการอ้างอิง โดยมีริจิสเตอร์เสริมช่วยนี้มากกว่า การอ้างอิงหน่วยความจำโดยตรง เพียงไม่เกิน 10% เท่านั้น

แต่ถ้าไม่พบหมายเลขหน้าที่ต้องการในรีจิสเตอร์เสริมเหล่านี้ เราก็ต้องทำแบบเดิมคือ อ่านตารางเลขหน้า จากหน่วยความจำหลัก และนำค่าที่ได้ไปอ่านหน่วยความจำจริงอีกทีหนึ่ง อาจจะกล่าวได้ว่า การใช้รีจิสเตอร์เสริม ช่วยให้ การแปลงหมายเลขหน้าเป็นหน้าจริง ทำได้เร็วขึ้น เมื่อมีการอ้างอิงซ้ำอีก ในครั้งต่อไป

อัตราส่วนของการพบหมายเลขหน้าที่ต้องการในรีจิสเตอร์เสริม ต่อจำนวนการอ้างอิงทั้งหมด เรียกว่า อัตราส่วนการชน (Hit Ratio) เช่น มีอัตราส่วนการชนเป็น 80% หมายความว่า หมายเลขหน้าที่ต้องการอยู่ในรีจิสเตอร์เสริม 80 % ของการอ้างอิงหน่วยความจำ ถ้าให้การค้นหาหมายเลขหน้าในรีจิสเตอร์เสริมเสียเวลา 20 นาโนวินาที และเวลาที่ใช้ในการอ่านหน่วยความจำเป็น 100 นาโนวินาที การอ้างอิงหน่วยความจำ โดยที่พบหมายเลขหน้าในรีจิสเตอร์เสริม จะเสียเวลาเท่ากับ 100 + 20 = 220 นาโนวินาที แต่ถ้าไม่พบในรีจิสเตอร์เสริม เสียเวลาค้นหาไปแล้ว 20 นาโนวินาที แต่ไม่พบ จะต้องอ่านหน่วยความจำอีก 2 ครั้ง (2*100 = 200 นาโนวินาที) จึงเสียเวลาไปรวม 20 + 200 = 220 นาโนวินาที การหาเวลาเฉลี่ยต้องคิด อัตราส่วนการชน เข้าไปด้วย ดังนั้น

เวลาเฉลี่ย ในการอ้างอิงหน่วยความจำหลัก = 80%* 120 + 20%*= 220

= 140 นาโนวินาที

จากตัวอย่างนี้ จะพบว่าการอ้างอิงหน่วยความจำหลักช้าลงกว่าการอ่านโดยตรง 29 % คือจาก 100 เป็น 140 นาโนวินาที ถ้าให้อัตราส่วนการชนเป็น 90 %

เวลาเฉลี่ยในการอ้างอิงหน่วยความจำหลัก = 90% * 120 + 10% *220

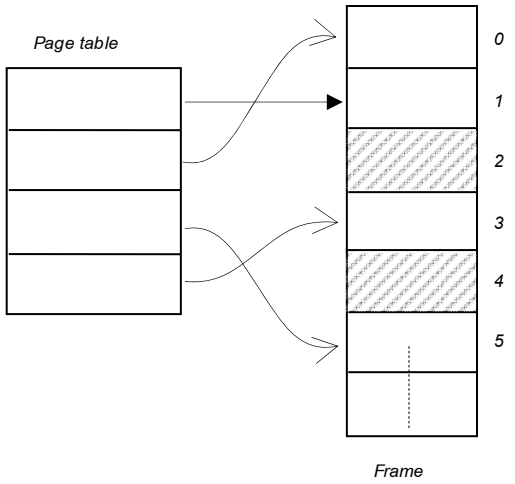
= 130 นาโนวินาที

ลดเวลาที่เสียไป ในการอ้างอิงหน่วยความจำหลักเป็นช้าลงกว่าแบบโดยตรงเพียง 23%

จำนวนรีจิสเตอร์เสริม ย่อมมีผลโดยตรงต่ออัตราส่วนการชน จากการทดลองพบว่าการใช้รีจิสเตอร์เสริมระหว่าง 16 ถึง 512 ตัว สามารถให้อัตราส่วนการชน 80 ถึง 95% หน่วยประมวลผล 68030 ของโมโตโรล่า มี TLB = 22 ช่อง หน่วยประมวลผล 80486 ของอินเทล มีรีจิสเตอร์ 32 ตัว และอ้างว่าให้อัตราส่วนการชน ถึง 98 %

การใช้หน้าร่วม

ข้อดีอีกข้อหนึ่ง ของวิธีแบ่งเป็นหน้า ก็คือ อาจใช้หน้าร่วมกันได้ ซึ่งมีประโยชน์มากในระบบแบ่งเวลา (Time-Sharing) ลองคิดดูว่า ถ้ามีผู้ใช้ 40 คน กำลังใช้ระบบพร้อมกัน และใช้โปรแกรมบรรณาธิกรณ์ (Text Editor) แก้ไขแฟ้มข้อมูลอยู่ โดยที่โปรแกรมนี้ ใช้เนื้อที่โปรแกรม 150 K และเนื้อที่เก็บข้อมูล 50 K สำหรับผู้ใช้ 40 คน เราต้องใช้เนื้อที่ในหน่วยความจำถึง 40 x (150+50) = 8000 K ส่วนของโปรแกรมสามรถใช้ร่วมกันได้ ดูรูปที่ 7.17 ประกอบ ในรูปจะเห็นโปรแกรมบรรณาธิกรณ์ ขนาด 3 หน้า หน้าละ 50 K เราใช้ขนาดของหน้าใหญ่มากถึง 50 K เพียงเพื่อให้ดูง่ายเท่านั้น และมี กระบวนการ ผู้ใช้ 3 คน ใช้ร่วมกันอยู่



รูปที่ 7.17 การใช้โปรแกรมร่วมในระบบที่ใช้การแบ่งหน้า

โปรแกรมส่วนที่ไม่มีการแก้ไขระหว่างการทำงาน (Re-Entrant Code หรือ Pure Code) นี้สามารถใช้งานร่วมกันได้หลาย ๆ ประการในเวลาเดียวกัน โดยแต่ละกระบวนการมีรีจิสเตอร์ และข้อมูลภายในของตนเอง สำหรับการทำงาน เราจึงอาจ เก็บโปรแกรมในหน่วยความจำจริง ไว้เพียง

ชุดเดียว ตารางเลขหน้า ของแต่ละกระบวนการ จะชี้ไปยังหน้าจริงที่เดียวกัน โดยจะแตกต่างกันเฉพาะส่วนที่เป็นข้อมูล ดังนั้น เราใช้เนื้อที่จริงเพียง 150+40*50 = 2150 K โดยรวม แทนที่จะเป็น 8000 K

โปรแกรมอื่น ๆ ที่ใช้มาก ๆ สามารถใช้ร่วมกันได้ เช่นกัน เช่น ตัวแปลโปรแกรม ระบบหน้าต่าง ระบบฐานข้อมูล เป็นต้น โดยโปรแกรมต้องไม่มีการแก้ไขตัวเองในระหว่างการทำงาน ระบบปฏิบัติการ ต้องมีการป้องกัน การแก้ไขตัวเอง ของโปรแกรมเหล่านี้ด้วย ไม่ใช่เพียงเชื่อว่าโปรแกรมจะไม่มีการแก้ไขตัวเองหรือใช้อ่านอย่าเดียว เท่านั้น เพราะ อาจมีข้อผิดพลาดในโปรแกรมเหล่านี้ได้

การป้องกัน (Protection)

การป้องกันหน่วยความจำในระบบแบ่งเป็นหน้า ทำโดยการใช้ “บิต ป้องกัน” (Protection Bits) ควบคู่ไปกับทุก ๆ หน้า โดยใส่บิต ป้องกันไว้ในตารางเลขหน้าเพียง 1 บิต ก็เพียงพอ ที่จะบอกว่าหน้านี้เป็นแบบที่แก้ไขได้ หรือใช้อ่านเท่านั้น เนื่องจากการอ้างอิงหน่วยความจำทุกครั้งต้องผ่านตารางเลขหน้า เราจึงสามารถใช้บิต ป้องกันเพื่อตรวจสอบว่ามีการเขียนลงไปในหน้าชนิดที่สำหรับอ่านอย่างเดียวหรือไม่ ถ้ามีก็รายงานเป็นข้อผิดพลาดไปยังระบบปฏิบัติการ คือละเมิดการป้องกันหน่วยความจำ

เราอาจขยายวิธีการใช้บิตป้องกันนี้ไปให้ครอบคลุม การป้องกันมากขึ้นอีก โดยเพิ่มชนิดของบิตป้องกันเป็นสำหรับอ่าน (Read) เท่านั้น อ่านและเขียน (Read/Write) หรือใช้ทำงาน (Execute) เท่านั้น เพื่อป้องกันการใช้ผิดวัตถุประสงค์ได้

กระบวนการทั่ว ๆ ไป มักไม่ได้ใช้ตำแหน่งทางตรรกะทั้งหมด หรือไม่ได้มีขนาดใหญ่จนเต็มหน่วยความจำส่วนใหญ่ใช้เนื้อที่เพียงเล็กน้อย ก็พอเพียงต่อการทำงานแล้ว ดังนั้นถ้าเราสร้างตารางเลขหน้าทั้งตารางให้แต่ละกระบวนการ ก็จะเป็นการเสียเนื้อที่ไปโดยเปล่าประโยชน์ บางระบบจึงมีฮาร์ดแวร์เสริมอีก เรียกว่า รีจิสเตอร์ขอบเขตของตารางเลขหน้า (Page-Table Length Register : PTLR) เพื่อเก็บ ขนาดของตารางเลขหน้า ถ้ามีการอ้างอิงตำแหน่งเกิน ขนาดของตารางเลขหน้านี้ไป ก็จะเกิดข้อผิดพลาดไปยังระบบปฏิบัติการ

หน่วยความทางตรรกะ กับหน่วยความจำจริง (Two View of Memory)

ในระบบการแบ่งเป็นหน้านี้ ทำให้ผู้ใช้มองเห็นหน่วยความจำทางตรรกะ ต่างไปจากหน่วยความจำจริงโปรแกรมผู้ใช้ (User Program) มองว่าหน่วยความจำเป็นพื้นที่ต่อเนื่องกันผืนเดียว และมีโปรแกรมอยู่เพียงโปรแกรมเดียวเท่านั้น แต่ในความเป็นจริงโปรแกรมของผู้ใช้ อาจกระจัดกระจายอยู่ทั่วไปในหน่วยความจำ ซึ่งยังมีโปรแกรมของผู้ใช้อื่นอยู่ด้วย ฮาร์ดแวร์ของเครื่อง

เป็นผู้แปลงตำแหน่งทางตรรกะไปเป็นตำแหน่งจริง ซึ่งระบบปฏิบัติการเป็นผู้ควบคุมการทำงานนี้อีกชิ้นหนึ่งโดยผู้ใช้มองไม่เห็นเลย

การแบ่งแยก หน่วยความจำทางตรรกะ จากหน่วยความจำจริง ทำให้เราสามารถสร้างหน่วยความจั้งสอง ให้มีขนาดแตกต่างกันได้ เครื่องคอมพิวเตอร์ขนาดกลางในยุคแรก ๆ กำหนดให้โปรแกรมมีขนาดเล็ก เพราะฮาร์ดแวร์หน่วยความจำ มีราคาแพง โดยกำหนดตำแหน่งหน่วยความจำด้วย 15 หรือ 16 บิต ต่อมา ชิ้นส่วนคอมพิวเตอร์มีราคาถูกลง การขยายหน่วยความจำจริง เริ่มคุ้มค่าเพราะถูกลง แต่การเพิ่มขนาดของหน่วยความจำ ทำให้ต้องใช้ บิตอ้างอิงตำแหน่งเพิ่มด้วย เป็น 17 หรือ 18 บิต ซึ่งส่งผลกระทบโดยตรง ต่อคำสั่งในหน่วยประมวลผล ที่เคยใช้ตำแหน่งเพียง 15 หรือ 16 บิต ทำให้อาจต้องมีการแก้ไขหน่วยประมวลผลด้วย ทั้งยังต้องแก้ไขโปรแกรมต่าง ๆ ที่มีอยู่เดิมใหม่หมดด้วย ผู้ผลิตส่วนใหญ่แก้ปัญหานี้ โดยการให้หน่วยความจำทางตรรกะเป็น 15 หรือ 16 บิต อย่างเดิม และให้หน่วยความจำจริงเป็น 17 หรือ 18 บิต ในระบบทำงานแบบหลายโปรแกรมระบบอาจใช้เนื้อที่หน่วยความจริงทั้งหมดได้ แต่ผู้ใช้แต่ละคน ยังคงใช้เนื้อที่หน่วยความจำทางตรรกะได้เท่าเดิม เพราะตำแหน่งอ้างอิง ยังคงมีขนาดเท่าเดิม

ระบบปฏิบัติการเป็นผู้ควบคุมการจับคู่ตำแหน่งทางตรรกะกับตำแหน่งจริง โดยใช้บังคับกับกระบวนการของผู้ใช้ แต่ระบบเองสามารถอ้างอิงตำแหน่งจริงได้โดยตรง ดังนั้น ระบบจะต้องรู้ข้อมูลทุกอย่างเกี่ยวกับการจัดสรรเนื้อที่ในหน่วยความจริงเหล่านี้ เช่น หน้าใดยังว่างอยู่ หน้าใดกระบวนการใดครอบครองอยู่ มีเนื้อที่จริงทั้งหมดกี่หน้า เป็นต้น ข้อมูลเหล่านี้ปกติจะเก็บอยู่ในตารางเลขหน้าจริง (Frame Table)

นอกจากนี้ ระบบปฏิบัติการจะต้องระลึกอยู่เสมอว่า กระบวนการผู้ใช้กำลังทำงานอยู่ในพื้นที่ของผู้ใช้เอง การอ้างอิงตำแหน่งทางตรรกะโดยกระบวนการผู้ใช้จะต้องถูกแปลงเป็นตำแหน่งจริงเสมอ ไม่ว่าผู้ใช้จะทำคำสั่งใด ๆ เช่น ผู้ใช้ทำคำสั่งเรียกระบบ ให้รับสั่งข้อมูลผ่านอุปกรณ์และบอกตำแหน่งของที่พักข้อมูล มาด้วยในตัวแปร ระบบก็ต้อง แปลงค่าตำแหน่งเหล่านี้ เป็นตำแหน่งจริงก่อนที่จะส่งข้อมูลไปยังอุปกรณ์รับส่งข้อมูลที่ต้องการระบบต้องเก็บตารางเลขหน้าของกระบวนการผู้ใช้แต่ละตัวเหมือนการเก็บข้อมูล ค่าตัวชี้คำสั่ง และค่ารีจิสเตอร์ต่าง ๆ ของกระบวนการ ตารางเลขหน้าเป็นสิ่งจำเป็นอย่างมากในการแปลงตำแหน่งทางตรรกะ เป็นตำแหน่งจริง

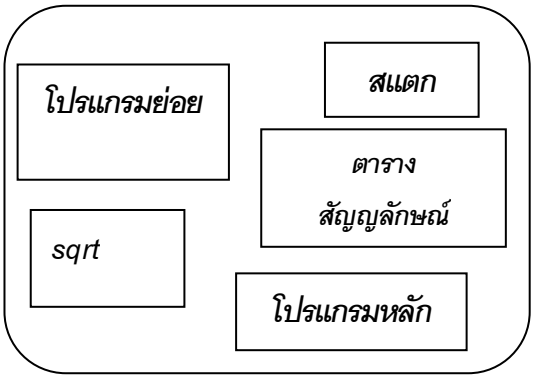
การแบ่งเป็นตอน

การแบ่งหน่วยความจำ เป็นหน้าทำให้เกิดภาพหน่วยความจำทางตรรกะ ซึ่งผู้ใช้มองเห็นแตกต่างไป จากลักษณะของหน่วยความจริงอย่างหลีกเลี่ยงไม่ได้ โดยมีระบบแปลงตำแหน่งทาง

ตรรกะ ให้เป็นตำแหน่งจริง ทำให้ระบบทำงานได้ถูกต้อง แม้ว่าทั้งสองภาพจะมีความแตกต่างกันก็ตาม

หน่วยความจำในมุมมองของผู้ใช้ (User's View of Memory)

ผู้ใช้มองเห็นหน่วยความจำมีลักษณะอย่างไร อาจมองเห็นมีลักษณะเป็นเนื้อที่เก็บช่องละคำ เรียงต่อกันเป็นเส้นตรง บางส่วนเก็บโปรแกรม บางส่วนเก็บข้อมูลหรือว่าเป็นแบบอื่น ๆ โดยทั่วไป ผู้ใช้จะมองหน่วยความจำหลัก เป็นกลุ่มหรือเป็นตอน ที่มีขนาดแปรเปลี่ยนได้ และไม่ได้เรียงต่อกันด้วย (ดูรูปที่ 7.18)



เนื้อที่หน่วยความจำหลักทางตรรกะ

รูปที่ 7.18 โปรแกรมในมุมมองของผู้ใช้

ลองพิจารณาดู เวลาเราเขียนโปรแกรมจะมีโปรแกรมหลัก (Main Program) โปรแกรมย่อย (Subprogram) ฟังก์ชั่น โมดูล โครงสร้างข้อมูล ตาราง แถวลำดับ แสตก ตัวแปร และ อื่น ๆ ส่วนต่างๆ เหล่านี้ จะถูกอ้างอิงด้วยชื่อ ผู้เขียนโปรแกรมจะอ้างอิง ชื่อของฟังชั่น sqrt จากในโปรแกรมหลัก (Main Program) โดยไม่สนใจตำแหน่งจริง ๆ ในหน่วยจริง ๆ ว่าตอนใดจะอยู่ก่อนหรือหลังตอนใด โปรแกรมแต่ละตอน (Segment) จะมีขนาดแตกต่างกัน ทั้งนี้ขึ้นอยู่กับงานใช้งานของตอนนั้น ๆ

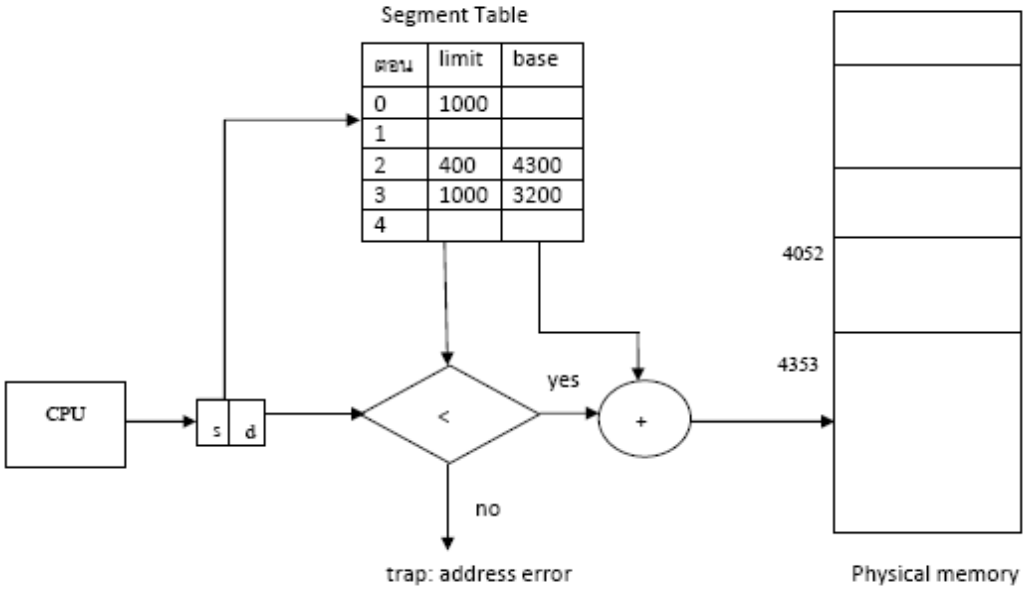
การแบ่งเป็นตอน (Segmentation) เป็นการจัดการหน่วยความจำหลัก ตามมุมของผู้ใช้ หน่วยความจำทางตรรกะ จะถูกแบ่งเป็นตอน ๆ แต่ละตอนจะมีชื่อ และขนาด ตำแหน่งอ้างอิงก็จะมีชื่อตอน กับระยะห่างจากขอบ (Offset) ดังนั้น ผู้ใช้สามารถอ้างอิงหน่วยความจำ โดยบอกชื่อตอนและระยะห่างจากขอบ (Offset) ซึ่งต่างจากระบบแบ่งเป็นหน้า ที่ผู้ใช้อ้างอิงตำแหน่งเป็นค่าเดียว แต่ฮาร์ดแวร์ของระบบจะแบ่งค่าตัวเลขเป็น 2 ส่วนเอง คือเลขหน้าและระยะจากขอบ โดยที่ผู้ใช้มองไม่เห็น

เพื่อความสะดวก ชื่อตอน มักใช้ตัวเลขแทนเรียกว่า “เลขตอน” (Segment Number) โดยปกติตัวแปลภาษาแอสแซมบลี้ หรือตัวแปลภาษาชั้นสูงอื่น ๆ จะแบ่งโปรแกรมเป็นตอน ๆ โดยอัตโนมัติ เช่น ตัวแปลภาษาปาสคาล อาจแบ่งตอน เป็น ตอนที่ 1 เก็บตัวแปรร่วม (Global Variables) ตอนที่ 2 เป็นเนื้อที่สำหรับ การเรียกโปรแกรมย่อย (Subprogram) เพื่อใช้เก็บค่าตัวแปรต่าง ๆ และตำแหน่งในการเรียกกลับ ตอนที่ 3 เก็บคำสั่งของโปรแกรมย่อยต่าง ๆ และ ตอนที่ 4 เก็บตัวแปรภายใน (Local Varibles) สำหรับโปรแกรมย่อย (Subprogram) ตัวแปลภาษาฟอร์แทรน อาจแบ่งเป็น ตอนหนึ่งสำหรับตัวแปรร่วม ตอนหนึ่งสำหรับ ตัวแปรประเภทแถวลำดับ (Arrays) ตัวบรรจุโปรแกรมลงหน่วยความจำ จะกำหนดหมายเลขตอน ให้ตอนเหล่านี้

หน่วงประมวลผล 8086 ของอินเทล มีการจัดการหน่วยความจำแบบแบ่งเป็นตอน โดยจัดแบ่งโปรแกรมเป็น 3 ตอน คือ ส่วนคำสั่ง (Code) ส่วนข้อมูล (Data) และส่วนแสตก (Stack)

ฮาร์ดแวร์สนับสนุน (Hardware Support)

แม้ว่าผู้ใช้สามารถ อ้างอิงส่วนต่าง ๆ ของโปรแกรมโดยใช้ตำแหน่งแบบ 2 มิติ แต่หน่วยความจำจริง ยังคงเป็นแบบมิติเดียว คือเป็นแถวของคำเรียงต่อกันไป จึงต้องมีวิธีการจับคู่ตำแหน่งทางตรรกะ แบบสองมิติ ให้เป็นตำแหน่งจริงมิติเดียว โดยใช้ตารางเลขตอน (Segment Table) ดังรูปที่ 7.19 แสดงวิธีใช้ ตารางเลขตอน ตำแหน่งทางตรรกะ แบ่งได้เป็น 2 ส่วน คือ หมายเลขตอน s และระยะจากขอบ d เราใช้หมายเลขตอน เป็นตัวชี้ไปยังข้อมูลในตารางเลขตอน ข้อมูลแต่ละช่อง ในตารางเลขตอน มีค่าฐาน (Base) และขอบเขตของตอน (Limit) ระยะจากขอบ d จะมีค่าระหว่าง 0 จนถึงขอบเขตของตอน ถ้า d มากกว่า ขอบเขตขอบตอนแล้ว จะเกิดข้อผิดพลาด จะรายงานไปยังระบบปฏิบัติการว่าเกิดการผิดพลาดจากการอ้างอิงตำแหน่งออกนอกตอน ถ้าค่า d ไม่เกินค่าขอบเขตตอนบน ฮาร์ดแวร์ก็จะนำค่า d ไปบวกกับค่าฐาน เป็น ค่าตำแหน่งจริงของหน่วยความจำ จะเห็นได้ว่า ตารางเลขตอนก็คือแถวลำดับของรีจิสเตอร์ฐาน (Base Register) และขอบเขตดังตัวอย่างจากรูปที่ 7.20 ประกอบ มี 5 ตอน หมายเลข 0 จนถึง 4 แต่ละตอน เก็บอยู่ในหน่วยความจริง ดังแสดงในรูป ตารางเลขตอน จะมีค่าฐานและขอบเขตของแต่ละตอน เช่น ตอนที่ 2 มีค่าฐานหรือตำแหน่งเริ่มต้นที่ 4300 และ ค่าขอบเขต 400 ไบท์ ดังนั้น



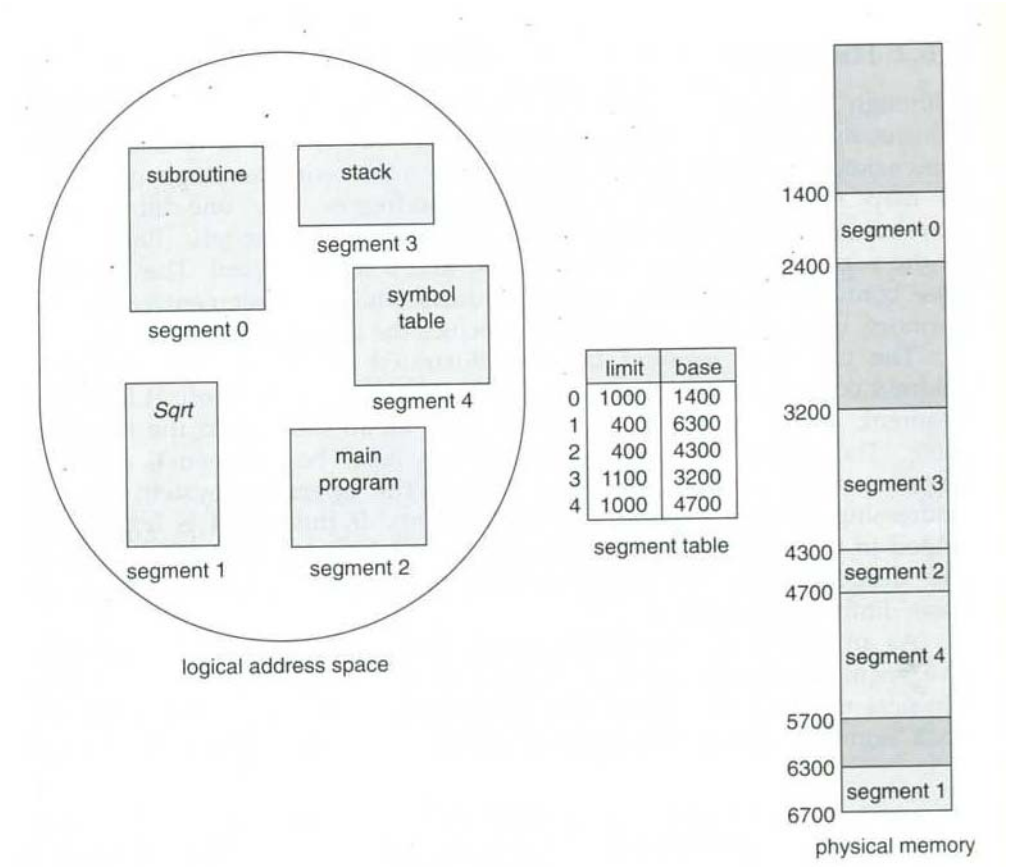
รูปที่ 7.19 ฮาร์ดแวร์ที่ใช้ในระบบแบ่งเป็นตอน

การอ้างอิงไบท์ที่ 53 ของตอนที่ 2 จะเป็นค่าตำแหน่งจริง 4300 + 53 = 4353 ตำแหน่งทางตรรกะของตอนที่ 3 ไบท์ที่ 852 เท่ากับตำแหน่งจริง 3200 + 852 = 4052 การอ้างอิงตำแหน่งไบท์ที่ 1222 ของตอนที่ 0 จะทำให้เกิดข้อผิดพลาด รายงานไปยังระบบปฏิบัติการ เพราะค่า 1222 เกินกว่าค่าขอบเขต ซึ่ง > 1000 ไบท์ ของตอนที่ 0

การสร้างตารางเลขตอน (Implementation of Segment Tables)

การแบ่งเป็นตอน คล้ายกับการแบ่งหน่วยความจำหลักออกเป็นตอน ๆ หรือส่วนจากในหัวข้อต้น ๆ มีข้อแตกต่าง คือ โปรแกรมหนึ่ง ๆ อาจมีได้หลายตอนหรือหลายส่วน การแบ่งเป็นตอน จึงมีความซับซ้อนกว่า เราอาจเก็บตารางเลขตอน ไว้ในรีสเตอร์พิเศษ ที่มีความเร็วสูง หรือเก็บในหน่วยความจำหลัก เหมือนกับตารางเลขหน้าฮาร์ดแวร์ของเครื่อง สามารถเปรียบเทียบ ค่าขอบเขต กับค่าระยะห่างจากขอบไปพร้อม ๆ กับ การบวกกับค่าฐานได้

ถ้าโปรแกรมหนึ่ง แบ่งเป็นตอน ๆ จำนวนมาก เราก็ไม่สามารถเก็บตารางเลขตอน ในรีจิสเตอร์พิเศษได้ ต้องเก็บตารางเลขตอนซึ่งมีขนาดใหญ่ไว้ในหน่วยความจำหลักแทน แล้วใช้รีจีสเตอร์พิเศษ (Segment Table Base Register : STBR) เก็บตัวชี้ไปยังตารางเลขตอนอีกที เนื่องจากโปรแกรมต่าง ๆ มีจำนวนตอนไม่เท่ากัน เราจึงอาจเก็บค่า จำนวนตอน ไว้ในรีจิสเตอร์พิเศษ ที่



รูปที่ 7.20 ตัวอย่างการแบ่งหน่วยความจำหลักเป็นตอน

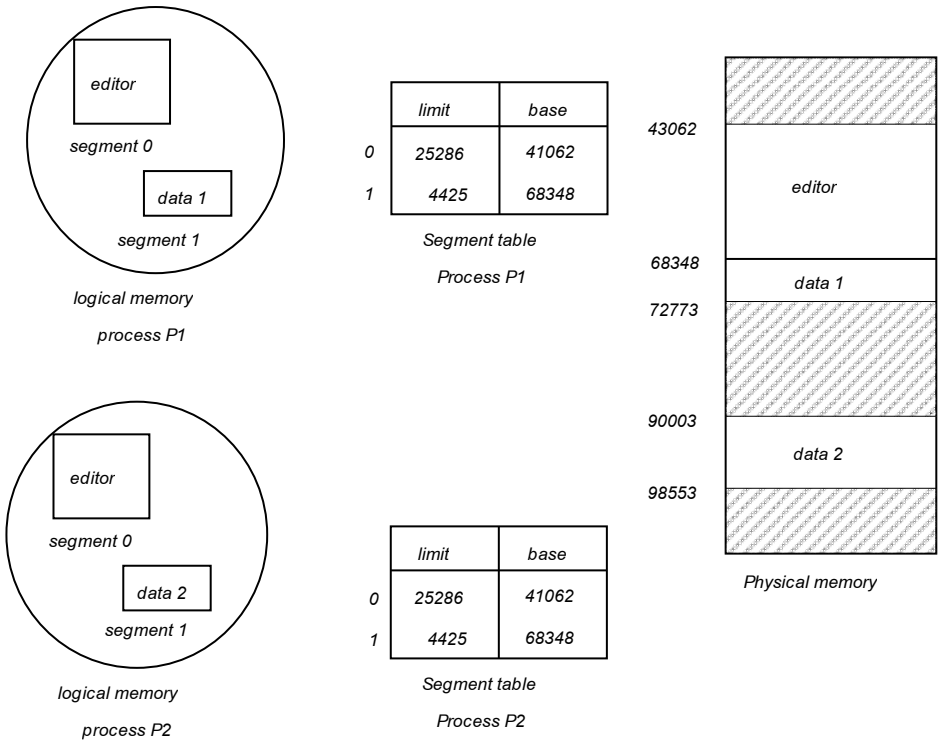
เรียกว่า Segment Table Length Register (STLR) เช่น ตำแหน่งทางตรรกะเป็น (s,d) เราต้องตรวจดูว่า s < STLR คือหมายเลขตอนไม่เกิน จำนวนตอนที่มีจริง แล้วอ่านค่าข้อมูลจาก ตารางเลขตอนในหน่วยความจำ ที่ตำแหน่ง ของ STBR + s เมื่อได้ข้อมูลเลขฐานและขอบเขตแล้วก็ทำเหมือนเดิมคือตรวจสอบว่า d < ขอบเขตหรือไม่ แล้วเอาค่า d บวกกับค่าฐานเป็นตำแหน่งจริง ในหน่วยความจำหลัก

เหมือนกับ การแบ่งเป็นหน้า การอ่านหน่วยความจำ 2 ครั้ง ต่อ การอ้างอิงตำแหน่งครั้งหนึ่ง ทำให้ระบบทำงานช้าลง 2 เท่า จึงมีการใช้รีจิสเตอร์เสริมหลายตัว ช่วยเก็บค่าที่เพิ่งจะใช้ไป แล้วเช่นเดียวกัน เราใช้รีจิสเตอร์เสริม ไม่มากนัก ก็สามารถลด เวลาเฉลี่ยในการอ้างอิงหน่วยความจำลงเป็นไม่เกิน 10 หรือ 15 % จากการอ้างอิงแบบโดยตรง

การป้องกันและการใช้ตอนร่วมกัน (Protection and Sharing)

ข้อดีหลักของการแบ่งเป็นตอน คือ สามารถผนวกการป้องกันไปกับแต่ละตอนได้ เพราะแต่ละตอนคือ ส่วนต่าง ๆ ของโปรแกรมที่มีลักษณะต่าง ๆ กัน ข้อมูลในตอนเดียวกัน มักจะมีการใช้งานเหมือน ๆ กัน เช่น ตอนของโปรแกรม ตอนของข้อมูล เป็นต้น ในระบบทั่วไป เราอาจกำหนด ตอนของโปรแกรมหรือคำสั่ง ให้เป็นแบบอ่านได้เท่านั้น (Read-Only) หรือใช้ทำงานเท่านั้น (Execute-Only) โดยการใช้บิต ป้องกันควบคู่กับแต่ละตอน ในตารางเลขตอน เพื่อป้องกันการใช้ผิดประเภท เช่น เขียนลงใน ตอนที่ใช้อ่านได้เท่านั้น หรือใช้ทำงานได้เท่านั้น หรือ ใส่ข้อมูลประเภทแถวลำดับ (Array) ไว้ในตอนเฉพาะ ฮาร์ดแวร์ของระบบก็จะสามารถช่วยตรวจดูได้ว่า มีการอ้างอิงข้อมูล ออกนอกขอบเขตแถวลำดับหรือไม่ ดังนั้นฮาร์ดแวร์ อาจช่วยตรวจจับข้อผิดพลาดเบื้องต้นต่าง ๆ ในโปรแกรมได้

ข้อดีอีกข้อหนึ่งก็คือ สามารถใช้ข้อมูล หรือโปรแกรมร่วมกันได้สะดวก กระบวนการ แต่ละตัวจะมี ตารางเลขตอนของตนเอง เก็บอยู่ใน ตารางข้อมูลเฉพาะของกระบวนการ กระบวนการหลายตัว อาจใช้ตอนร่วมกันได้ โดยให้ตัวชี้เลขตอน ชี้ไปยังที่เดียวกัน ในหน่วยความจำจริง ดังรูปที่ 7.21 กระบวนการ p1 และ p2 ใช้หน่วยความของ editor ร่วมกัน



รูปที่ 7.21 แสดงการใช้หน่วยความจำร่วมกันในระบบเซกเมนท์

การใช้ข้อมูลร่วมกันนี้ เป็นระดับตอน ดังนั้นข้อมูลทุกชนิด ที่กำหนดเป็นตอน สามารถใช้ร่วมกันได้โดยไม่จำกัดจำนวนตอนที่จะใช้ร่วมกัน กระบวนการหลายตัวจึงสามารถใช้โปรแกรมร่วมกันได้อย่างสะดวก เช่น การใช้โปรแกรมบรรณาธิกรณ์ (Text Editor) ในระบบแบ่งเวลา (Time Sharing) ซึ่งมีผู้ใช้หลายคนกำลังใช้โปรแกรมนี้อยู่ โปรแกรมนี้อาจประกอบด้วยหลาย ๆ ตอน ซึ่งสามารถใช้ร่วมกันได้ ทำให้ความต้องการเนื้อที่หน่วยความจำจริงโดยรวมลดลงได้อย่างมาก โดยผู้ใช้แต่ละคนใช้โปรแกรมบรรณาธิกรณ์ร่วมกัน แต่มีตอนสำหรับเก็บข้อมูลภายในแยกกันหรืออาจใช้โปรแกรมร่วมกัน เป็นบางส่วนก็ได้ ไม่หมดทั้งโปรแกรม เช่น โปรแกรมย่อยที่เป็นมาตรฐานต่าง ๆ อาจให้ผู้ใช้หลายคน ใช้ร่วมกันในเวลาเดียวกัน แต่ต้องเป็นประเภทอ่านได้เท่านั้น หรือใช้งานได้เท่านั้น ตัวอย่าง โปรแกรมภาษาฟอร์แทรน 2 ชุด อาจใช้โปรแกรมย่อย (Subprogram) sqrt ร่วมกัน

แม้ว่า การใช้โปรแกรมร่วมกันนี้ จะดูง่าย ๆ แต่ก็มีปัญหาเหมือนกัน เพราะในโปรแกรมมักมีคำสั่งอ้างอิงถึงตัวเอง เช่น คำสั่งให้ย้ายไปทำงานต่อ อีกตำแหน่งหนึ่ง ตำแหน่งใหม่นี้ อาจอ้างอิงโดยระบุหมายเลขตอน และระยะจากขอบ ซึ่งหมายเลขตอนนี้ จะเป็นหมายเลขของตอน ที่เป็นคำสั่ง ถ้าเราพยายามใช้ตอนคำสั่งนี้ ร่วมกันหลาย ๆ กระบวนการ กระบวนการ เหล่านั้น จะต้องกำหนด ตอนร่วม ด้วยหมายเลขตอนเดียวกัน ตัวอย่าง ถ้าเราใช้โปรแกรมย่อย (Subprogram) sqrt ร่วมกัน กระบวนการ หนึ่ง กำหนดให้ โปรแกรมนี้เป็น ตอนที่ 4 และอีกกระบวนการ หนึ่งให้เป็นตอนที่ 17 แล้วโปรแกรม sqrt จะอ้างอิงตัวเองอย่างไร เพราะว่ามีโปรแกรมเพียง 1 ชุด ในหน่วยความจำจริง แต่ใช้ร่วมกัน โปรแกรม sqrt จะต้องอ้างอิง ถึงตัวเองเหมือนที่ผู้ใช้ทั้งสองใช้ จึงต้องกำหนดเป็นหมายเลขตอนเดียวกัน ถ้ามีผู้ใช้ร่วมมากคนขึ้น การกำหนดหมายเลขตอนร่วมกันย่อมยากขึ้นเป็นลำดับเช่นกัน

ข้อมูลร่วมประเภท ใช้อ่านเท่านั้น (Read-Only) ห้ามมีตัวชี้ อาจสามารถใช้ร่วมกันได้ โดยใช้หมายเลขตอนต่างกัน หรือ โปรแกรมที่ไม่มีการอ้างอิงตัวเองโดยตรง มีเพียงการอ้างอิงตนเองโดยอ้อม ก็อาจใช้ร่วมกันได้ โดยใช้หมายเลขตอนต่างกัน เช่น คำสั่งย้ายไปทำงานต่อ อีกตำแหน่งหนึ่ง บอกตำแหน่งเป็น ระยะจากบรรทัดคำสั่งปัจจุบัน หรือระยะจากขอบของตอน ที่ระบุหมายเลขในรีจิสเตอร์เพื่อเป็นการหลีกเลี่ยงการอ้างอิงตำแหน่งแบบโดยตรง

การสูญเปล่าพื้นที่ย่อย (Fragmentation)

ตัวจัดตารางการทำงานระยะยาว มีหน้าที่ จัดหาเนื้อที่ ในหน่วยความจำหลัก ให้โปรแกรมของผู้ใช้ทุก ๆ คน ซึ่งก็เหมือนในระบบแบ่งเป็นหน้า ยกเว้นว่า การแบ่งเป็นตอน ขนาดของพื้นที่แต่ละตอนไม่เท่ากันเนื่องจากขนาดของหน้าเท่ากันหมดทุกหน้า ดังนั้นการจัดสรรพื้นที่ จะ

เหมือนกับการจัดสรรพื้นที่แบบขนาดไม่เท่ากัน ซึ่งนิยมใช้แบบแรกเหมาะ (First-Fit) หรือแบบเหมาะสมที่สุด (Best-Fit)

การแบ่งเป็นตอน มักทำให้เกิดการสูญเปล่าพื้นที่ย่อยภายนอก ซึ่งเกิดจาก พื้นที่ว่าง อยู่กระจัดกระจายกัน และแต่ละพื้นที่ มีขนาดเล็กไป สำหรับกระบวนการหนึ่ง ๆ กระบวนการ อาจรอจนมีพื้นที่ว่างพอ หรือระบบอาจบีบอัดหน่วยความจำ เพื่อให้เกิดพื้นที่ว่างต่อเนื่อง ขนาดใหญ่พอเพียง ในกรณีที่พื้นที่ว่างไม่พอ ตัวจัดตารางการทำงานหน่วยประมวลผลกลางอาจรอ หรือข้ามไปดูกระบวนการที่มีระดับความสำคัญ (Priority) ต่ำกว่าก็ได้

ในระบบแบ่งเป็นตอนนี้ ปัญหาการสูญเปล่าพื้นที่ย่อยภายนอก เป็นปัญหามากเพียงไร และการใช้ตัวจัดตารางการทำงานระยะยาวกับการบีบอัดหน่วยความจำจะช่วยได้หรือไม่ ทั้งสองประการนี้ขึ้นอยู่กับขนาดของตอนเป็นหลัก ถ้าแต่ละกระบวนการมีเพียง 1 ตอน จะทำให้ระบบกลายเป็นการจัดการหน่วยความจำแบบแบ่งส่วนไม่เท่ากัน หรือในทางตรงข้าม ถ้าแต่ละกระบวนการถูกแบ่งเป็นตอนย่อย ๆ ตอนละ 1 คำเท่านั้น ทุก ๆ คำ จะมีตอนเป็นของตัวเอง และสามารถย้ายไปไว้ที่ใดก็ได้ในหน่วยความจำ จะไม่มีการสูญเปล่าพื้นที่ย่อยภายนอกเลย แต่ทุก ๆ ตอน หรือทุก ๆ คำ จะมีต้องมีค่าเลขฐานของตนเอง ทำให้สิ้นเปลืองเนื้อที่ในการเก็บเป็น 2 เท่า ถ้าเราขยายขนาดตอนให้โตขึ้น แต่มีขนาดเท่า ๆ กัน ก็จะกลายเป็นระบบแบ่งเป็นหน้า โดยทั่วไปแล้ว ถ้าตอนมีขนาดเฉลี่ยเล็ก พื้นที่ย่อยภายนอกก็จะมีขนาดเล็กด้วยโดยการเปรียบเทียบ สมมุติว่าเราพยายามเรียงกระเป๋าเสื้อผ้าหลาย ๆ ใบลงในท้ายรถ และดูท่าทางจะใส่ไม่หมด แต่ถ้าเราเปิดกระเป๋าแล้วเทข้าวของลงไปไว้ในท้ายรถแทน ก็จะสามารถใส่ลงได้หมดเพราะว่าเมื่อแบ่งเป็นหลาย ๆ ตอน แต่ละตอนย่อมมีขนาดเล็กกว่า 1 กระบวนการ ซึ่งน่าจะจัดสรรลงในหน่วยความได้ง่ายกว่า

การจัดการแบบผสมของเพจและเซกเมนท์

การจัดการหน่วยความจำทั้งแบบแบ่งเป็นหน้า และแบบแบ่งเป็นตอน ต่างก็มีข้อดีและข้อเสียต่างๆ กันเช่น ตัวประมวลผล 2 ชนิด คือ ตัวประมวลผล ในตระกูล 68000 ของโมโตโรล่า จัดการหน่วยความจำเป็นแบบเส้นตรง และตัวประมวลผลในตระกูล 8086 ของอินเทล ใช้วิธีแบ่งเป็นตอน ทั้งสองชนิดกำลังพัฒนาไปสู่การออกแบบร่วมกันเป็นแบบผสม ระหว่างวิธีแบ่งเป็นหน้า และแบ่งเป็นตอนซึ่งจะช่วยให้ ได้ผลดีของทั้งสองวิธีรวมกัน ระบบปฏิบัติการมัลติก (MULTICS) เป็นระบบแรก ที่ใช้ ทั้ง 2 วิธีรวมกันได้อย่างเหมาะสม แต่ตัวระบบเองไม่ค่อยได้รับความนิยมนัก ซึ่งเป็นเพราะสาเหตุอื่น อีกหลายประการ

ในระบบปฏิบัติการมัลติก ตำแหน่งทางกรรก ประกอบด้วย หมายเลขตอน 18 บิต และระยะจากขอบ 16 บิต รวมเป็นตำแหน่งอ้างอิง 34 บิต ซึ่งค่อนข้างจะใหญ่โตมาก แต่การเก็บตาราง

เลขตอนไม่เป็นปัญหา เพราะการแบ่งเป็นตอน สามารถใช้จำนวนตอนไม่เท่ากันได้ ในแต่ละกระบวนการ จำนวนตอนเก็บไว้ใน STLR จึงไม่จำเป็นจะต้องมีช่องว่าง ๆ ในตารางเลขตอน ให้เสียเนื้อที่

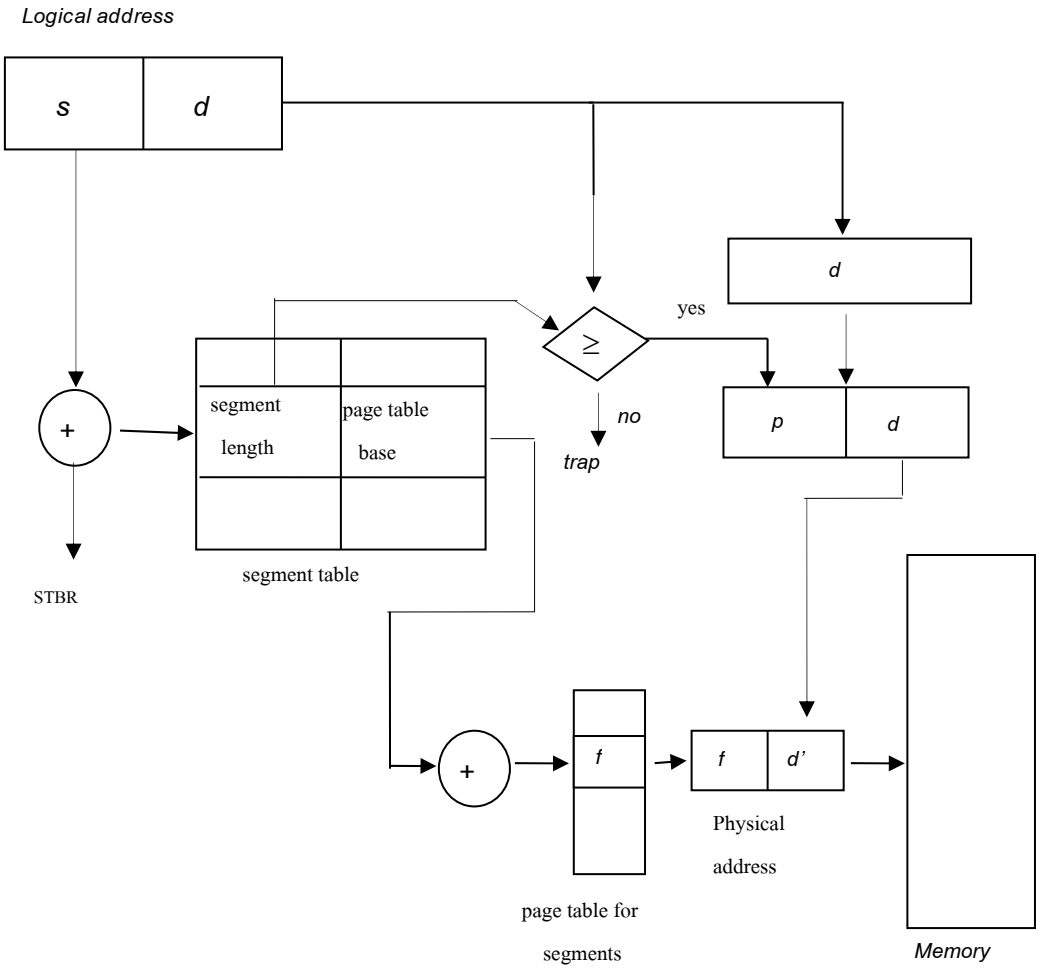
แต่อย่างไรก็ตาม เนื้อที่ตอนขนาด 64 K คำ แบบ16 บิต อาจทำให้เกิด การสูญเปล่าพื้นที่ย่อยภายนอกได้มาก ถึงแม้จะไม่เป็นปัญหามากนัก แต่การจัดหาเนื้อที่ โดยใช้วิธีแรกเหมาะหรือเหมาะสมที่สุด ก็จะต้องใช้เวลานาน ทำให้ระบบสูญเสียเนื้อที่ไปบางส่วน เนื่องจากการสูญเปล่าพื้นที่ย่อยภายนอกและยังต้องเสียเวลาไป ในการจัดหาเนื้อที่ว่างอีกด้วย

การแก้ปัญหาจึงทำโดย แบ่งตอนออกเป็นหน้า ๆ (Paged Segmentation) การแบ่งหน้าทำให้การสูญเปล่าพื้นที่ย่อยภายนอกหมดไป และการจัดสรรเนื้อที่ว่าง ก็จะทำได้ง่ายมากด้วย เพราะถ้ามีหน้าว่างที่ใด ก็สามารถจัดสรรให้ได้ทันที ไม่จำเป็นต้องเป็นเนื้อที่ ว่างต่อเนื่องกันอีกต่อไป ดังรูปที่ 7.22 จะสังเกตเป็น ส่วนที่แตกต่างจาก ระบบแบ่งเป็นตอนอย่างเดียวว่าในตารางเลขตอน จะมีค่าตำแหน่งฐานของตารางเลขหน้า (Page Table Base)

แทนที่จะเป็นตำแหน่งฐาน หรือตำแหน่งเริ่มต้น ในหน่วยความจำจริง ระยะจากขอบของตอน (Segment Offset) แบ่งเป็น 2 ส่วน 6 บิต แรก ใช้เป็นหมายเลขหน้า อีก 10 บิต หลังใช้เป็นระยะจากขอบหน้า เราใช้หมายเลขหน้านี้ ชี้ไปยังตารางเลขหน้า เพื่อดูตำแหน่งหน้าจริง แล้วนำมาบวกกับ ระยะจากขอบหน้าเป็นตำแหน่งจริง ในหน่วยความจำหลัก

เราจึงต้องมีที่เก็บตารางเลขหน้า สำหรับแต่ละตอน ขนาดของตารางเลขหน้า ไม่จำเป็นต้องโต ถึง 10 ช่อง แบบขนาด 6 บิต เราอาจจัดสรร เฉพาะเท่าที่ต้องการได้ เพราะขนาดของตอน แต่ละตอนไม่เท่ากัน ในหน้าสุดท้าย ของแต่ละตอนมักจะไม่ได้ใช้จนเต็ม ซึ่งก็คือการสูญเปล่าพื้นที่ย่อยภายใน โดยเฉลี่ยเท่ากับครึ่งหนึ่งของขนาดของตอน ในตัวอย่างนี้ คือ $(0.5) \times 2^{10} = 512$ คำ จะเห็นได้ว่า เราพยายาม ลดการสูญเปล่าพื้นที่ย่อยภายนอก แต่ก็เกิดการสูญเปล่าพื้นที่ย่อยภายในขึ้นแทน

ถึงแม้ว่า วิธีการจัดการหน่วยความจำแบบผสมที่กล่าวมา ในระบบปฏิบัติการมัลติกจะแลดูง่าย ๆ แต่ในความเป็นจริงการที่ระบบใช้หมายเลขตอนถึง 18 บิต ทำให้สามารถมีตอนได้ถึง 262,144 ตอน ใช้เนื้อที่ 256K นับเป็นขนาดตารางเลขตอนที่ใหญ่มาก ระบบมัลติกจึงแบ่งตารางเลขตอนนี้ออกเป็นหน้าอีก ดังนั้นการอ้างอิงตำแหน่งในระบบมัลติกจะต้องใช้หมายเลขตอน เพื่อหาหมายเลขหน้า จากตารางเลขหน้าของตอนนั้น ๆ เมื่อให้หมายเลขหน้าของตอน ผนวกกับระยะจากขอบหน้า ในหมายเลขตอน จะได้ตำแหน่งช่อง ในตารางเลขหน้า ซึ่งชี้ไปยังตารางเลขหน้า



รูปที่ 7.22 การแบ่งหน่วยความจำหลักแบบผสมผสมของเพจและเซกเมนท์

ของตอนอีกที จากตารางเลขหน้าครั้งหลังนี้ ก็จะได้หมายเลขหน้าจริง ในหน่วยความจำ บวกกับค่าระยะจากขอบในการอ้างอิงตำแหน่งชุดแรก จะเป็นตำแหน่งจริงในที่สุด จากรูปที่ 7.22 ในการเข้าถึงหน่วยความจำ ซีพียูจะส่ง logical address ซึ่งมี 2 ส่วรคือเซกเมนท์ s และ ออบเซต d โดยที่ s เป็นตัวชี้ไปยัง STBR (Segment Table Base Register) ซึ่งเป็นตารางรีจีสเตอร์ฐานของแต่ละเซกเมนท์ ทำการตรวจสอบขนาดของเซกเมนท์ต้องมากกว่าค่าของรีจีสเตอร์ฐานของเซกเมนท์นั้น จะได้เลขหน้า (Page) ของออบเซต d นำมารวมกับค่าของรีจีสเตอร์ฐานของหน้าในตารางรีจีสเตอร์ฐานของแต่ละเซกเมนท์ ซึ่งเป็นตำแหน่งที่อยู่ของหน้าของของเซกเมนท์ที่ตำแหน่งออบเซต d เป็นตำแหน่งในหน่วยความจำหลักที่เข้าถึง

ปฏิบัติการที่ 7

1. ให้นักศึกษาแต่ละกลุ่มใช้ VMWARE ทำการจำลองเครื่องคอมพิวเตอร์ และทำการติดตั้งระบบปฏิบัติการที่รับมอบหมายไว้จากปฏิบัติการที่ 1
2. ศึกษาการใช้คำสั่งการทำงานพื้นฐาน เช่นคำสั่งเกี่ยวกับการจัดการไฟล์การคัดลอกไฟล์ การลบไฟล์

คำถามท้ายบท

1. จงอธิบายวิธีการนำโปรแกรมลงสู่หน่วยความจำซึ่งมีหลายวิธีแต่ละวิธีเหมือนกันหรือต่างกันอย่างไร
2. จงอธิบายวิธีการจัดการเศษของพื้นที่ว่างในหน่วยความจำหลัก (Fragmentation) แต่ละวิธีดำเนินการอย่างไร
3. วิธีการนำข้อมูลไปใส่ในพื้นที่ช่องว่างของหน่วยความจำมีกี่วิธีแต่ละวิธีจัดการอย่างไร มีข้อดีข้อเสียอย่างไร
4. อธิบายวิธีการจัดสรรเนื้อที่หน่วยความจำหลักแบบหลายส่วน (Multiple-Partition Allocation)
5. อธิบายวิธีการจัดการหน่วยความจำหลักแบบเพจมีวิธีจัดการอย่างไรมีข้อดีข้อเสียอย่างไร
6. อธิบายวิธีการจัดการหน่วยความจำหลักแบบเซกเมนท์มีวิธีจัดการอย่างไรมีข้อดีข้อเสียอย่างไร
7. อธิบายวิธีการป้องกันและการใช้ตอนร่วมกันมีวิธีจัดการอย่างไร
8. อธิบายวิธีการจัดการหน่วยความจำหลักแบบการจัดการแบบผสมของเพจและเซกเมนท์
9. แสดงคิดเห็นว่าการจัดการหน่วยความจำแบบใดจะสามารถใช้งานได้มีประสิทธิภาพที่สุดและมีการจัดการอย่างไร