บทที่ 6

การจัดการหน่วยความจำหลัก

หน่วยความจำหลักเป็นทรัพยากรที่สำคัญมากในระบบคอมพิวเตอร์ หากไม่มีหน่วยความจำหลัก คอมพิวเตอร์จะไม่สามารถทำงานได้เลย หน่วยความจำหลักมีการทำงานที่ซับซ้อนและใช้ประโยชน์ได้อย่างหลากหลาย เช่นใช้ในการเก็บข้อมูลของกระบวนการที่ต้องการจะประมวลผล, หรือกระบวนการของระบบปฏิบัติการเอง เช่นส่วนแกนกลาง นอกจากนั้นยังสามารถใช้เป็นส่วนติดต่อกับหน่วยนำข้อมูลเข้า/ออก โดยอุปกรณ์หน่วยนำข้อมูลเข้า/ออกจะนำข้อมูลมาวางหรืออ่านข้อมูลจากส่วนนี้ William Stallings (1992, หน้า 265) ได้กล่าวว่าในระบบที่ทำงานทีละโปรแกรม (uniprogramming) จะมีการแบ่งหน่วยความจำออกเป็น 2 ส่วน ได้แก่ส่วนของระบบปฏิบัติการและส่วนของโปรแกรมของผู้ใช้ และสำหรับระบบที่ทำงานแบบหลายโปรแกรม ส่วนของผู้ใช้ในหน่วยความจำจะถูกแบ่งออกเป็นหลายๆส่วนและทำงานแบบมีพลวัต (dynamic) ซึ่งจะถูกจัดการด้วยระบบปฏิบัติการในส่วนที่เรียกว่าส่วนจัดการหน่วยความจำหลัก

การจัดการหน่วยความจำหลักในระบบการทำงานแบบหลายโปรแกรม เป็นสิ่งที่สำคัญมาก เช่นหากมีกระบวนการอยู่ในหน่วยความจำหลักเพียงเล็กน้อย หากกระบวนการทั้งหมดอยู่ในช่วงการรอคอยการทำงานของหน่วยนำข้อมูลเข้า/ออก ทำให้หน่วยประมวลผลว่างงาน การออกแบบการจัดการหน่วยความจำหลักที่ดีจะต้องให้ใช้หน่วยความจำได้อย่างคุ้มค่า การเลือกรูปแบบของการจัดการหน่วยความจำหลักขึ้นอยู่กับหลายปัจจัย โดยเฉพาะอย่างยิ่งการออกแบบฮาร์ดแวร์ของระบบ ซึ่งการเลือกใช้อัลกอริทึมจะต้องดูว่าเหมาะสมกับฮาร์ดแวร์นั้นหรือไม่

6.1 ความรู้พื้นฐาน

จากที่ได้กล่าวมาแล้วว่าคอมพิวเตอร์กำเนิดมาจาก จอห์น วอน นอยย์แมน ซึ่งประกอบไปด้วยหน่วยความจำหลักที่เก็บข้อมูลและโปรแกรมที่ต้องการประมวลผลที่หน่วยประมวลผลกลาง ข้อมูลจะรับคำสั่งโดยนำมาจากหน่วยความจำหลัก นำไปคำนวณในหน่วยคณิตศาสตร์และตรรกะ ผลลัพธ์ที่ได้วางไว้ที่รีจิสเตอร์ และนำกลับไปไว้ที่หน่วยความจำหลักและอาจส่งไปที่อุปกรณ์หน่วยนำข้อมูลเข้า/ออกต่อไป วัตถุประสงค์ของส่วนจัดการหน่วยความจำหลักก็คือบริหารการใช้งานหน่วยความจำหลักรวมทั้งการเคลื่อนย้ายโปรแกรมและข้อมูลไปมาระหว่างหน่วยความจำหลักและหน่วยความจำสำรอง

6.1.1 ความต้องการพื้นฐานของหน่วยความจำหลัก

จากการศึกษาของ Gary Nutt (2002, หน้า 326) พบว่าหน่วยความจำหลักและหน่วยประมวลผลกลางเป็นทรัพยากรพื้นฐานของกระบวนการทั้งหมด หากไม่มีหน่วยความจำหลัก กระบวนการจะไม่มีที่เก็บข้อมูลและโปรแกรมที่ใช้ในการประมวลผล ความต้องการพื้นฐานสำหรับหน่วยความจำหลักมี 3 ประการ ดังนี้

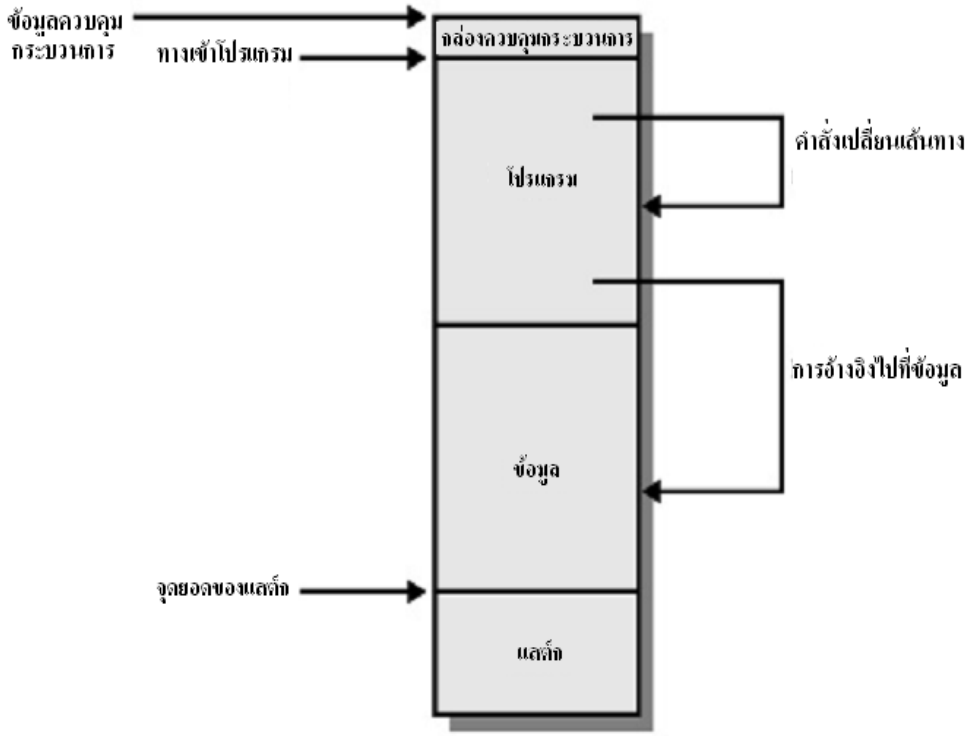
1) เวลาในการเข้าถึงหน่วยความจำหลักจะต้องใช้เวลาน้อยที่สุดเท่าที่จะทำได้ ซึ่งสิ่งนี้ต้องการการออกแบบฮาร์ดแวร์และซอฟท์แวร์ที่ดี
2) หน่วยความจำหลักจะต้องมีขนาดใหญ่ที่สุดเท่าที่จะทำได้ ด้วยการใช้หน่วยความจำเสมือน, ฮาร์ดแวร์ และซอฟท์แวร์ จะทำให้หน่วยความจำหลักมีมากเกินความต้องการ
3) หน่วยความจำหลักจะต้องมีการคุ้มการลงทุน โดยต้นทุนของหน่วยความจำหลักจะต้องมีปริมาณร้อยละเพียงเล็กน้อย หากเทียบกับคอมพิวเตอร์ทั้งระบบ

ในความเป็นจริงมีความเป็นไปได้ที่จะสร้างหน่วยความจำให้หน่วยประมวลผลกลางเข้าถึงได้ด้วยระดับความเร็วเท่าเทียมกัน หน่วยความจำนั้นได้แก่รีจิสเตอร์ แต่การสร้างรีจิสเตอร์มีต้นทุนสูงมาก หน่วยประมวลผลกลางทุกวันนี้มีจำนวนรีจิสเตอร์น้อยกว่า 100 ตัว ซึ่งเทคโนโลยีทุกวันนี้เวลาในการเข้าถึงหน่วยความจำหลักสูงกว่ารีจิสเตอร์ตั้งแต่ 1.5 ถึง 4 เท่า แต่อย่างไรก็ตามหน่วยความจำหลักกลับมีจำนวนมากกว่ารีจิสเตอร์มากกว่า 1 ล้านเท่า โดยมีมากกว่า 100 MB ขึ้นไป

สำหรับกระบวนการนั้น ประกอบไปด้วยพื้นที่แอดเดรส ซึ่งเมื่อต้องการการประมวลผลจะมีการเชื่อมโยงเพื่อให้เข้าไปจองพื้นที่ในหน่วยความจำหลัก ดังรูปที่ 1

รูปที่ 1 การเชื่อมโยงของพื้นที่แอดเดรสเพื่อให้เข้าไปจองพื้นที่ในหน่วยความจำหลัก
ที่มา : Gary Nutt, 2002, หน้า 327

สำหรับกระบวนการแต่ละกระบวนการ เมื่อต้องการประมวลผลจะต้องนำเข้าไปที่หน่วยความจำหลัก รูปที่ 2 แสดงภาพของกระบวนการเมื่ออยู่ในหน่วยความจำหลัก



รูปที่ 2 ภาพของกระบวนการเมื่ออยู่ในหน่วยความจำหลัก
ที่มา : William Stallings, 1992, หน้า 267

ส่วนจัดการหน่วยความจำหลัก จะต้องทำหน้าที่ 3 ส่วน ได้แก่

1) จองพื้นที่ในหน่วยความจำหลักให้กับกระบวนการ
2) เชื่อมโยงพื้นที่แอดเดรสของกระบวนการไปสู่ส่วนพื้นที่ที่จองไว้ในหน่วยความจำหลัก
3) ลดเวลาการเข้าถึงโดยการใช้หน่วยความจำให้คุ้มค่ามากที่สุด

สำหรับการศึกษาของ William Stallings (1992, หน้า 266) ได้กล่าวถึงหน้าที่ด้านอื่นๆของส่วนจัดการหน่วยความจำหลัก ดังนี้

4) การย้ายตำแหน่ง (relocation)
5) การป้องกัน

6) การแบ่งปัน

7) การจัดระเบียบในเชิงตรรกะ (logical organization)

8) การจัดระเบียบในทางกายภาพ (physical organization)

6.1.2 การเชื่อมโยงของพื้นที่แอดเดรสไปสู่หน่วยความจำหลัก

Gary Nutt (2002, หน้า 327) ได้กล่าวว่าในมุมมองของผู้เขียนโปรแกรมที่มีต่อระบบได้แก่ภาษาโปรแกรม และส่วนเชื่อมต่อกับเวลาทำงาน (run time interface) และ Silberchatz Galvin (1999, หน้า 240) ได้กล่าวถึงการเชื่อมโยงแอดเดรส (address binding) ว่าระบบปฏิบัติการสามารถนำกระบวนการไปวางไว้ที่ใดของหน่วยความจำหลักก็ได้ ดังนั้นในการอ้างถึงแอดเดรสของกระบวนการใดๆจะมีขั้นตอนหลายขั้นตอน เริ่มตั้งแต่การแปลรหัสโปรแกรมต้นฉบับ อาจมีการอ้างถึงแอดเดรสด้วยการใช้ลาเบล (label) ต่างๆเช่น COUNT เป็นต้น เมื่อตัวแปลภาษา (compiler) ทำการแปลก็อาจจะเชื่อมโยง COUNT ให้กลายเป็นแอดเดรสแบบอ้างอิงเช่น 14 ไบท์จากจุดเริ่มต้นของหน่วยโปรแกรม จากนั้นตัวเชื่อมโยง (linker) และตัวนำเข้า (loader) ก็จะแปลงแอดเดรสอ้างอิงนี้ให้กลายเป็นแอดเดรสที่แท้จริงเช่นอาจได้แอดเดรสที่ 74014 ซึ่งขั้นตอนการเชื่อมโยงแอดเดรสของโปรแกรมใดๆสามารถแบ่งเป็นขั้นตอนต่างๆได้ดังนี้

6.1.2.1 การคอมไพล์ (compile time)

ในการพัฒนาโปรแกรมต่างๆนั้น ก็จะมีการเขียนรหัสโปรแกรมต้นฉบับแล้วนำไปแปลงเป็นภาษาที่ใกล้เคียงภาษาเครื่อง เช่นภาษาแอสเซมบลี ซึ่งจะใช้โปรแกรมตัวแปลภาษาทำหน้าที่ในการแปลง ในขั้นตอนนี้แอดเดรสที่ได้จากการแปลจะเป็นแอดเดรสที่สามารถย้ายตำแหน่งได้ (relocatable address) นอกจากนั้นแล้วขั้นตอนนี้จะมีการสร้างแอดเดรสให้กับตัวแปรประเภทต่างๆเช่นสำหรับการเขียนโปรแกรม หากมีการใช้ตัวแปรประเภทตายตัว (static variable) การกำหนดแอดเดรสจะกำหนดให้อยู่ในส่วนของเซ็กเมนท์เก็บข้อมูล (data segment) แต่หากเป็นตัวแปรแบบทั่วไป หรือตัวแปรอัตโนมัติ (automatic variable) แอดเดรสที่เกิดขึ้นจะมีการใช้งานแสต็ก โดยจะใช้เมื่ออยู่ในเวลาทำงาน (run time) การกำหนดแอดเดรสของเซ็กเมนท์เก็บข้อมูลของทั้งโปรแกรมจะยังไม่รู้ค่าในขั้นตอนนี้ จนกว่าจะผ่านการเชื่อมโยง สำหรับแอดเดรสของทางเข้าโปรแกรม ตัวแปลภาษาไม่สามารถกำหนดได้เช่นกัน เนื่องจากระบบปฏิบัติการสามารถย้ายตำแหน่งไปที่ใดก็ได้ในหน่วยความจำหลัก หรือหากโปรแกรมมีการเรียกการทำงานของห้องสมุดคำสั่ง แอดเดรสที่เชื่อมโยงไปยังห้องสมุด

คำสั่งก็จะกำหนดโดยการเชื่อมโยงเช่นกัน รูปที่ 3 แสดงตัวอย่างของรหัสโปรแกรมต้นฉบับที่เขียนด้วยภาษาซี และรูปที่ 4 แสดงการแปลรหัสโปรแกรมต้นฉบับให้เป็นแอดเดรสที่สามารถย้ายตำแหน่งได้

รูปที่ 3 : ตัวอย่างของรหัสโปรแกรมต้นฉบับที่เขียนด้วยภาษาซี
ที่มา : Gary Nutt, 2002, หน้า 330

รูปที่ 4 การแปลรหัสโปรแกรมต้นฉบับให้เป็นแอดเดรสที่สามารถย้ายตำแหน่งได้
ที่มา : Gary Nutt, 2002, หน้า 331

จากรูปที่ 3 จะเห็นตัวอย่างรหัสโปรแกรมต้นฉบับที่เขียนด้วยภาษาซี ในตอนต้นของโปรแกรมมีการประกาศตัวแปรเป็นแบบตายตัวจำนวนหนึ่ง จากนั้นเป็นการประการส่วนของฟังก์ชั่นที่ชื่อว่า proc_a โดยรับค่าพารามิเตอร์เป็นจำนวนเต็มผ่านทางตัวแปรชื่อว่า arg ฟังก์ชั่นนี้หากทำงานเสร็จจะคืนค่ากลับเป็นประเภทจำนวนเต็ม ภายในฟังก์ชั่นมีการกำหนดค่าให้กับตัวแปร gVar ให้เป็น 7 และมีการเรียกการทำงานของฟังก์ชั่นย่อยชื่อว่า put_record โดยส่งค่า gVar ไปเป็นพารามิเตอร์

รูปที่ 4 เป็นการแสดงส่วนของแอดเดรสที่สามารถย้ายตำแหน่งได้ ซึ่งสร้างมาจากโปรแกรมแปลภาษา จะเห็นว่าแอดเดรสเริ่มต้นจะเป็นแอดเดรส 0000 ซึ่งเป็นแอดเดรสจริงในเชิงตรรกะ ส่วนทางเข้าฟังก์ชั่น proc_a อยู่ในแอดเดรสที่ 0008 สำหรับตัวแปร gVar ก็จะเก็บไว้ที่แอดเดรส 0036 เป็นต้นไป

โดยจะเก็บค่าของตัวแปรไว้ในตารางสัญลักษณ์ที่เริ่มที่แอดเดรส 0600 ในการเรียกการทำงานของฟังก์ชั่น put_record จะใช้การอ้างอิงไปสู่ภายนอก เนื่องจากเป็นฟังก์ชั่นที่อยู่ในห้องสมุด โดยกระบวนการจะมีตารางอ้างอิงไปสู่ภายนอก ซึ่งตารางอยู่ที่แอดเดรส 0400

6.1.2.1 ขั้นตอนการเชื่อมโยง (link time)

ในขั้นตอนการเชื่อมโยง เซ็กเมนท์ของข้อมูลจากที่ต่างๆที่ได้จากขั้นตอนการแปลจะถูกเชื่อมโยงเข้าหากันกลายเป็นเซ็กเมนท์เดียวกัน หลังจากนั้นส่วนการเชื่อมโยงก็จะแปลงแอดเดรสทั้งหมดให้เป็นแอดเดรสจริงในเชิงตรรกะ (logical address) ซึ่งยังไม่ใช่แอดเดรสจริงในทางกายภาพ (physical address) ในหน่วยความจำหลัก รูปที่ 5 แสดงการเชื่อมโยงแอดเดรสในขั้นตอนการเชื่อมโยง

รูปที่ 5 การเชื่อมโยงแอดเดรสที่สามารถย้ายตำแหน่งได้เป็นแอดเดรสจริงเชิงตรรกะ
ที่มา : Gary Nutt, 2002, หน้า 332

จากรูปที่ 5 จะพบว่าแอดเดรสเริ่มต้นของโปรแกรมเปลี่ยนแปลงไปเป็นจำนวน 1000 โดยทางเข้าของฟังก์ชั่นเริ่มที่แอดเดรส 1008 ดังนั้นแอดเดรสในส่วนอื่นๆก็จะถูกเลื่อนไป 1000 ตำแหน่งเช่นกัน และด้านท้ายของโปรแกรมส่วนของฟังก์ชั่น put_record ถูกเพิ่มเข้ามาในโปรแกรมและได้รับการกำหนดแอดเดรสให้เป็น 2334 และส่วนของตารางสัญลักษณ์ก็จะมีการย้ายตำแหน่งเช่นกัน ซึ่งแอดเดรสใหม่ทั้งหมดนี้ ได้มาจากขั้นตอนการเชื่อมโยงโปรแกรม

6.1.2.2 ขั้นตอนการนำเข้า

ขั้นตอนนี้จะนำแอดเดรสจริงในเชิงตรรกะจากขั้นตอนที่แล้ว มาแปลงเป็นแอดเดรสจริงในทางกายภาพ โดยการบวก เนื่องจากแอดเดรสจริงในเชิงตรรกะจะเริ่มต้นที่แอดเดรส 0 ดังนั้นขั้นตอนการนำเข้าระบบปฏิบัติการจะเลือกตำแหน่งเริ่มต้นที่จะนำกระบวนการเข้าไปวางในหน่วยความจำหลัก ดังนั้นระบบปฏิบัติการเพียงแต่บวกแอดเดรสจริงในทางกายภาพที่เป็นแอดเดรสเริ่มต้นเข้าไปที่แอดเดรสจริงในเชิงตรรกะ ก็จะได้แอดเดรสจริงในทางกายภาพของโปรแกรมทั้งหมด รูปที่ 6 แสดงการกำหนดแอดเดรสจริงในทางกายภาพให้กับโปรแกรม

รูปที่ 6 การกำหนดแอดเดรสที่จริงในทางกายภาพเมื่อถูกนำเข้า
ที่มา : Gary Nutt, 2002, หน้า 333

จากรูปจะพบว่ามีการบวกแอดเดรสที่ได้จากรูปที่ 5 เข้ากับ 4000 ซึ่งเป็นแอดเดรสเริ่มต้นของกระบวนการ และเป็นแอดเดรสจริงในทางกายภาพ ดังนั้นแอดเดรสทั้งหมดก็จะต้องถูกบวกเข้ากับ 4000 นี้เช่นกัน

6.1.2.3 ขั้นตอนการทำงานของโปรแกรม

เมื่อนำเข้าโปรแกรมสู่หน่วยความจำหลักแล้ว หน่วยประมวลผลกลางก็สามารถรับคำสั่งเข้ามาทำงานได้ โปรแกรมจะเริ่มการประมวลผลตั้งแต่คำสั่งแรก และในบางครั้งโปรแกรมอาจจะมีการเรียกใช้งานห้องสมุดคำสั่งแบบพลวัต (dynamic library) ซึ่งเป็นห้องสมุดคำสั่งที่ถูกนำเข้ามาขณะที่โปรแกรมกำลังทำงานซึ่งอาจต้องมีการเชื่อมโยงแอดเดรสของโปรแกรมในแบบพลวัตเช่นกัน

การเชื่อมโยงแอดเดรสในขั้นตอนต่างๆแสดงได้ดังรูป 7

รูปที่ 7 การเชื่อมโยงแอดเดรสในขั้นตอนต่างๆ

ที่มา : Silberchatz Galvin, 1999, หน้า 241

6.2 การจองพื้นที่ในหน่วยความจำหลัก

เมื่อโปรแกรมผ่านขั้นตอนการแปลและเชื่อมโยงแล้ว ก็จะได้ชุดของแอดเดรสจริงในเชิงตรรกะ จากนั้นจะมีการจองพื้นที่ในหน่วยความจำหลักให้กับกระบวนการ ดังนั้นพื้นที่แอดเดรสจะถูกเชื่อมโยงเข้ากับพื้นที่จริงในหน่วยความจำหลัก เพื่อการประมวลผลของกระบวนการ (Gary Nutt, 2002, หน้า 335) ดังนั้นก่อนที่จะสามารถเชื่อมโยงแอดเดรสจริงได้จะต้องมีพื้นที่ว่างในหน่วยความจำหลักก่อน ส่วนจัดการหน่วยความจำหลักในระบบงานแบบหลายโปรแกรม มีหน้าที่จัดสรรหน่วยความจำหลักให้กับกระบวนการต่างๆด้วยหลักการของการแบ่งพื้นที่โดยการสลับกันใช้พื้นที่ (space multiplex)

สมมติว่าระบบปฏิบัติการรองรับการทำงานแบบหลายโปรแกรมได้ครั้งละ 4 โปรแกรม ส่วนจัดการหน่วยความจำหลักก็จะแบ่งพื้นที่ในหน่วยความจำหลักออกเป็น 4 ส่วนเพื่อเก็บกระบวนการทั้งหมด ดังรูปที่ 8

รูปที่ 8 การทำงานแบบหลายโปรแกรมได้ครั้งละ 4 โปรแกรม

ที่มา : Gary Nutt, 2002, หน้า 335

เนื่องจากส่วนของระบบปฏิบัติการต้องการพื้นที่ของตนเอง เพื่อเก็บรหัสคำสั่งและตารางต่างๆ ดังนั้นหน่วยความจำหลักจะแบ่งออกเป็น 5 ส่วน มีกลยุทธ์จำนวนมากที่สามารถใช้กับการจองพื้นที่ในหน่วยความจำ เช่น ใช้การแบ่งพื้นที่ในหน่วยความจำออกเป็นพื้นที่ย่อยขนาดเท่าๆกัน หรือการแบ่งเป็นพื้นที่ย่อยในขนาดที่สามารถปรับเปลี่ยนได้ ซึ่งปัญหาที่จะต้องคำนึงถึงก็คือปัญหาของการเกิดเศษพื้นที่ (fragmentation) ได้แก่พื้นที่ในหน่วยความจำหลักที่ไม่สามารถจองได้ ในทางอุดมคติแล้วควรสามารถจองพื้นที่ในหน่วยความจำหลักได้ทุกๆไบท์ แต่ในทางปฏิบัตินั้นไม่สามารถทำได้เนื่องจากข้อจำกัดของวิธีการจองพื้นที่

6.2.1 การแบ่งพื้นที่แบบตายตัว (fixed partitioning)

William Stallings (1992, หน้า 271) ได้กล่าวถึงการจองพื้นที่ด้วยการแบ่งพื้นที่แบบตายตัว ว่าระบบปฏิบัติการจะแบ่งหน่วยความจำหลักออกเป็นส่วนต่างๆซึ่งมีขนาดที่แน่นอนตายตัว ซึ่งแต่ละพื้นที่การแบ่ง (partition) อาจมีขนาดเท่ากันทั้งหมดหรือไม่เท่ากันก็ได้ ดังรูปที่ 9

รูป 9ก. รูป 9ข.

รูปที่ 9 การแบ่งพื้นที่แบบตายตัวของหน่วยความจำหลักขนาด 4 MB
ที่มา : William Stallings, 1992, หน้า 272

สำหรับการแบ่งพื้นที่โดยมีขนาดไม่เท่ากัน กระบวนการที่เล็กกว่าหรือเท่ากับขนาดของพื้นที่การแบ่งจะสามารถจองในพื้นที่การแบ่งนั้นได้ โดยนำเข้าไปที่พื้นที่การแบ่งไหนก็ได้ที่ยังว่างอยู่ หากพื้นที่การแบ่งมีการจองครบหมดแล้ว ส่วนจัดการหน่วยความจำหลักจะสลับบางกระบวนการออกจากหน่วยความจำหลักไปไว้ที่หน่วยความจำสำรอง เพื่อให้มีพื้นที่ที่ว่างสำหรับการใช้งาน ปัญหาของการจองพื้นที่แบบตายตัวได้แก่

1) กระบวนการอาจมีขนาดใหญ่กว่าพื้นที่การแบ่ง ซึ่งผู้เขียนโปรแกรมอาจต้องออกแบบโปรแกรมให้มีขนาดเหมาะสมกับพื้นที่การแบ่ง โดยแยกส่วนการทำงานและใช้การนำเข้าส่วนของกระบวนการในภายหลังเมื่อต้องการการประมวลผล
2) อาจเกิดปัญหาเศษพื้นที่ภายใน (internal fragmentation) ซึ่งเกิดจากมีพื้นที่ว่างที่ใช้งานไม่ได้ภายในขอบเขตของกระบวนการ ซึ่งเกิดจากการจองพื้นที่ ที่ต้องจองเป็นจำนวนของพื้นที่การแบ่ง ซึ่งเกิดความไม่ลงตัวกับขนาดของกระบวนการ เช่นจากรูปที่ 9 หากกระบวนการมีขนาด 300 KB ก็ต้องจองจำนวน 2 พื้นที่การแบ่ง แต่ใช้งานได้ไม่ครบทั้งหมดจำนวน 512 KB
3) การจองพื้นที่ของกระบวนการมีจำนวนที่จำกัด เท่ากับจำนวนของพื้นที่การแบ่งที่กำหนด

ปัญหาทั้ง 2 ข้อสามารถแก้ไขให้ลดน้อยลงได้ แต่ไม่ใช่การแก้ไขแบบสมบูรณ์แบบ โดยการใช้พื้นที่การแบ่งที่มีขนาดไม่เท่ากัน ดังรูปที่ 9ข. ซึ่งการจองพื้นที่ให้กับกระบวนการจะดูจากขนาดของพื้นที่การแบ่งที่เหมาะสม

6.2.1.1 อัลกอริทึมการจอง สำหรับอัลกอริทึมการจองของการแบ่งพื้นที่แบบตายตัวไม่มีความซับซ้อนมากนัก โดยจะตรวจสอบว่ามีพื้นที่การแบ่งว่างหรือไม่ หากเป็นแบบพื้นที่การแบ่งเท่ากันทั้งหมด กระบวนการก็สามารถนำเข้าไปในพื้นที่การแบ่งนั้นได้ หากทุกพื้นที่การแบ่งถูกจองเต็มหมด จะต้องนำกระบวนการใดกระบวนการหนึ่งสลับออกไปไว้ที่ฮาร์ดดิสก์ เพื่อสร้างพื้นที่ว่าง สำหรับการแบ่งพื้นที่ขนาดไม่เท่ากัน มี 2 วิธีในการจองพื้นที่ วิธีแรกคือค้นหาขนาดของพื้นที่การแบ่งที่เล็กที่

สุดที่สามารถบรรจุกระบวนการได้ ซึ่งวิธีนี้แต่ละพื้นที่การแบ่งจะมีคิวของตนเอง ดังรูปที่ 10ก. แต่การจัดคิวในลักษณะนี้ก็มีข้อเสีย เช่นจากรูปที่ 9ข. หากสมมติว่าไม่มีกระบวนการที่มีขนาดระหว่าง 768K ถึง 1M เลย ก็จะทำให้พื้นที่การแบ่งนี้ไม่ถูกใช้งานเลย ทั้งที่ในความเป็นจริงแล้วกระบวนการขนาดเล็กก็สามารถใช้งานในพื้นที่นี้ได้ วิธีการแก้ไขสามารถทำได้โดยใช้คิวเพียงคิวเดียว เมื่อมีกระบวนการเข้ามาในคิว จะให้ใช้พื้นที่การแบ่งที่เล็กที่สุดที่สามารถใช้ได้และยังว่างอยู่ สำหรับการสลับออกควรสลับกระบวนการที่อยู่ในพื้นที่การแบ่งขนาดเล็กออกไปก่อน นอกจากนั้นยังอาจดูจากคุณสมบัติอื่นๆเช่นค่าระดับความสำคัญของกระบวนการ เป็นต้น ระบบปฏิบัติการที่ใช้การแบ่งพื้นที่แบบตายตัวส่วนใหญ่ได้แก่ระบบปฏิบัติการรุ่นเก่า ได้แก่ระบบปฏิบัติการโอเอส เอ็มเอฟที (OS/MFT) ของไอบีเอ็ม เป็นต้น

6.2.2 การแบ่งพื้นที่แบบพลวัต (dynamic partitioning)

การแบ่งพื้นที่แบบพลวัต เป็นการแก้ปัญหาที่เกิดขึ้นกับการแบ่งพื้นที่แบบตายตัว ซึ่งใช้ในระบบปฏิบัติการโอเอส เอ็มวีที (OS/MVT) ของไอบีเอ็ม การแบ่งพื้นที่แบบพลวัต ขนาดและจำนวนของพื้นที่การแบ่งจะไม่ตายตัว เมื่อกระบวนการต้องการจองพื้นที่ ระบบปฏิบัติการจะจัดสรรขนาดให้เท่ากับที่กระบวนการต้องการ รูปที่ 10

รูปที่ 10 การทำงานของการแบ่งพื้นที่แบบพลวัต

ที่มา : William Stallings, 1992, หน้า 275

จากรูปที่ 10 สมมติว่าหน่วยความจำหลักมีขนาด 1 MB และระบบปฏิบัติการจองพื้นที่ไป 128K ดังนั้นจะมีพื้นที่ว่างเหลือ 896KB รูปที่ 10 ข. ถึง ง. แสดงขั้นตอนของการนำเข้ากระบวนการที่ 1, 2 และ 3 ตามลำดับ แต่ละกระบวนการจะจองพื้นที่ตามที่ตนเองต้องการ โดยเริ่มจองต่อจากพื้นที่ของระบบปฏิบัติการ เมื่อกระบวนการเรียงต่อกันไปเรื่อยๆก็จะพบว่ามีช่องว่างเล็กๆเกิดขึ้นที่ด้านล่างของหน่วยความจำหลัก ซึ่งอาจมีขนาดเล็กเกินกว่าที่จะให้กระบวนการได้ใช้ ดังนั้นระบบปฏิบัติการก็จะเลือกกระบวนการที่จะถูกสลับออกไปที่ฮาร์ดดิสก์เพื่อเพิ่มพื้นที่ว่างให้กับหน่วยความจำหลัก สมมติว่าระบบปฏิบัติการเลือกสลับออกกระบวนการที่ 2 ก็จะเกิดพื้นที่ว่างขึ้น 2 ตำแหน่ง ได้แก่ขนาด 224 K และ 64K ตามลำดับ และเมื่อกระบวนการที่ 4 เข้ามาก็จะใช้พื้นที่ขนาด 224K โดยใช้ไป 128K เหลือพื้นที่ว่าง 96K ซึ่งอาจกลายเป็นช่องว่างเล็กๆที่ใช้งานไม่ได้อีก สมมติว่าจากนั้นระบบปฏิบัติการเลือกสลับออกกระบวนการที่ 1 ก็จะเกิดพื้นที่ว่าง 320K จากนั้นสลับเข้ากระบวนการที่ 2 กลับมาอีกครั้งหนึ่ง โดยใช้พื้นที่ไป 224K เหลือพื้นที่ว่าง 96K จากจุดนี้จะเห็นว่าเกิดพื้นที่ว่างขนาดเล็กๆถึง 3 จุดที่กระบวนการไม่สามารถใช้งานได้ ปัญหาที่เกิดขึ้นนี้จะเรียกว่าการเกิดเศษพื้นที่ภายนอก (external fragmentation)

สำหรับปัญหาเศษพื้นที่ภายนอกสามารถแก้ไขได้โดยการใช้การบีบอัด (compaction) ซึ่งระบบปฏิบัติการจะทำการบีบอัดตามช่วงเวลา โดยจะเลื่อนกระบวนการทั้งหมดให้ขึ้นมาอยู่ติดกันทางด้านบน ทำให้เกิดพื้นที่ว่างเป็นจำนวนมากขึ้นทางด้านล่างของหน่วยความจำ แต่ปัญหาที่ตามมาก็คือการบีบอัดต้องใช้เวลาในการดำเนินการ ทำให้หน่วยประมวลผลกลางไม่สามารถประมวลผลโปรแกรมอื่นได้

6.2.2.1 อัลกอริทึมการจอง การบีบอัดนั้นเป็นการกระทำที่เสียเวลาการประมวลผลของหน่วยประมวลผลกลาง ดังนั้นระบบปฏิบัติการควรมีอัลกอริทึมที่มีประสิทธิภาพเพื่อไม่ให้เกิดปัญหานี้ เช่นเมื่อกระบวนการต้องการจองพื้นที่ในหน่วยความจำแต่มีพื้นที่ว่างกระจายตัวอยู่หลายจุด ระบบปฏิบัติการจะต้องเลือกตำแหน่งที่ต้องการจองให้เหมาะสม ซึ่งมีอัลกอริทึมการจองอยู่ 4 แบบ ได้แก่

1) **แบบพอดีอันแรก (first-fit)** ระบบปฏิบัติการจะตรวจสอบพื้นที่ว่างในหน่วยความจำหลักจากบนลงล่าง หากพบพื้นที่ว่างใดมีขนาดที่กระบวนการสามารถใช้ได้ไม่ว่าจะเหลือพื้นที่ว่างขนาดใดก็ตาม ระบบปฏิบัติการก็จะจองพื้นที่นั้นให้กระบวนการทันที

2) **แบบพอดีที่สุด (best-fit)** ระบบปฏิบัติการจะตรวจสอบพื้นที่ว่างทั้งหมดในหน่วยความจำหลัก และคำนวณว่าพื้นที่ว่างใดมีขนาดพอดีกับกระบวนการสามารถมากที่สุด ซึ่งเมื่อจองแล้วจะเหลือพื้นที่ว่างน้อยที่สุด ระบบปฏิบัติการก็จะจองพื้นที่นั้นให้กระบวนการ

3) **แบบพอดีถัดไป (next-fit)** อัลกอริทึมนี้ระบบปฏิบัติการจะตรวจสอบพื้นที่ว่างอ้างอิงจากการจองครั้งที่ผ่านมา โดยคำนวณว่าถัดจากพื้นที่ว่างที่จองไว้ในครั้งที่ผ่านมา พื้นที่ว่างใดมีขนาดที่กระบวนการสามารถจองได้ ระบบปฏิบัติการก็จะจองพื้นที่นั้นให้กับกระบวนการ

4) **แบบพอดีน้อยที่สุด (worst-fit)** ระบบปฏิบัติการจะตรวจสอบพื้นที่ว่างทั้งหมดในหน่วยความจำหลัก และคำนวณว่าพื้นที่ว่างใดมีขนาดพอดีกับกระบวนการสามารถน้อยที่สุด ซึ่งเมื่อจองแล้วจะเหลือพื้นที่ว่างมากที่สุด ระบบปฏิบัติการก็จะจองพื้นที่นั้นให้กระบวนการ

William Stallings (1992, หน้า 276) ได้กล่าวถึงประสิทธิภาพการทำงานของอัลกอริทึมในข้อที่ 1-3 โดยกล่าวว่าประสิทธิภาพจะขึ้นอยู่กับลำดับของการสลับกระบวนการที่เกิดขึ้น และขนาดของกระบวนการ แต่ Brent, R. (1989) , Shore. J. (1975) และ Bays, C. (1977) ได้กล่าวให้ข้อคิดเห็นเกี่ยวกับอัลกอริทึม 1-3 ว่า การจองพื้นที่แบบพอดีอันแรกเป็นอัลกอริทึมที่ง่ายและดีที่สุด ส่วนอัลกอริทึมแบบพอดีถัดไปมักจะได้การจองพื้นที่ในบริเวณด้านท้ายของหน่วยความจำ ซึ่งมีผลให้พื้นที่ว่างขนาดใหญ่ที่มักจะอยู่ด้านล่างของหน่วยความจำหลัก ถูกแบ่งออกเป็นเศษพื้นที่เล็กๆจำนวนมาก ดังนั้นทำให้เกิดการบีบอัดบ่อยครั้ง ส่วนการจองแบบพอดีมากที่สุด ดูเหมือนว่าจะเหมาะสมที่สุดหากดูจากชื่อ แต่ในความเป็นจริงมีโอกาสที่จะเกิดเศษพื้นที่เล็กๆจำนวนมากเช่นกัน เนื่องจากมันต้องมองหาพื้นที่ที่พอดีที่สุด ซึ่งในความเป็นจริงมีโอกาสเกิดขึ้นได้ยาก จึงต้องจองในพื้นที่ที่มีขนาดใกล้เคียงที่สุด ทำให้เกิดเศษพื้นที่ตามมา ซึ่งจะเกิดการบีบอัดในความถี่ที่สูงที่สุดเช่นกัน ส่วนการจองพื้นที่แบบพอดีถัดไปอาจทำให้เกิดการกระจายของการจองพื้นที่ในหน่วยความจำหลักเป็นจำนวนมาก
(http://www.memorymanagement.org/glossary/n.html,2006,April)

6.2.2.1 อัลกอริทึมการแทนที่ สำหรับการจองพื้นที่แบบพลวัต ซึ่งมีขนาดของพื้นที่การแบ่งไม่แน่นอน เมื่อทำงานไปเรื่อยๆ ก็สามารถใช้พื้นที่ในหน่วยความจำหลักจนเต็มได้ หากพิจารณาถึงสถานะของกระบวนการในหน่วยความจำหลักแล้ว อาจมีบางกระบวนการที่อยู่ในสถานะรอคอย ดังนั้นการที่กระบวนการอยู่ในหน่วยความจำหลักแต่ไม่พร้อมจะประมวลผลนั้น ทำให้เกิดการใช้งานที่ไม่คุ้มค่า โดยปกติ เมื่อหน่วยความจำหลักเต็ม ระบบปฏิบัติการจะเลือกกระบวนการเพื่อทำการสลับออก ซึ่งอัลกอริทึมในการเลือกกระบวนการที่จะถูกสลับออกนี้เรียกว่าอัลกอริทึมการแทนที่ สำหรับเนื้อหาในส่วนของอัลกอริทึมการแทนที่ สามารถศึกษาเพิ่มเติมได้จากเรื่อง หน่วยความจำเสมือน (virtual memory) ในวิชาระบบปฏิบัติการ 2 (รหัสวิชา 4121402)

6.3 การย้ายตำแหน่ง (relocation)

เมื่อกระบวนการถูกนำเข้ามาสู่หน่วยความจำหลัก ระบบปฏิบัติการสามารถย้ายตำแหน่งให้กับแอดเดรสจริงในเชิงตรรกะของพื้นที่แอดเดรสของกระบวนการได้ วิธีนี้ทำให้ระบบปฏิบัติการสามารถเลือกตำแหน่งพื้นที่การจองให้กับกระบวนการในตำแหน่งใดก็ได้ William Stallings (1992, หน้า 278) ได้กล่าวถึงการย้ายตำแหน่งว่า หากเป็นการจองพื้นที่แบบตายตัวที่พื้นที่การแบ่งมีขนาดเท่ากันทั้งหมด และในกรณีของพื้นที่การแบ่งที่มีขนาดไม่เท่ากันแต่มีคิวอันเดียวนั้น กระบวนการสามารถจองพื้นที่ในตำแหน่งใดก็ได้ตลอดช่วงระยะเวลาการประมวลผลของตน เมื่อกระบวนการถูกสร้างขึ้นมาเป็นครั้งแรกจะจองที่ตำแหน่งใดก็ได้ หลังจากนั้นอาจมีการสลับออกไปที่ฮาร์ดดิสก์ และต่อมาถูกสลับเข้ามาในหน่วยความจำหลักอีกครั้งหนึ่ง แต่การจองครั้งนี้กระบวนการอาจไม่ได้จองในพื้นที่การแบ่งเดิมก็ได้ สำหรับการจองพื้นที่แบบพลวัตก็เช่นกัน กระบวนการไม่จำเป็นจะต้องจองพื้นที่ในตำแหน่งเดิม นอกจากนั้นหากระบบมีการบีบอัดก็จะมีการย้ายตำแหน่งเกิดขึ้นเช่นกัน

หากพิจารณาโปรแกรมของกระบวนการ ประกอบไปด้วยคำสั่งและข้อมูล คำสั่งบางคำสั่งอาจมีการอ้างแอดเดรสในหน่วยความจำ ซึ่งการอ้างแอดเดรสนั้นมี 2 ประเภท ได้แก่ การอ้างแอดเดรสไปที่ข้อมูล และการอ้างแอดเดรสไปที่คำสั่งที่ใช้ในการเปลี่ยนเส้นทาง ซึ่งการอ้างแอดเดรสนี้จะใช้การอ้างแบบตายตัวไม่ได้ เนื่องจากกระบวนการสามารถย้ายตำแหน่งได้ ดังที่ได้แสดงให้เห็นในหัวข้อที่ 6.1.2 ซึ่งพบว่าการอ้างแอดเดรสของข้อมูลและคำสั่งของโปรแกรมนั้น จะเป็นการอ้างแอดเดรสแบบเชิงสัมพัทธ์ ซึ่งอ้างจากแอดเดรสจริงเชิงตรรกะที่เริ่มที่หมายเลข 0

เมื่อกระบวนการเข้าสู่สถานะการทำงาน หน่วยประมวลผลกลาง จะใช้รีจิสเตอร์พิเศษที่ชื่อว่าเบสรีจิสเตอร์โดยจะได้รับการกำหนดค่าแอดเดรสเริ่มต้นของกระบวนการ ซึ่งเป็นแอดเดรสจริงในหน่วยความจำ (หรือแอดเดรสจริงในทางกายภาพ) ส่วนรีจิสเตอร์ชื่อลิมิตรีจิสเตอร์จะใช้ในการกำหนดขอบเขตของกระบวนการในหน่วยความจำหลัก เมื่อกระบวนการมีการอ้างแอดเดรสในแบบสัมพัทธ์ค่าแอดเดรสนั้นจะนำมาบวกกับค่าในเบสรีจิสเตอร์ จากนั้นนำไปเปรียบเทียบกับค่าในลิมิตรีจิสเตอร์ หากค่าไม่เกินขอบเขต ค่าผลบวกนั้นก็จะเป็นค่าแอดเดรสจริงในทางกายภาพของกระบวนการที่อ้างถึง แต่หากเมื่อเปรียบเทียบแล้วค่าที่ได้เกินขอบเขต จะเกิดสัญญานอินเทอร์รัพท์ไปที่ระบบปฏิบัติการทันที ดังรูปที่ 11

รูปที่ 11 ฮาร์ดแวร์ที่รองรับการย้ายตำแหน่ง
ที่มา : William Stallings, 1992, หน้า 279

สำหรับการอ้างแอดเดรสในการศึกษาของ Gary Nutt (2002, หน้า 343) จะเรียกรีจิสเตอร์ที่นำมาบวกกับแอดเดรสแบบสัมพัทธ์ว่า รีจิสเตอร์การย้ายตำแหน่ง (relocation register) นอกจากนั้นยังได้กล่าวถึงรายละเอียดเพิ่มเติมถึงการใช้รีจิสเตอร์ย้ายตำแหน่งกับ เซ็กเมนท์ของโปรแกรม ได้แก่เซ็กเมนท์รหัสคำสั่ง, ข้อมูลและแสต็ก ซึ่งจะใช้การย้ายตำแหน่งที่แยกจากกันและใช้รีจิสเตอร์คนละตัว ได้แก่ โค๊ดรีจิสเตอร์ (code register), ดาต้ารีจิสเตอร์ (data register) และแสต็กรีจิสเตอร์ (stack register) ตามลำดับ ซึ่งเมื่อกระบวนการประมวลผล รีจิสเตอร์ทั้ง 3 จะนำมาบวกกับแอดเดรสแบบสัมพัทธ์เพื่อได้แอดเดรสจริงทางกายภาพ ดังรูปที่ 12

รูปที่ 12 รีจิสเตอร์ที่รองรับการย้ายตำแหน่งแบบหลายเซ็กเมนท์
ที่มา : Gary Nutt, 2002, หน้า 344

ตัวแปลภาษาในยุคสมัยปัจจุบันนี้ มีการใช้ประโยชน์จากการย้ายตำแหน่งโดยฮาร์ดแวร์อย่างเต็มที่ โดยจะออกแบบโปรแกรมให้ใช้เซ็กเมนท์ต่างๆสำหรับเก็บรหัสคำสั่ง, ข้อมูลและแสต็ก โดยที่เซ็กเมนท์ต่างๆเหล่านี้สามารถย้ายตำแหน่งได้ (Gary Nutt 2002, หน้า 344) โค๊ดรีจิสเตอร์ จะเก็บค่าแอดเดรสของเซ็กเมนท์ของรหัสคำสั่ง ดาต้ารีจิสเตอร์ จะเก็บค่าแอดเดรสของเซ็กเมนท์ของข้อมูล และแสต็กรีจิสเตอร์จะเก็บค่าแอดเดรสของเซ็กเมนท์ของแสต็กของกระบวนการนั้นๆ

ระบบปฏิบัติการจะเป็นผู้กำหนดค่าให้กับโค๊ดรีจิสเตอร์ เพื่อกำหนดตำแหน่งการวางในหน่วยความจำหลัก โดยที่ค่าแอดเดรสของเซ็กเมนท์ของรหัสคำสั่งนี้ จะถูกบวกเข้ากับแอดเดรสแบบสัมพัทธ์ที่ได้จากโปรแกรม ซึ่งผ่านการแปลจากตัวแปลภาษา สำหรับคำสั่งที่มีขนาด 16 บิท แต่ละเซ็กเมนท์จะมีขนาดเท่ากับ 64 KB โดยการอ้างแอดเดรสแบบสัมพัทธ์จะอ้างภายในเซ็กเมนท์นั้นๆ ยกเว้นในกรณีที่โปรแกรมมีการเปลี่ยนเส้นทางการทำงานไปที่เซ็กเมนท์อื่น จะต้องมีการอ้างแอดเดรสแบบภายนอก (external reference) ซึ่งจะต้องมีการเปลี่ยนแปลงค่าในโค๊ดรีจิสเตอร์ ตัวอย่างเช่นการเรียกฟังก์ชั่นของโปรแกรม ซึ่งจะมีการเปลี่ยนเส้นทางไปที่เซ็กเมนท์ที่เก็บรหัสคำสั่งของฟังก์ชั่นนั้นๆ ตัวแปลภาษาจะสร้างคำสั่งเพื่อให้สามารถเปลี่ยนแปลงค่าในโค๊ดรีจิสเตอร์ได้ โดยการกำหนดค่าให้กับโค๊ดรีจิสเตอร์ก่อนที่จะเรียกคำสั่ง call ไปที่ฟังก์ชั่นดังรูปที่ 13

รูปที่ 13 การเปลี่ยนแปลงค่าที่โค๊ดรีจิสเตอร์

ที่มา : Gary Nutt, 2002, หน้า 345

โดยปกติแล้ว ตัวแปลภาษาจะไม่ได้กำหนดแอดเดรสจริงในเชิงตรรกะ ให้กับเซ็กเมนท์ของฟังก์ชั่น แต่จะใช้การอ้างผ่านสัญลักษณ์แทน ซึ่งผู้ที่จะกำหนดแอดเดรสนี้ให้ก็คือตัวเชื่อมโยง โดยจะกำหนดในขั้นตอนของการเชื่อมโยง และสำหรับการอ้างถึงข้อมูลที่อยู่คนละเซ็กเมนท์กัน เช่นเมื่อโปรแกรมมีการอ้างถึงตัวแปรประเภทตายตัว ก็ใช้หลักการของการเปลี่ยนแปลงค่าที่ดาต้ารีจิสเตอร์ก่อนการเข้าถึงข้อมูลเช่นกัน

สำหรับการทำงานของการเปลี่ยนแปลงค่าที่โค๊ดรีจิสเตอร์หรือดาต้ารีจิสเตอร์นี้ ส่วนใหญ่จะเป็นการทำงานในรูปแบบการทำงานแบบควบคุม โดยก่อนที่โปรแกรมจะเปลี่ยนเซ็กเมนท์จะต้องเรียกคำสั่ง trap ก่อน หลังจากนั้นระบบปฏิบัติการจะเรียกการทำงานของส่วนจัดการหน่วยความจำหลักเพื่อตรวจสอบการเปลี่ยนเซ็กเมนท์นั้น และจะเปลี่ยนแปลงค่าที่รีจิสเตอร์ PC เพื่อให้หน่วยประมวลผลกลางนำเข้าข้อมูลหรือคำสั่งตามที่โปรแกรมต้องการ (Gary Nutt 2002, หน้า 346)

6.4 กลยุทธ์การจัดการหน่วยความจำหลัก

การทำงานของกระบวนการในหน่วยความจำหลัก มีความซับซ้อนในการทำงาน โดยต้องการการประสานการทำงานกันระหว่างกระบวนการ ระบบปฏิบัติการ และฮาร์ดแวร์ที่รองรับ เมื่อกระบวนการมีการประมวลผล จะมีการใช้งานหน่วยความจำหลัก สำหรับระบบแบบหลายโปรแกรม จะต้องมีการแบ่งปันหน่วยความจำหลักระหว่างกระบวนการต่างๆ ดังนั้นระบบปฏิบัติการจะต้องมีวิธีการในการจัดการที่ดี เพื่อให้สามารถใช้หน่วยความจำหลักได้อย่างคุ้มค่าที่สุด

จากการศึกษาของ Gary Nutt (2002, หน้า 348) พบว่าระบบปฏิบัติการในปัจจุบันจะใช้หลักของหน่วยความจำเสมือน สำหรับการจัดการหน่วยความจำหลัก ซึ่งประกอบไปด้วยการสลับ (swapping) ในการทำงานของฮาร์ดแวร์ระดับล่าง ซึ่งการสลับเป็นการใช้งานการย้ายตำแหน่งได้อย่างมีประสิทธิภาพ

6.4.1 การสลับ

Silberchatz Galvin (1999, หน้า 246) ได้กล่าวถึงเรื่องการสลับว่า โดยปกติกระบวนการจะต้องอยู่ในหน่วยความจำหลักเพื่อการประมวลผล แต่กระบวนการสามารถสลับออก ไปอยู่ที่หน่วยความจำสำรองได้ชั่วคราว และเมื่อต้องการประมวลผลในภายหลัง ก็สามารถสลับเข้า กลับคืนสู่หน่วยความจำหลักได้เช่นกัน ตัวอย่างการทำงานของการสลับเช่น การจัดตารางการทำงานแบบเวียนรอบ เมื่อหมดเวลาของชิ้นเวลาที่ตั้งไว้ ระบบปฏิบัติการก็อาจจะสลับกระบวนการที่หมดเวลานั้นออกจากหน่วย

ความจำหลัก และในทางตรงกันข้าม ก็อาจจะสลับกระบวนการที่พร้อมจะประมวลผลเข้ามาสู่หน่วยความจำหลัก

นอกจากนั้นแล้ว ในการจัดตางรางการทำงานแบบกำหนดระดับความสำคัญ เมื่อกระบวนการที่เข้ามาประมวลผลมีค่าระดับความสำคัญสูง ระบบปฏิบัติการก็จะสลับกระบวนการที่มีค่าระดับความสำคัญต่ำออกจากหน่วยความจำหลัก และเมื่อกระบวนการที่มีค่าระดับความสำคัญสูงนั้นจบการทำงานแล้ว ก็สามารถสลับกระบวนการที่มีค่าระดับความสำคัญต่ำที่สลับออกไป กลับเข้ามาสู่หน่วยความจำหลักเช่นเดิมได้

Gary Nutt (2002, หน้า 349) ได้กล่าวถึงการสลับว่าเหมาะสมกับระบบงานแบบแบ่งปันเวลา เนื่องจากอาจมีบางเวลาที่ผู้ใช้บางคนไม่ได้เรียกการบริการในอัตราสูง โดยส่วนจัดการหน่วยความจำหลัก สามารถสลับกระบวนการที่ยังไม่จำเป็นต้องประมวลผลในขณะนั้นออกก่อนเพื่อเพิ่มพื้นที่ว่างให้กับหน่วยความจำหลัก นอกจากนั้นยังกล่าวถึงกุญแจสำคัญของการสลับว่า หากกระบวนการไม่ได้ใช้งานหน่วยประมวลผลกลางเป็นเวลานานช่วงหนึ่งแล้ว กระบวนการนั้นก็ควรสลับออกจากหน่วยความจำหลัก ส่วนนโยบายของการสลับ มีความสำคัญกับระบบการทำงานแบบหลายโปรแกรม และแบบมีปฏิสัมพันธ์กับผู้ใช้เช่นกัน แม้ว่าระบบจะมีหน่วยความจำหลักมากเพียงไรก็ตาม เนื่องจากการทำงานของระบบมีการทำงานแบบต่อเนื่อง และไม่สามารถทำนายพฤติกรรมของผู้ใช้ได้ ซึ่งจะใช้หลักการว่าจะไม่สลับกระบวนการออกจากหน่วยความจำหลักหากไม่จำเป็น เนื่องจากจำนวนครั้งของการสลับ มีผลต่อประสิทธิภาพการทำงานของระบบโดยตรง ซึ่งอาจทำให้เวลาตอบสนองเพิ่มขึ้น

6.4.2 หน่วยความจำเสมือน (virtual memory)

กลยุทธ์ของหน่วยความจำเสมือน อนุญาตให้กระบวนการใช้หน่วยประมวลผลกลาง เมื่อมีส่วนของพื้นที่แอดเดรสของกระบวนการนั้นอยู่ในหน่วยความจำหลักเท่านั้น (Gary Nutt, 2002, หน้า 350) สิ่งนี้ทำให้สามารถแบ่งกระบวนการออกเป็นเซ็กเมนท์ต่างๆ และนำเข้าหน่วยความจำหลักเมื่อต้องการใช้งาน และนำออกไปไว้ที่หน่วยความจำสำรอง เมื่อไม่ได้ใช้งานแล้ว ดังรูปที่ 14

รูปที่ 14 แนวคิดของหน่วยความจำเสมือน

ที่มา : Gary Nutt, 2002, หน้า 351

Silberchatz Galvin (1999, หน้า 289) ได้อธิบายถึงความหมายของหน่วยความจำเสมือนว่าเป็นเทคนิคที่อนุญาตให้กระบวนการสามารถประมวลผลได้โดยไม่จำเป็นต้องอยู่ในหน่วยความจำหลักทั้งหมด ซึ่งมีข้อดีคือสามารถใช้โปรแกรมที่มีขนาดใหญ่กว่าหน่วยความจำหลักได้ โดยจะสร้างภาพจำลองว่าหน่วยความจำหลักมีความจุเป็นจำนวนมาก นอกจากนั้น Silberchatz ยังได้กล่าวเพิ่มเติมว่า หน่วยความจำเสมือนสามารถทำให้ประสิทธิภาพของระบบลดลงได้ ส่วน William Stallings (1992, หน้า 286) ได้กล่าวว่าหน่วยความจำเสมือนใช้หลักการของการกระทำกับหน้า (paging) และการกระทำกับเซ็กเมนท์ (segmentation) โดยมีหลักการทำงาน 2 ข้อได้แก่

1) การอ้างถึงหน่วยความจำภายในกระบวนการเป็นการอ้างถึงโดยการใช้แอดเดรสเชิงตรรกะ ซึ่งสามารถเปลี่ยนแปลงเป็นแอดเดรสในทางกายภาพได้อย่างมีพลวัต เมื่อกระบวนการอยู่ในเวลาทำงาน ซึ่งทำให้มันสามารถสลับเข้า-ออกจากหน่วยความจำหลักได้ และสามารถเปลี่ยนแปลงแอดเดรสการวางในหน่วยความจำหลักของตนได้
2) กระบวนการสามารถแบ่งออกเป็นชิ้นได้ (หน้าหรือเซ็กเมนท์) โดยที่แต่ละชิ้นนี้ต้องการพื้นที่ที่ต่อเนื่องกันในหน่วยความจำหลักเมื่อมันจะประมวลผล การกระทำสิ่งนี้ได้ต้องการการทำงานร่วมกันระหว่างการแปลงแอดเดรสแบบพลวัตขณะกระบวนการทำงาน และการใช้หน้าและเซ็กเมนท์

สำหรับเรื่องของหน่วยความจำเสมือน สามารถศึกษาเพิ่มเติมได้จากวิชาระบบปฏิบัติการ 2 (รหัสวิชา 4121402)

6.5 สรุป

การจัดการหน่วยความจำหลัก มีความสำคัญต่อการทำงานของระบบปฏิบัติการเป็นอย่างมาก เนื่องจากโปรแกรมต่างๆที่ต้องการประมวลผลต้องนำเข้าไปไว้ที่หน่วยความจำหลัก ก่อนที่จะประมวลผลที่หน่วยประมวลผลกลาง ซึ่งการจัดการหน่วยความจำหลัก มีผลต่อประสิทธิภาพของระบบปฏิบัติการ เนื่องจากหน่วยความจำหลัก เป็นทรัพยากรที่มีจำนวนจำกัด ซึ่งมีเทคนิคการจัดการเป็นจำนวนมากที่สามารถนำมาใช้ได้ เช่น การสลับ, หน่วยความจำเสมือน, การย้ายตำแหน่ง เป็นต้น และการนำกระบวนการเข้ามาไว้ที่หน่วยความจำหลักนั้น สามารถแบ่งหน่วยความจำหลักเป็นแบบต่างๆได้ เช่น

การแบ่งพื้นที่ย่อยขนาดเท่าๆกัน หรือการแบ่งเป็นพื้นที่ย่อยในขนาดที่สามารถปรับเปลี่ยนได้ นอกจากนั้นในการนำกระบวนการเข้ามาวางในหน่วยความจำหลัก สามารถใช้อัลกอริทึมการจองในแบบต่างๆได้ เช่น แบบพอดีอันแรก, พอดีที่สุด, แบบพอดีถัดไป, พอดีน้อยที่สุดเป็นต้น การจัดการหน่วยความจำต้องการความร่วมมือจากฮาร์ดแวร์ในระบบคอมพิวเตอร์ เช่นการย้ายตำแหน่ง ซึ่งการออกแบบระบบปฏิบัติการจะต้องคำนึงถึงด้วย เพื่อประสิทธิภาพและความมีเสถียรภาพในการทำงานของระบบ