บทที่ 6
บทบาทหน้าที่ระบบปฏิบัติการในการจัดการหน่วยความจำหลัก

ระบบคอมพิวเตอร์ในปัจจุบันได้ถูกพัฒนาให้สามารถประมวลผลโปรแกรมได้หลายโปรแกรมในเวลาเดียวกัน ซึ่งทำให้แต่ละโปรแกรมต้องแบ่งพื้นที่หน่วยความจำเพื่อใช้งานร่วมกัน จึงส่งผลให้เกิดภาวะหน่วยความจำไม่เพียงพอสำหรับรองรับโปรแกรมจำนวนมาก ดังนั้นเพื่อเพิ่มขีดความสามารถให้ระบบคอมพิวเตอร์สามารถดำเนินการได้อย่างมีประสิทธิภาพสูงสุด จึงต้องมีการจัดการกับหน่วยความจำหลักซึ่งเป็นหน้าที่สำคัญประการหนึ่งของระบบปฏิบัติการให้สามารถรองรับการทำงานของโปรแกรมในปัจจุบันที่มีความสลับซับซ้อน มีความสามารถสูง ทำให้ต้องใช้หน่วยความจำในการประมวลผลเป็นจำนวนมาก ในบทนี้จะอธิบายเกี่ยวกับ 1) ความหมายของหน่วยความจำหลัก 2) โครงสร้างหน่วยความจำหลัก 3) การวิเคราะห์หน่วยความจำหลัก 4) รูปแบบการจองพื้นที่หน่วยความจำหลัก และ 5) การคืนพื้นที่หน่วยความจำหลักเพื่อให้ระบบปฏิบัติการสามารถจัดการกับหน่วยความจำได้อย่างมีประสิทธิภาพ

6.1 ความหมายของหน่วยความจำหลัก

หน่วยความจำหลักมีนักวิชาการได้ให้ความหมายของหน่วยความจำหลักไว้ดังต่อไปนี้

ไพศาล โมลิสกุลมงคลและคณะ(2545: 145-146) กล่าวว่า หน่วยความจำหลักหมายถึง พื้นที่เก็บข้อมูลขนาดใหญ่ที่ประกอบไปด้วย พื้นที่เก็บข้อมูลย่อยที่มีขนาดเป็นไบต์ (Byte) โดยแต่ละไบต์จะมีแอดเดรส (Address) บอกตำแหน่งของตัวเองและโดยทั่วไปแล้วโปรแกรมเมอร์ล้วนแต่ต้องการระบบที่มีหน่วยความจำหลักแบบไม่จำกัด มีความเร็วสูงและเป็นหน่วยความจำที่มีความเสถียรภาพสูงหรือเป็นแบบไม่ถูกลบเลือน (Nonvolatile) แม้ว่าจะไม่มีไฟฟ้าเลี้ยงก็ตาม หน่วยความจำหลักแบ่งออกเป็นชั้นๆ (Memory Hierarchy) ได้ดังนี้

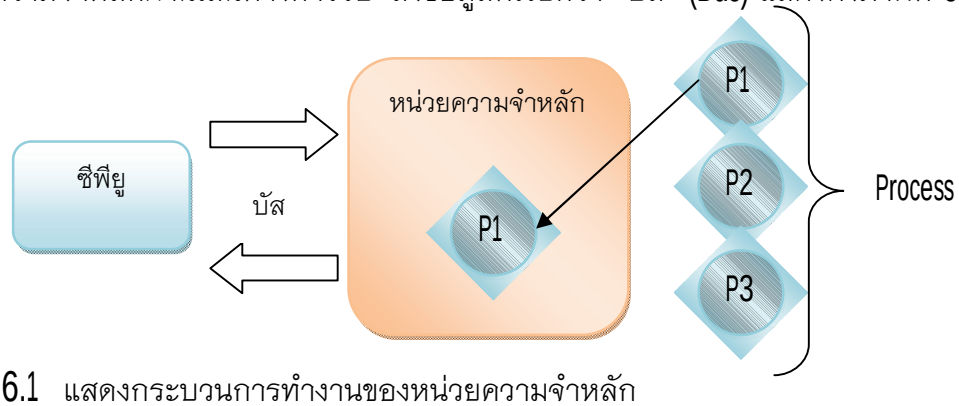
1. หน่วยความจำขนาดเล็กมีความเร็วสูง ราคาแพงมากและเป็นหน่วยความจำที่เป็นแบบลบเลือนได้ (Volatile Cache Memory)

2. หน่วยความจำขนาดกลาง ประมาณ 100 กว่าเมกะไบต์ขึ้นไป มีความเร็วปานกลาง ราคาปานกลางและเป็นหน่วยความจำที่เป็นแบบลบเลือนได้ คือ แรม (RAM)

3. หน่วยความจำขนาดใหญ่ประมาณ 10-100 กิกะไบต์ ที่มีความเร็วต่ำ ราคาถูกและเป็นดิสก์เก็บข้อมูลที่เป็นแบบไม่ลบเลือน

ส่วนของระบบปฏิบัติการที่ใช้ในการจัดการหน่วยความจำของลำดับชั้นต่างๆ จะถูกเรียกว่า ตัวจัดการหน่วยความจำ (Memory Manager) ซึ่งมีหน้าที่ในการตรวจสอบว่าส่วนใดของหน่วยความจำที่กำลังถูกใช้งาน ส่วนใดที่ว่างอยู่ทำหน้าที่จัดสรรหน่วยความจำให้กับโปรเซสที่ทำงาน เก็บหน่วยความจำเหล่านั้นกลับคืนสู่ระบบเมื่องานเสร็จและจัดการสลับหน่วยความจำหลักกับพื้นที่ฮาร์ดดิสก์เมื่อหน่วยความจำหลักมีขนาดเล็กเกินไปที่จะให้โปรเซสทำงานได้

พิรพร หมุนสนิทและคณะ (2553: 68) กล่าวว่า หน่วยความจำหลัก (Main Memory) คือ พื้นที่สำหรับจัดเก็บข้อมูลแบบชั่วคราว ซึ่งแบ่งออกเป็นหน่วยเก็บข้อมูลย่อยที่มีหน่วยเป็นไบต์ (Byte) หรือเวิร์ด (Word) แต่ละไบต์จะมีหมายเลขสำหรับบอกตำแหน่ง (Address) ของตัวเอง โดยซีพียู (CPU) จะเป็นผู้อ่านหรือเขียนข้อมูลลงในตำแหน่งต่างๆ ของหน่วยความจำหลักผ่านเส้นทางการรับ-ส่งข้อมูลที่เรียกว่า “บัส” (Bus) แสดงดังภาพที่ 6.1



ภาพที่ 6.1 แสดงกระบวนการทำงานของหน่วยความจำหลัก
ที่มา : พิรพร หมุนสนิทและคณะ (2553: 68)

จากภาพที่ 6.1 จะเห็นว่า ซีพียูจะโหลดโปรเซสที่ต้องการจากหน่วยความจำหลักหรือรีจิสเตอร์ (Register) ซึ่งทำหน้าที่จัดเก็บข้อมูลมาถอดรหัสและนำข้อมูลนั้นไปประมวลโดยผลลัพธ์ที่ได้จะถูกโหลดเข้าสู่ตำแหน่งในหน่วยความจำหลัก ซึ่งถูกสร้างขึ้นขณะกำลังประมวลผลเช่นเดิม โดยทั่วไปซีพียูจะสามารถเข้าถึง Register ได้โดยตรงในเวลาหนึ่งรอบการทำงานของซีพียู ส่วนการเข้าถึงหน่วยความจำหลักจะใช้เวลามากกว่ารีจิสเตอร์หลายรอบ

สรุปได้ว่า หน่วยความจำหลัก (Main Memory) หมายถึง พื้นที่สำหรับจัดเก็บข้อมูลแบบชั่วคราวในขณะการประมวลผล มีหน่วยในการเก็บข้อมูลเป็นไบต์ (Byte) โดยแต่ละไบต์จะมีหมายเลขแอดเดรส (Address) ของหน่วยความจำ ในการทำงานจะมีซีพียูเป็นตัวควบคุม

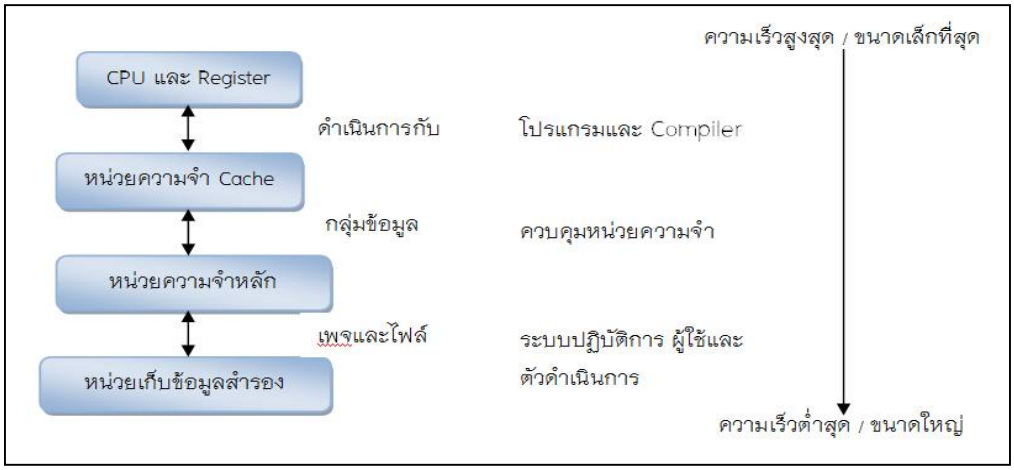
การทำงานของหน่วยความจำ คือ อ่านและเขียนข้อมูลลงหน่วยความจำหลักผ่านเส้นทางการรับ-ส่งข้อมูลที่เรียกว่า “บัส”(Bus) ซึ่งหน่วยความจำหลักมี 2 ชนิด แบบลบเลือนได้เมื่อไม่มีกระแสไฟฟ้ามาล่อเลี้ยง เรียกว่า แรม (RAM) และแบบลบเลือนไม่ได้แม้ไม่มีกระแสไฟฟ้ามาหล่อเลี้ยงเรียกว่ารอม (ROM)

6.2 โครงสร้างของหน่วยความจำ

หน่วยความจำเป็นองค์ประกอบสำคัญที่ช่วยสนับสนุนให้การดำเนินการในระบบคอมพิวเตอร์มีประสิทธิภาพมากขึ้น โดยหน่วยความจำจะทำหน้าที่เก็บโปรเซสคำสั่งและข้อมูลต่างๆ ของโปรแกรม เพื่อให้ซีพียูนำไปใช้ในการประมวล ในส่วนโครงสร้างของหน่วยความจำผู้สอนได้ศึกษาจากนักวิชาการดังนี้ 1) วิเชษฐ์ พลายมาศ (2552: 172-199) 2) พีระพนธ์ โสพัศสถิตย์ (2552: 50-60) 3) พิรพร หมุนสนิทและคณะ (2553: 69-71) แล้วนำมากล่าวโดยสรุปได้ดังต่อไปนี้

6.2.1 ลำดับชั้นของหน่วยความจำ

โดยทั่วไปซีพียูและรีจิสเตอร์จะใช้แอดเดรสในการเข้าถึงข้อมูลในหน่วยความจำหลัก โดยดำเนินการกับคำสั่งต่างๆ ผ่านหน่วยความจำแคช (Cache)เพื่อเข้าถึงข้อมูลในหน่วยความจำหลัก ถ้าหน่วยความจำแคชได้รับการออกแบบเป็นอย่างดีจะช่วยให้ซีพียูและรีจิสเตอร์ สามารถเข้าถึงข้อมูลที่อยู่ในหน่วยความจำหลักได้ง่ายขึ้น ดังภาพที่ 6.2



ภาพที่ 6.2 แสดงลำดับชั้นของหน่วยความจำ

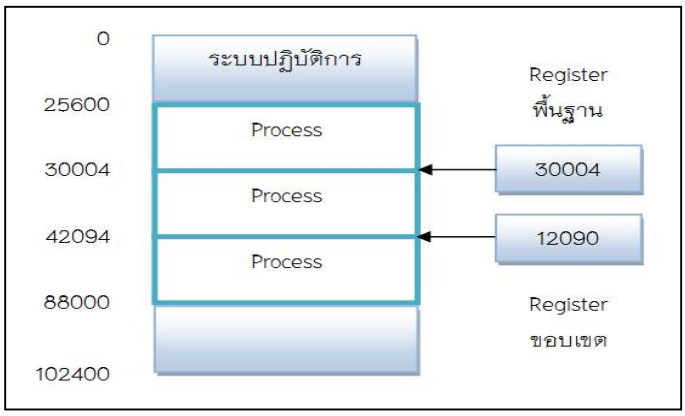
ที่มา : พิรพร หมุนสนิทและคณะ (2553: 69)

จากภาพที่ 6.2 จะเห็นว่าหน่วยความจำในระดับบนจะมีขนาดเล็ก สามารถอ่านหรือจัดเก็บข้อมูลด้วยความเร็วสูงและใช้เวลาในการเข้าถึงข้อมูลน้อยแต่มีราคาแพง ส่วนหน่วยความจำในระดับล่างจะมีขนาดใหญ่ การอ่านหรือจัดเก็บข้อมูลมีความเร็วต่ำและใช้เวลาในการเข้าถึงข้อมูลนานแต่มีราคาถูกกว่าหน่วยความจำระดับบน ในทางปฏิบัติระบบคอมพิวเตอร์ไม่สามารถใช้หน่วยความจำระดับบนได้เพียงอย่างเดียว เนื่องจากมีราคาแพง ดังนั้นจึงแก้ปัญหาโดยใช้หน่วยความจำระดับล่างมากขึ้นเพื่อให้เกิดความสมดุลระหว่างประสิทธิภาพและราคา

6.2.2 ความสามารถด้านฮาร์ดแวร์

ในระบบคอมพิวเตอร์จะผนวกหน่วยความจำหลักและรีจิสเตอร์เข้ากับซีพียูเพื่อจัดเก็บคำสั่งหรือข้อมูลต่างๆ ที่ใช้สำหรับประมวลผลโดยระบบปฏิบัติการจะนำข้อมูลจากหน่วยความจำหลักมาประมวลผลและนำผลลัพธ์ที่ได้จัดเก็บไว้ในรีจิสเตอร์ หน่วยความจำหลักหรืออุปกรณ์รับ-ส่งข้อมูลต่อไป การป้องกันหน่วยความจำหลักจะใช้วิธีแบ่งหน่วยความจำให้แต่ละโปรเซสแยกกันอย่างอิสระเพื่อป้องกันไม่ให้โปรเซสหนึ่งเข้าไปอ่านหรือแก้ไขข้อมูลของโปรเซสอื่นได้ โดยใช้รีจิสเตอร์ทำหน้าที่ป้องกันหน่วยความจำหลัก ซึ่งแบ่งออกเป็น 2 ส่วน ได้แก่ รีจิสเตอร์พื้นฐาน (Base Register) ใช้จัดเก็บโปรเซส ที่มีแอดเดรสต่ำสุดและรีจิสเตอร์ขอบเขต (Limit Register)ใช้จัดเก็บขนาดพื้นที่ห้องทั้งหมดของโปรเซส โดยระบบปฏิบัติการจะใช้รีจิสเตอร์ทั้งสองนี้กำหนดพื้นที่ทางตรรกะ (Logical Address Space) ให้แต่ละโปรเซสทำงานในพื้นที่ที่แยกจากกัน

ตัวอย่าง เช่น ระบบปฏิบัติการมีรีจิสเตอร์พื้นฐานที่มี Address เป็น 30004 และรีจิสเตอร์ขอบเขตมี Address เป็น 12090 แสดงได้ดังภาพที่ 6.3



ภาพที่ 6.3 แสดงการทำงานของรีจิสเตอร์พื้นฐานและรีจิสเตอร์ ขอบเขต

ที่มา : พิรพร หมุนสนิทและคณะ (2553: 70)

6.2.3 การเชื่อมโยง Address

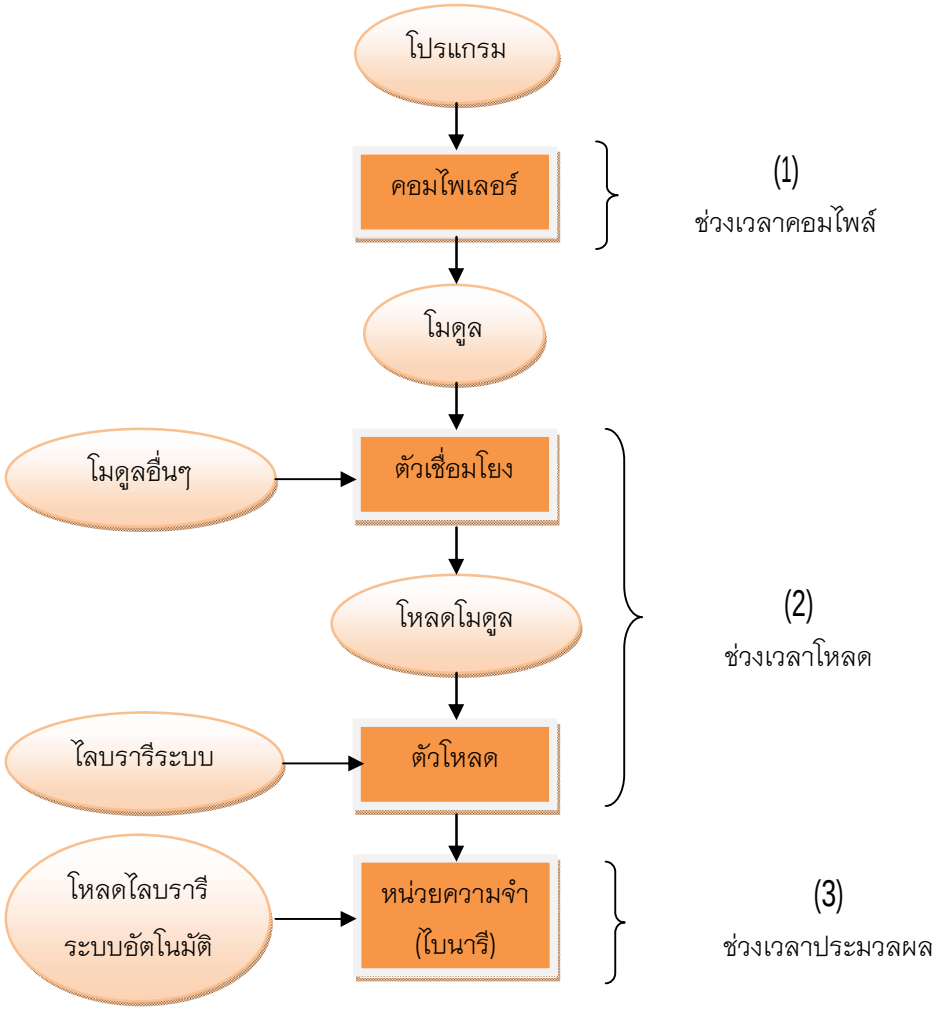
โดยทั่วไปโปรเซสที่ถูกนำไปประมวลผลมีการจองพื้นที่ในหน่วยความจำหลักให้แต่ละโปรเซส ซึ่งอาจถูกเคลื่อนย้ายไปมาระหว่างดิสก์และหน่วยความจำหลักจนกว่าจะถูกประมวลผล โดยระบบปฏิบัติการโหลดโปรเซสจากคิวเข้าสู่หน่วยความจำหลักและโปรเซสนั้นสามารถเข้าถึงคำสั่งและข้อมูลต่างๆ ในหน่วยความจำหลักได้จนกว่าจะดำเนินการเสร็จสิ้น รวมถึงเชื่อมโยงกับค่าของแอดเดรสเพื่อโหลดโปรเซสใหม่เข้าสู่หน่วยความจำหลัก การทำงานลักษณะนี้เรียกว่า “การเชื่อมโยงแอดเดรส (Address Binding)” โดยจำแนกค่าของแอดเดรสออกเป็น 2 ค่า คือ Absolute Address เป็นแอดเดรสจริงของโปรเซสที่อยู่ในหน่วยความจำหลัก และ Relative Address เป็นแอดเดรสของโปรแกรมหรือชุดคำสั่งที่ได้รับจากการคอมไพล์ ซึ่งการเชื่อมโยงแอดเดรสคำสั่งข้อมูลต่างๆ ของโปรแกรมเข้ากับแอดเดรสในหน่วยความจำหลักจะแบ่งระยะเวลาการทำงานออกเป็น 3 ช่วงเวลา ดังนี้

6.2.3.1 ช่วงเวลาคอมไพล์ (Compile Time) คือช่วงเวลาในการคอมไพล์คำสั่งหรือข้อมูลต่างๆ โดยมีคอมไพเลอร์(Compiler) หรือตัวคอมไพล์ทำหน้าที่แปลคำสั่งหรือข้อมูลเหล่านั้นให้เป็นส่วนย่อยที่เรียกว่า “โมดูล (Module)”ถ้าคอมไพเลอร์ทราบแอดเดรสของหน่วยความจำหลักก็จะสามารถสร้างโค้ดที่ประกอบด้วยแอดเดรสจริงของข้อมูลแต่ละตัว (Absolute Code) ทำให้ระบบสามารถประมวลผลคำสั่งหรือข้อมูลนั้นได้ทันที แต่หากแอดเดรสมีการเปลี่ยนแปลงคอมไพเลอร์จะต้องคอมไพล์คำสั่งหรือข้อมูลต่างๆ ใหม่ทุกครั้ง

6.2.3.2 ช่วงเวลาโหลด (Load Time) คือช่วงเวลาในการโหลดคำสั่งหรือข้อมูลต่างๆ เข้าสู่หน่วยความจำหลัก โดยคอมไพเลอร์จะสร้างโค้ดที่สามารถประมวลผลได้ทันที (Reloadable Code) ขึ้นมาก่อน เมื่อระบบโหลดคำสั่งหรือข้อมูลต่างๆ เข้าสู่หน่วยความจำหลักเป็นที่เรียบร้อยแล้ว คอมไพเลอร์จึงจะแปลแอดเดรสของคำสั่งหรือข้อมูลเหล่านั้นให้เป็นแอดเดรสจริง จากนั้นระบบจึงสามารถประมวลผลคำสั่งหรือข้อมูลนั้นได้ ช่วงเวลานี้มีข้อดี คือ ระบบสามารถอ่านคำสั่งหรือข้อมูลเข้าสู่หน่วยความจำหลักได้โดยไม่จำเป็นต้องคอมไพล์คำสั่งหรือข้อมูลนั้นใหม่ ส่วนข้อเสีย คือ ระบบจะใช้เวลาในการโหลดคำสั่งหรือข้อมูลต่างๆ นานขึ้น

6.2.3.3 ช่วงเวลาประมวลผล(Execution Time) คือช่วงเวลาในการประมวลผลคำสั่งหรือข้อมูลต่างๆ โดยคอมไพเลอร์จะเชื่อมโยงแอดเดรสและแปลโค้ดคำสั่งของแอดเดรสเพื่อให้ระบบสามารถโหลดคำสั่งหรือรับข้อมูลต่างๆ เข้าไว้ในหน่วยความจำหลัก ทำให้เกิดความคล่องตัวในการโหลดข้อมูลเหล่านี้ไปยังตำแหน่งต่างๆ ในหน่วยความจำหลัก

ขณะกำลังประมวลผลได้ แต่อาจทำให้เสียเวลาในการประมวลผลมากขึ้น เนื่องจากตัวแปลแอดเดรสของคำสั่งหรือข้อมูลต่างๆ ก่อนประมวลผลทุกครั้ง



ภาพที่ 6.4 แสดงช่วงเวลาการเชื่อมโยงแอดเดรส
ที่มา : พิรพร หมุนสนิทและคณะ (2553: 71)

จากภาพที่ 6.4 1) โปรแกรมจะใช้คอมไพเลอร์คอมไพล์โมดูลต่างๆ เพื่อส่งเข้าสู่ช่วงเวลาที่ 2) โดยตัวเชื่อมโยงจะเชื่อมโยงโมดูลที่ได้รับเข้ากับโมดูลอื่นๆ เพื่อทำการโหลดโมดูลทั้งหมดเข้าสู่หน่วยความจำหลักและในช่วงเวลาที่ 3) โมดูลต่างๆ จะถูกจัดเก็บลงหน่วยความจำหลักขณะประมวลผล

6.3 วิเคราะห์หน่วยความจำหลัก

ในเวลาหนึ่งๆ หน่วยความจำหลักอาจมีโปรเซสเพียงโปรเซสเดียวอาศัยอยู่หรือมีหลายๆ โปรเซสอาศัยอยู่ ซึ่งถ้ามีหลายๆ โปรเซสอาศัยอยู่ในหน่วยความจำหลักพร้อมๆ กัน ซึ่งผู้เขียนได้ศึกษาสิ่งที่ต้องวิเคราะห์เกี่ยวกับหน่วยความจำหลักจากนักวิชากรดังนี้ 1) ชูเกียรติ วรสุชีพ (2551: 172-187) 2) อรพิน ประวัติบริสุทธิ์ (2551: 111-112) แล้วนำมากล่าวโดยสรุปดังนี้

6.3.1 วิเคราะห์ว่าจะให้ทุกโปรเซสใช้พื้นที่หน่วยความจำหลักที่เดียวกันทั้งหมดหรือจะแบ่งพื้นที่หน่วยความจำหลักออกเป็นส่วนๆ เรียกว่า พาร์ทิชัน (Partition) โดยแต่ละพาร์ทิชันมีขนาดที่แตกต่างกันออกไป แล้วจัดสรรพื้นที่หน่วยความจำหลักในแต่ละพาร์ทิชันนี้ให้กับโปรเซสแต่ละตัว

6.3.2 วิเคราะห์ว่าขนาดของพาร์ทิชันแต่ละส่วนนั้นถูกกำหนดไว้ตายตัวไม่สามารถเปลี่ยนแปลงได้หรือไม่หรือสามารถปรับเปลี่ยนขนาดของพาร์ทิชันได้ตามความต้องการของโปรเซส

6.3.3 วิเคราะห์ว่าการประมวลผลโปรเซสนั้น จำเป็นหรือไม่ที่จะต้องให้โปรเซสประมวลผลอยู่ในส่วนของพาร์ทิชันที่กำหนดหรือสามารถให้โปรเซสประมวลผลได้ในทุกพาร์ทิชันของหน่วยความจำที่มีขนาดเพียงพอที่โปรเซสจะไปอาศัยอยู่ได้

6.3.4 วิเคราะห์ว่าจำเป็นหรือไม่ที่จะต้องให้แต่ละโปรเซสอาศัยอยู่ในส่วนของหน่วยความจำหลักที่ติดกันหรือสามารถแยกโปรเซสออกเป็นส่วนๆ และวางโปรเซสแต่ละส่วนลงในแต่ละพาร์ทิชันของหน่วยความจำหลักที่มีขนาดเพียงพอให้ส่วนของโปรเซสไปอาศัยอยู่ได้โดยที่แต่ละพาร์ทิชันไม่ได้อยู่ติดกัน

6.4 รูปแบบการจองพื้นที่หน่วยความจำหลัก

จากการวิเคราะห์หน่วยความจำหลักดังกล่าวข้างต้น มีรูปแบบที่สามารถนำมาใช้ในการจัดการกับหน่วยความจำหลักได้หลายรูปแบบ ซึ่งผู้เขียนได้ศึกษาเกี่ยวกับรูปแบบการจองพื้นที่หน่วยความจำหลักจากนักวิชาการดังนี้ 1) ไพศาล โมลิสกุลมงคลและคณะ (2545: 151-158) 2)สัลยุทธ์ สว่างวรรณ (2548: 102-110) 3) อรพิน ประวัติบริสุทธิ์ (2551: 116-125) ได้ดังต่อไปนี้

6.4.1 ระบบผู้ใช้งานคนเดียวเพื่อการจัดสรรหน่วยความจำ (Single-User Contiguous Memory Allocation)

ระบบคอมพิวเตอร์ในยุคเริ่มแรกนั้น ในเวลาหนึ่งๆ จะมีผู้ใช้งานเพียง 1 คนเท่านั้นที่จะสามารถใช้งานเครื่องคอมพิวเตอร์ได้ ดังนั้นรูปแบบการจองพื้นที่หน่วยความจำหลักในรูปแบบนี้จึงมีหลักการ คือ ต้องจองพื้นที่ว่างในหน่วยความจำหลักที่ติดกันเท่าที่จำเป็นสำหรับการประมวลผลงานและต้องโหลดส่วนของงานทั้งงานของผู้ใช้งานที่ต้องการประมวลผลไปไว้ยังหน่วยความจำหลักที่ได้จองไว้ ดังนั้น หากงานมีขนาดใหญ่มากจนกระทั่งไม่สามารถจองพื้นที่หน่วยความจำหลักให้กับงานได้ ระบบก็จะไม่สามารถประมวลผลงานนั้นได้ ซึ่งหากเกิดปัญหาดังกล่าวข้างต้น จำเป็นอย่างยิ่งที่จะต้องเพิ่มขนาดของหน่วยความจำหลักให้ใหญ่ขึ้น แก้ไขงานใหม่ทำให้งานมีขนาดเล็กลง แบ่งส่วนของงานออกเป็นส่วนๆ แล้วนำแต่ละส่วนจากหน่วยเก็บข้อมูลสำรองโหลดไปไว้ในหน่วยความจำหลักเพื่อประมวลผล วิธีนี้เรียกว่า Overlay

หลักการของการวางซ้อน (Overlay) คือ จะแบ่งส่วนของงานที่จำเป็นสำหรับการเริ่มต้นทำงานออกเป็นส่วนหนึ่งและแบ่งส่วนที่เหลือตามลักษณะการทำงาน เช่น ส่วนของการประมวลผลงานและการแสดงผลลัพธ์ของงานออกเป็นอีกส่วนหนึ่ง เมื่อเริ่มต้นการทำงานก็จะโหลดส่วนที่จำเป็นสำหรับการเริ่มต้นทำงานของโปรแกรมลงสู่หน่วยความจำหลักและเริ่มประมวลผล ส่วนนี้เมื่อประมวลผลเสร็จแล้วก็จะโหลดส่วนของการประมวลผลงานทับลงไปในพื้นที่หน่วยความจำหลักเดิม สามารถสรุปการทำงานได้ดังนี้

6.4.1.1 การจองพื้นที่หน่วยความจำหลักในรูปแบบนี้ งานทุกงานที่มีขนาดเล็กกว่าหน่วยความจำหลักจะได้รับการประมวลผลทั้งหมด เพียงแต่ต้องรอทำงานตามคิวที่ละงานคือต้องทำงานแรกให้เสร็จก่อน งานต่อๆ มาจึงจะได้รับการประมวลผล

6.4.1.2 ขณะทำงานจะมีเพียง 1 งานเท่านั้นที่อยู่ในหน่วยความจำหลัก กล่าวคือ หากมีงานอยู่ 2 งาน งานที่สองต้องรอให้งานแรกทำงานเสร็จก่อนจึงจะสามารถโหลดงานที่ 2 ลงสู่หน่วยความจำหลักเพื่อประมวลผลได้

6.4.1.3 ทุกส่วนของงานจะต้องถูกโหลดให้อยู่ต่อเนื่องกันในหน่วยความจำหลัก

6.4.1.4 ถ้างานใดมีขนาดใหญ่กว่าหน่วยความจำหลัก จะต้องส่งข้อความแจ้งให้ผู้ใช้งานทราบเพื่อให้ผู้ใช้งานแก้ไขปัญหาดังกล่าว

6.4.2 พาร์ทิชันคงที่(Fixed Partitions)

การจองพื้นที่หน่วยความจำหลักในรูปแบบ Single-User Contiguous Memory Allocation นั้น หากโปรเซสมีการร้องขอติดต่อกับอุปกรณ์อินพุต / เอาต์พุตแล้ว โปรเซสนั้นก็จะกินเวลาการประมวลผลของโปรเซสเซอร์ เนื่องจากโปรเซสเซอร์จะไม่สามารถไปประมวลผลโปรเซสอื่นได้ แต่ต้องรอจนกระทั่งโปรเซสนั้นได้รับการตอบสนองจากอุปกรณ์อินพุต / เอาต์พุต (I/O speed) จะช้ากว่าความเร็วของโปรเซสเซอร์ (Processor speed) ดังนั้นจึงทำให้ใช้งานโปรเซสเซอร์ได้อย่างไม่มีประสิทธิภาพเท่าที่ควร

ต่อมาระบบมัลติโปรแกรมมิ่ง (Multiprogramming) ได้ถูกพัฒนาขึ้นจึงทำให้การใช้งานโปรเซสเซอร์มีประสิทธิภาพมากขึ้น เนื่องจากระบบมัลติโปรแกรมมิ่งจะช่วยให้ผู้ใช้งานหลายๆ คนทำงานไปพร้อมกันได้ ดังนั้นจึงมีงานหลายๆ งานอยู่ในหน่วยความจำหลักในเวลาเดียวกันเพื่อช่วยให้ใช้งานโปรเซสเซอร์ได้มีประสิทธิภาพมากขึ้น ทำให้กำลังผลิต (Throughput) ของระบบเพิ่มขึ้นโดยระบบมัลติโปรแกรมมิ่งในยุคเริ่มแรกที่ถูกพัฒนาขึ้น เรียกว่าระบบพาร์ทิชันคงที่แบบหลายโปรแกรม(Fixed Partitions Multiprogramming) บางครั้งเรียกว่า Static Partitions

รูปแบบการจองพื้นที่หน่วยความจำหลักในรูปแบบนี้มีหลักการ คือ ระบบจะแบ่งพื้นที่หน่วยความจำหลักออกเป็นหลายๆ พาร์ทิชันเมื่อสตาร์ทระบบขึ้นมาแต่ละพาร์ทิชันจะถูกกำหนดขนาดไว้ตายตัวไม่สามารถเปลี่ยนแปลงได้ โดยจะเปลี่ยนแปลงขนาดของพาร์ทิชันได้ก็ต่อเมื่อปิดระบบแล้วแก้ไขขนาดพาร์ทิชัน จากนั้นค่อยรีสตาร์ทระบบขึ้นมาใหม่ ในแต่ละพาร์ทิชันจะเก็บงานได้เพียง 1 งานเท่านั้น ซึ่งหากงานนั้นๆ ใช้งานพาร์ทิชันไม่เต็มพื้นที่ (ยังเหลือพื้นที่ว่างอยู่) งานอื่นก็จะไม่สามารถเข้ามาใช้งานพื้นที่พาร์ทิชันนั้นได้ โดยเรียกพื้นที่ว่างที่เหลือในพาร์ทิชันว่า “Internal Fragmentation” การจองพื้นที่หน่วยความจำหลักในรูปแบบพาร์ทิชันนั้นตัวจัดการหน่วยความจำ “Memory Manager” ซึ่งมีหน้าที่จัดการหน่วยความจำหลักจะต้อง

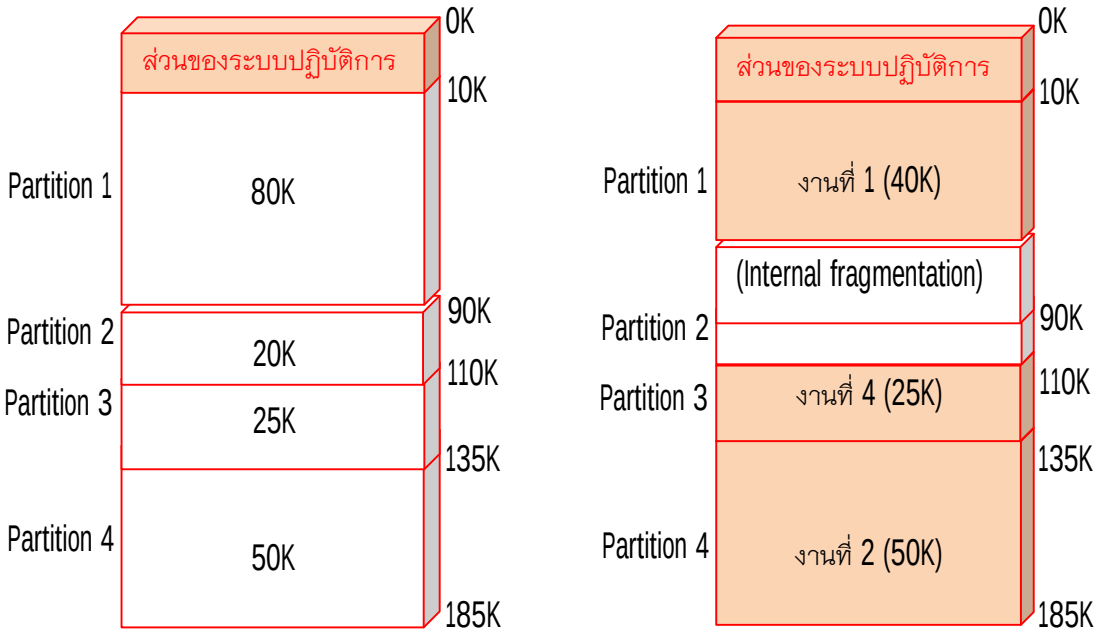
เก็บตารางแสดงขนาดของพาร์ทิชัน ที่อยู่ของพาร์ทิชันในหน่วยความจำหลัก งานที่ครอบครองพาร์ทิชันและสถานะของพาร์ทิชันดังตารางที่ 6.1

ตารางที่ 6.1 แสดงรายละเอียดของพาร์ทิชันทั้งหมด 4 พาร์ทิชัน

ขนาดพาร์ทิชัน	ที่อยู่ของพาร์ทิชันในหน่วยความจำหลัก	งานที่ครอบครองพาร์ทิชัน	สถานะของพาร์ทิชัน
80K	10K	งานที่ 1	Busy
20K	90K		Free
25K	110K	งานที่ 4	Busy
50K	135K	งานที่ 2	Busy

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 116)

จากตารางรายละเอียดของงานต่างๆ มีดังนี้ งานที่ 1 มีขนาด 40K งานที่ 2 มีขนาด 50K งานที่ 3 มีขนาด 30K และงานที่ 4 มีขนาด 25K ซึ่งสามารถแสดงภาพของหน่วยความจำหลักได้ดังต่อไปนี้



(a) หน่วยความจำหลัก (b) งานที่เข้าไปอยู่ในหน่วยความจำหลัก

ภาพที่ 6.5 แสดงหน่วยความจำหลักและงานที่เข้าไปอยู่ในหน่วยความจำหลัก

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 117)

จากภาพที่ 6.5 สามารถอธิบายการทำงานได้ดังนี้

6.4.2.1 เมื่อเริ่มต้นการทำงานทุกพาร์ทิชันจะถูกเซตให้มีสถานะเริ่มต้นเป็น Free หมายถึง ว่างสามารถเข้าใช้งานพาร์ทิชันได้

6.4.2.2 เมื่องานหนึ่งๆ ถูกส่งเข้ามา งานนั้นจะเข้าไปอยู่ในพาร์ทิชันแรกของหน่วยความจำหลักที่มีพื้นที่เพียงพอที่จะเข้าไปอยู่ได้

6.4.2.3 ดังนั้นเมื่องานที่ 1 ซึ่งมีขนาด 40K ถูกส่งเข้ามา จะพบว่าพื้นที่พาร์ทิชันแรก คือ พาร์ทิชันที่ 1 ซึ่งมีขนาด 80K ใหญ่เพียงพอที่งานที่ 1 จะเข้าไปอยู่ได้ ดังนั้นงานที่ 1 จึงถูกโหลดเข้าไปอยู่ในพาร์ทิชันนี้และเซตสถานะของพาร์ทิชันนี้ให้เป็น Busy เพื่อแสดงว่าพาร์ทิชันไม่ว่าง งานอื่นไม่สามารถใช้งานพาร์ทิชันนี้ได้

6.4.2.4 ต่อมางานที่ 2 ขนาด 50K ถูกส่งเข้ามา ซึ่งพื้นที่พาร์ทิชันแรกที่มีขนาดใหญ่เพียงพอสำหรับงานที่ 2 คือ พาร์ทิชันที่ 4 จากนั้นเซตสถานะของพาร์ทิชันที่4 ให้เป็น Busy

6.4.2.5 ต่อมางานที่ 3 ขนาด 30K ถูกส่งเข้ามา จะพบว่าแม้พาร์ทิชันที่ 1 จะมีพื้นที่เหลือ ถึง 40K เพียงพอสำหรับงานที่ 3 ก็ตาม แต่งานที่ 3 ก็ไม่สามารถเข้าไปอยู่ในพาร์ทิชันที่ 1 ได้ เนื่องจากขณะนี้พาร์ทิชันที่ 1 อยู่ในสถานะไม่ว่าง เพราะมีงานที่ 1 ใช้พื้นที่นี้อยู่ ส่วนพาร์ทิชันที่ 2 และ 3 ก็มีขนาดไม่ใหญ่พอสำหรับงานที่ 3 ดังนั้นงานที่ 3 จึงต้องรอ

6.4.2.6 ต่อมางานที่ 4 ขนาด 25K ถูกส่งเข้ามา ซึ่งพาร์ทิชันที่ 3 มีขนาดใหญ่พอสำหรับงานที่ 4 ดังนั้น งานที่ 4 จึงถูกโหลดเข้าไปอยู่ในพาร์ทิชันที่ 3 และเซตสถานะของพาร์ทิชันที่ 3 เป็น Busy

6.4.2.7 ซึ่งต่อมาหากงานใดประมวลผลเสร็จแล้ว ก็จะต้องเปลี่ยนสถานะของพาร์ทิชันที่งานนั้นเคยอาศัยอยู่ จาก Busy ให้กลับเป็น Free เหมือนเดิม เพื่อให้งานอื่นสามารถเข้าไปใช้งานพื้นที่พาร์ทิชันนั้นได้

จากการทำงานของหน่วยความจำแบบ Fixed Partitions สามารถสรุปการทำงานได้ว่าหากงานใดมีหนาดใหญ่กว่าขนาดของพาร์ทิชันที่ใหญ่ที่สุดก็จะไม่สามารถเข้ามายังหน่วยความจำหลักได้ กล่าวคือ งานนั้นก็จะไม่ได้รับการประมวลผล ซึ่งงานที่จะเข้ามายังพาร์ทิชันได้จะต้องมีขนาดน้อยกว่าหรือเท่ากับขนาดของพาร์ทิชัน ถ้างานใดยังไม่สามารถหาพาร์ทิชันลงได้เนื่องจากพาร์ทิชันที่จะลงได้มีสถานะเป็น Busy งานนั้นจะต้องรอจนกว่าพาร์ทิชันนั้นอยู่ในสถานะ Free สำหรับแต่ละงานที่ใช้พาร์ทิชันใดๆ เมื่อมีพื้นที่ว่างเหลือในพาร์ทิชันนั้นๆ (Internal Fragmentation) ก็จะถือว่าพื้นที่ว่างนี้ทำอะไรไม่ได้เลย งานอื่นก็ไม่สามารถเข้ามาใช้งานพื้นที่นี้ได้

6.4.3 พาร์ทิชันแบบไดนามิก (Dynamic Partitions)

การจองพื้นที่หน่วยความจำในรูปแบบ Fixed Partitions มีปัญหาหลายๆ เรื่อง เช่น ขนาดของพาร์ทิชันอาจจะเล็กมากจนกระทั่งงานไม่สามารถเข้าไปอยู่ได้ หรือไม่ก็ขนาดพาร์ทิชันอาจจะใหญ่มากเกินไปในขณะที่งานมีขนาดเล็กจึงก่อให้เกิดปัญหาขึ้น (เกิดพื้นที่ว่าง Internal Fragmentation ขนาดใหญ่) ดังนั้นเพื่อลดปัญหาดังกล่าวจึงได้มีการพัฒนาการจองพื้นที่หน่วยความจำหลักในรูปแบบไดนามิกขึ้น ซึ่งการจองพื้นที่หน่วยความจำหลักในรูปแบบนี้จะไม่มีการแบ่งพื้นที่หน่วยความจำหลักออกเป็นพาร์ทิชันแต่จะอนุญาตให้แต่ละงานใช้พื้นที่หน่วยความจำหลักได้ตามที่ต้องการโดยยังมีข้อแม้อยู่เหมือนเดิมที่ว่างานหนึ่งๆ จะต้องมีการวางอยู่ติดกันในหน่วยความจำหลักไม่สามารถแยกงานออกเป็นส่วนๆ แล้ววางกระจายกันได้ เพื่อความเข้าใจในการทำงานของการจองพื้นที่หน่วยความจำหลักในรูปแบบไดนามิกจะอธิบายโดยยกตัวอย่างดังต่อไปนี้

สมมติในตอนเริ่มต้นมีงานทั้งหมด 4 งานเข้ามาในระบบ คือ งานที่ 1 มีขนาด 10K งานที่ 2 มีขนาด 20K งานที่ 3 มีขนาด 10K และงานที่ 4 มีขนาด 5K ซึ่งสามารถแสดงภาพของหน่วยความจำหลักได้ดังต่อไปนี้

ภาพที่ 6.6 แสดงการจองพื้นที่หน่วยความจำหลักในรูปแบบ Dynamic Partitions ในตอนเริ่มต้น

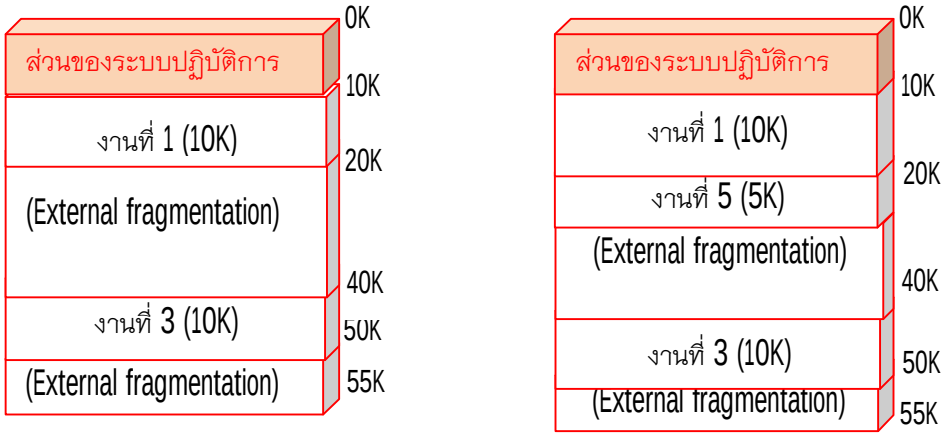
ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 120)

สำหรับการจองพื้นที่หน่วยความจำหลักในรูปแบบ Dynamic partitions ในตอนเริ่มต้นนั้นงานแต่ละงานจะเข้าไปอยู่ในหน่วยความจำหลักตามลำดับก่อนหลัง ตามรูปแบบของ FCFS ดังนั้น จากภาพที่ 6.6 พบว่างานที่ 1,2,3 และ 4 เข้าไปอยู่ในหน่วยความจำหลัก

ตามลำดับก่อนหลังของงานที่เข้ามา โดยสังเกตว่าแต่ละงานสามารถใช้พื้นที่หน่วยความจำหลักตามที่งานนั้นๆ ต้องการ

ต่อมางานที่ 2 และงานที่ 4 ประมวลผลเสร็จ จึงได้ออกจากหน่วยความจำหลักไป ทำให้เกิดช่องว่างขึ้นในหน่วยความจำหลัก เรียกว่า External Fragmentation ดังภาพที่ 6.7(a) ซึ่งช่องว่างนี้สามารถจัดสรรให้กับงานอื่นใช้ได้ ถ้าหากว่าช่องว่างนี้มีขนาดใหญ่เพียงพอกับขนาดของงานนั้นๆ โดยที่สถานะของช่องว่างนั้นเป็น Free

ในเวลาต่อมามีงานที่ 5 และงานที่ 6 เข้ามายังระบบ โดยงานที่ 5 มีขนาด 5K และงานที่ 6 มีขนาด 20K เนื่องจากงานที่ 2 ได้ประมวลผลเสร็จและออกไปจากหน่วยความจำหลักแล้ว จึงทำให้เกิดช่องว่างขึ้นในพื้นที่หน่วยความจำหลักขนาด 20K ซึ่งงานที่ 5 สามารถเข้าไปอยู่ในส่วนของ External Fragmentation ส่วนนี้ได้ เนื่องจากช่องว่างนี้เป็นพื้นที่แรกที่ว่างและมีขนาดเพียงพอกับขนาดของงานที่ 5 ได้เข้าไปอยู่ในพื้นที่หน่วยความจำหลัก แล้วจะทำให้เหลือพื้นที่ External Fragmentation ระหว่างงานที่ 5 และงานที่ 3 จำนวน 15K และยังเหลือพื้นที่ External Fragmentation ใต้งานที่ 3 อีกจำนวน 5K ดังภาพที่ 6.7 (b) ซึ่งจะพบว่าพื้นที่ External Fragmentation ทั้งสองส่วนที่เหลือว่างอยู่นี้ หากรวมกันก็จะมีขนาดเพียงพอกับงานที่ 6 ซึ่งมีขนาด 20K แต่เนื่องจากพื้นที่ว่างทั้งสองนี้ไม่ได้อยู่ติดกัน ดังนั้นงานที่ 6 จึงไม่สามารถเข้ามาในหน่วยความจำหลักได้ กรณีนี้งานที่ 6 จึงต้องรอ



(a) แสดง External fragmentation เมื่องานที่ 2 และ 4 ประมวลผลเสร็จ

(b) แสดง External fragmentation เมื่องานที่ 5 เข้ามายังระบบ

ภาพที่ 6.7 แสดงพื้นที่ External fragmentation

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 121)

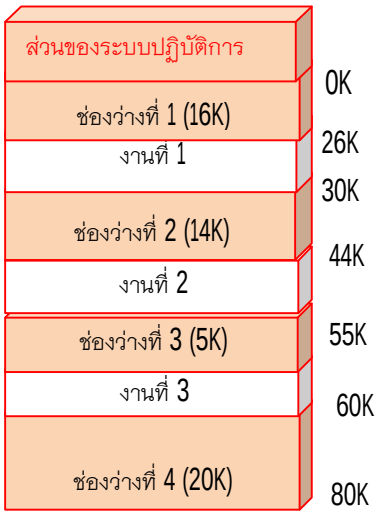
จากหลักการจองพื้นที่หน่วยความจำหลักในรูปแบบ Dynamic partitions ในข้างต้น เพื่อแก้ไขปัญหาพื้นที่ว่างที่ไม่สามารถใช้งานได้ จึงแบ่งรูปแบบ Dynamic partitions ได้เป็น 3 วิธีคือ

6.4.3.1 First-Fit เมื่อมีงานเข้ามาในระบบ งานนั้นจะถูกเก็บลงในพื้นที่หน่วยความจำหลัก ส่วนแรกที่ว่างและมีขนาดเพียงพอที่งานนั้นจะเข้าไปอยู่ได้ การจองพื้นที่หน่วยความจำในรูปแบบนี้ จะทำงานได้ง่ายและรวดเร็ว แต่ก็อาจทำให้เกิดพื้นที่ของ External Fragmentation ขนาดใหญ่ที่ใช้ประโยชน์ไม่ได้ก็เป็นได้ เรียกได้ว่าใช้งานพื้นที่หน่วยความจำหลักได้ไม่คุ้มค่าเท่าที่ควร

6.4.3.2 Best-Fit เมื่อมีงานเข้ามาในระบบงานนั้นจะถูกเก็บลงในพื้นที่หน่วยความจำหลักที่มีขนาดเล็กที่สุด ซึ่งมีขนาดเพียงพอที่งานนั้นจะเข้าไปอยู่ได้ ทั้งนี้ก็เพื่อให้เกิดพื้นที่ของ External Fragmentation น้อยที่สุดเท่าที่จะเป็นไปได้ แต่วิธีนี้ก็มีข้อเสียตรงที่จะมี Overhead เกิดขึ้นจากการค้นหาพื้นที่หน่วยความจำหลักที่มีขนาดเล็กที่สุดดังกล่าว

6.4.3.3 Worst-Fit เมื่อมีงานเข้ามาในระบบ งานนั้นจะถูกเก็บลงในพื้นที่หน่วยความจำหลักที่มีขนาดใหญ่ที่สุด ซึ่งมีขนาดเพียงพอที่งานนั้นจะเข้าไปอยู่ได้ โดยการกระทำเช่นนี้ หากงานมีขนาดเล็กแต่ถูกเก็บลงในพื้นที่หน่วยความจำหลักที่มีขนาดใหญ่ที่สุดแล้วก็จะทำให้เกิดพื้นที่ของ External Fragmentation ที่ใหญ่ บางครั้งก็อาจก่อให้เกิดการใช้งานพื้นที่หน่วยความจำได้ไม่คุ้มค่า แต่บางครั้งการทำให้เกิดพื้นที่ของ External Fragmentation ขนาดใหญ่ ก็อาจเป็นเหตุผลเพื่อให้สามารถนำงานอื่นมาใส่ได้โดยง่าย โดยวิธีนี้ยังมีข้อเสียเหมือนวิธีของ Best-Fit ด้วย ตรงที่จะมี Overhead เกิดขึ้นจากการค้นหาพื้นที่หน่วยความจำหลักที่มีขนาดใหญ่ที่สุด ซึ่งสามารถอธิบายได้ตามตัวอย่างดังต่อไปนี้

สมมติขณะนี้พื้นที่หน่วยความจำหลักเก็บงานดังภาพที่ 6.8 หากมีงานขนาด 12K เข้ามายังระบบแล้ว การจองพื้นที่หน่วยความจำหลักในรูปแบบ Dynamic partitions ด้วยวิธี First-Fit งานจะเข้าไปอยู่ในช่องว่างที่ 1 ขนาด 16K ของหน่วยความจำหลัก ส่วน Best-Fit งานจะเข้าไปอยู่ในช่องว่างที่ 2 ขนาด 14K ของหน่วยความจำหลักและสำหรับ Worst-Fit งานจะเข้าไปอยู่ช่องว่างที่ 4 ขนาด 20K ของหน่วยความจำหลัก

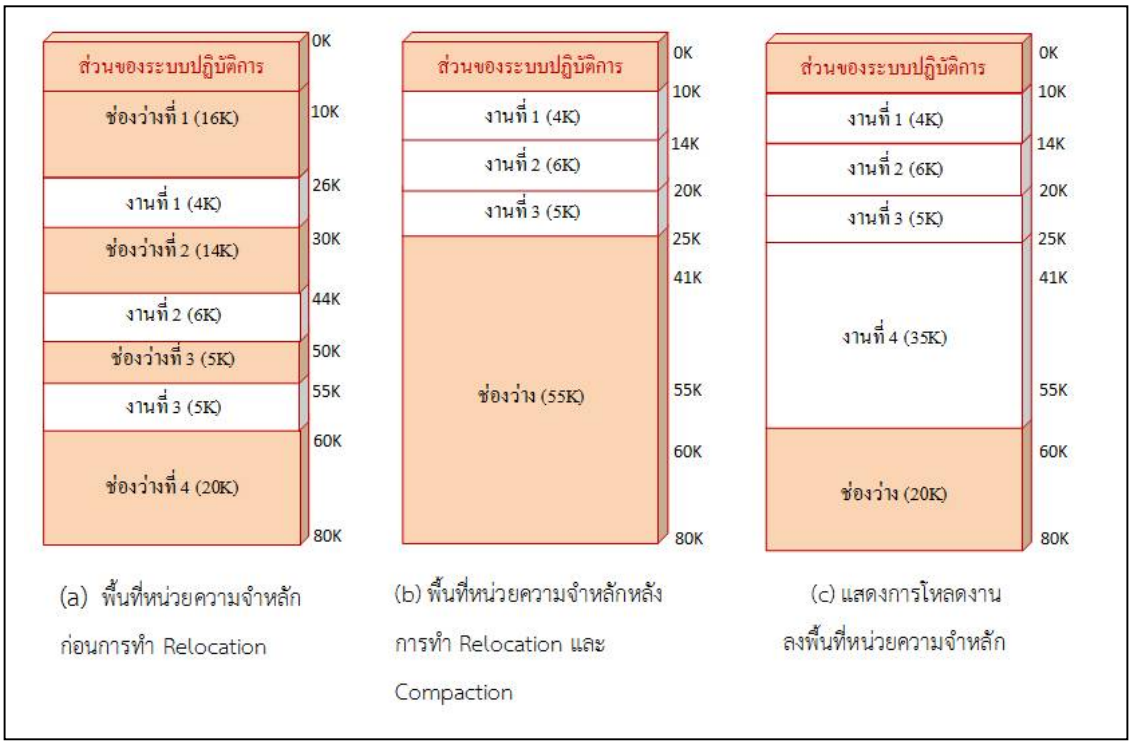


ภาพที่ 6.8 แสดงการเก็บงานของพื้นที่หน่วยความจำหลัก
ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 123)

6.4.4 พาร์ทิชันแบบไดนามิก Reloadable (Reloadable Dynamic Partitions)

การจองพื้นที่หน่วยความจำหลักในรูปแบบพาร์ทิชันไดนามิก มีปัญหาในเรื่องของ External Fragmentation เกิดขึ้น ทำให้ใช้พื้นที่หน่วยความจำหลักได้ไม่เต็มประสิทธิภาพเนื่องจากบางครั้งบล็อกว่างที่เหลือมีขนาดไม่ใหญ่พอที่งานจะเข้าไปอยู่ได้ ดังนั้นเพื่อลดปัญหาดังกล่าวจึงได้มีการพัฒนาการจองพื้นที่หน่วยความจำหลักในรูปแบบ Relocatable Dynamic Partitions ขึ้น

การจองพื้นที่หน่วยความจำในรูปแบบนี้มีหลักการ คือ Memory Manager ซึ่งมีหน้าที่ดูแลหน่วยความจำหลักและจะขยับพื้นที่ของโปรแกรมในหน่วยความจำหลักเสียใหม่ โดยขยับให้โปรแกรมมาอยู่ติดกันโดยการปรับตำแหน่งที่อยู่ของโปรแกรมใหม่รวมถึงการปรับอ้างอิงตำแหน่งที่อยู่ข้างๆ ภายในโปรแกรมใหม่ด้วย เรียกว่า Relocation จากนั้นก็จะรวบรวมพื้นที่บล็อกว่างต่างๆ ที่กระจายกันอยู่ในหน่วยความจำหลัก ให้รวมเข้าด้วยกันเป็นบล็อกใหญ่เพียงบล็อกเดียว เรียกว่า Compaction (บางครั้งเรียกว่า Garbage Collection หรือ Defragmentation) สามารถแสดงได้ดังตัวอย่างพิจารณาการจองพื้นที่หน่วยความจำหลักในรูปแบบ Relocate Dynamic Partitions ดังนี้



ภาพที่ 6.9 แสดงการจองพื้นที่ในหน่วยความจำหลัก
ที่มา : อรพิน ประวัติบริสุทธิ์(2551: 132)

สมมติมีงานที่ 4 ขนาด 35K เข้ามายังระบบ จากภาพที่ 6.9(a) จะพบว่าไม่สามารถโหลดงานลงสู่หน่วยความจำหลักได้เนื่องจากไม่มีพื้นที่ว่างที่มีขนาดใหญ่พอที่งานจะเข้าไปอยู่ แต่หลังจากทำ Relocation และ Compaction ดังภาพที่ 6.9(b) แล้ว ช่องว่างที่ 1 ถึง 4 จะมารวมกันเป็นบล็อกใหญ่บล็อกเดียวขนาด 55K ทำให้โหลดงานที่ 4 ลงในหน่วยความจำหลักได้ ดังภาพที่ 6.9 (c)

เหตุผลที่ทำ Relocation และ Compaction ก็เพื่อทำให้พื้นที่บล็อกว่างมีขนาดใหญ่เพียงพอที่โปรแกรมต่างๆจะเข้าไปอยู่ได้ แต่ก็มีข้อเสียคือก่อให้เกิด Overhead ขึ้น และในขณะที่ทำ Compaction จะทำงานอื่นไม่ได้จะต้องคอยจนกระทั่งทำ Compaction เสร็จจึงจะสามารถทำงานต่อไปได้ หลังจากทำ Relocation และ Compaction แล้วรายการที่แสดงสถานะ Free และ Busy จะต้องถูกปรับปรุง (Update) โดยรายการแสดงสถานะ Free จะแสดงพาร์ทิชันของบล็อกว่างบล็อกใหญ่ซึ่งเป็นผลมาจากการทำ Compaction โดยตำแหน่งเริ่มต้นของบล็อกว่างจะมีตำแหน่งต่อจากตำแหน่งสุดท้ายของงานสุดท้ายในหน่วยความจำหลัก ส่วนรายการที่แสดงสถานะ Busy จะแสดงรายละเอียดต่างๆ ของงานที่อยู่ในหน่วยความจำหลัก โดยจะแสดงตำแหน่งที่อยู่ของงานด้วยตำแหน่งใหม่ที่ได้หลังจากการทำ Relocation แล้ว

ในการทำงานของระบบปฏิบัติการมีการนำฮาร์ดแวร์มาช่วยในการทำงานเสมอซึ่งการทำ Relocation และ Compaction ก็เช่นกัน จะนำรีจิสเตอร์มาช่วยในการเก็บข้อมูล โดยรีจีสเตอร์ที่ถูกนำมาใช้ คือ 1) Bound register ใช้เก็บค่าตำแหน่งสูงสุด (ระบบปฏิบัติการบางตัวก็ใช้เก็บค่าตำแหน่งต่ำสุด) ของหน่วยความจำหลักที่แต่ละงานสามารถใช้งานได้ เหตุผลที่ต้องเก็บก็เพื่อป้องกันไม่ให้งานที่กำลังประมวลผลอยู่ไปใช้งานพื้นที่หน่วยความจำหลักที่ไม่ได้เป็นของงานนั้น กล่าวคือ ป้องกันไม่ให้งานหนึ่งๆ ใช้งานพื้นที่หน่วยความจำหลักเกินขอบเขตที่กำหนดไว้ให้กับงานนั้น 2) Relocation register ใช้เก็บค่าตำแหน่งของงานว่างงานเปลี่ยนตำแหน่งไปเท่าใด โดยค่านี้จะถูกเก็บไว้เพื่อนำมาใช้ในงาน กล่าวคือ หากภายในงานมีการอ้างอิงตำแหน่งที่อยู่ต่างๆก็จะต้องนำค่าที่อยู่ใน Relocation register นี้บวกเข้าไปกับตำแหน่งที่อยู่นั้นๆ ด้วย เพื่อที่งานจะได้อ้างอิงตำแหน่งที่ถูกต้องหลังจากทำ Relocation แล้ว ซึ่งหากงานนั้นไม่มีการทำ Relocation ค่าของ Relocation Register ก็จะมีค่าเป็นศูนย์

6.5 การคืนพื้นที่หน่วยความจำหลัก

เมื่องานหนึ่งๆ ประมวลผลเสร็จเรียบร้อยแล้วงานนั้นจะต้องคืนพื้นที่หน่วยความจำหลักให้กับระบบ ซึ่งผู้สอนได้ศึกษาเกี่ยวกับการคืนพื้นที่หน่วยความจำหลักจากนักวิชาการดังนี้ 1) อรพิน ประวัติบริสุทธิ์ (2551: 126-135) 2) พิรพร หมุนสนิทและคณะ (2553: 110-123) แล้วนำมาสรุปเกี่ยวกับการคืนพื้นที่หน่วยความจำหลักให้กับระบบของ Dynamic Partitions นั้นจะซับซ้อนกว่าแบบ Fixed partitions กล่าวคือ แบบ Fixed partitions นั้นจะแบ่งพื้นที่หน่วยความจำหลักออกเป็นพาร์ทิชัน โดยที่แต่ละพาร์ทิชันมีขนาดคงที่ไม่สามารถเปลี่ยนแปลงขนาดได้ ดังนั้นการคืนพื้นที่หน่วยความจำหลักให้กับระบบของ Fixed Partitions นั้นก็เพียงแค่เปลี่ยนสถานะของพาร์ทิชันจาก Busy ให้กลับมาเป็น Free เท่านั้น แต่สำหรับ Dynamic Partitions แล้วการคืนพื้นที่หน่วยความจำหลักมีความซับซ้อนมากกว่านั้น เนื่องจากไม่ได้แบ่งพื้นที่หน่วยความจำหลักออกเป็นพาร์ทิชัน

การคืนพื้นที่หน่วยความจำหลักของ Dynamic Partitions แบ่งออกได้เป็น 3 กรณี

6.5.1 **กรณีที่ 1** เมื่อบล็อกของงานที่ต้องการคืนพื้นที่หน่วยความจำหลักให้กับระบบอยู่ติดกับบล็อกว่างหนึ่งๆ โดยอาจจะเป็นบล็อกว่างที่อยู่เหนือบล็อกนี้หรืออยู่ใต้บล็อกนี้

ตารางที่ 6.2 แสดงการสมมติรายการแสดงสถานะ Free ของบล็อกของหน่วยความจำหลัก

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7560	20	Free
7800	5	Free
7865	5	Free

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 126)

ตารางที่ 6.3 แสดงรายการแสดงสถานะ Busy ของบล็อกของหน่วยความจำหลัก

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7580	20	Busy
7600	200	Busy
7805	5	Busy
7810	50	Busy
7860	5	Busy

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 126)

ในการทำงานของระบบปฏิบัติการ จะแยกรายการแสดงสถานะ Free และ Busy ของบล็อกของหน่วยความจำหลักออกจากกัน แต่เพื่อให้ง่ายต่อการอธิบาย จะขอนำรายการแสดงสถานะ Free และ Busy มารวมกัน แสดงดังตารางที่ 6.4

ตารางที่ 6.4 แสดงรายการแสดงสถานะ Free และ Busy รวมกัน

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7560	20	Free
7580	20	Busy
*7600	200	Busy
**7800	5	Free
7805	5	Busy
7810	50	Busy
7860	5	Busy
7865	5	Free

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 127)

ต่อมางานที่ 1 ซึ่งมีตำแหน่งเริ่มต้นในหน่วยความจำหลักเป็น 7600 ได้รับการประมวลผลเสร็จแล้ว และต้องการคืนหน่วยความจำหลักให้กับระบบ จากการพิจารณาพบว่ามีบล็อกว่างที่อยู่ใต้บล็อกนี้ ได้แก่ บล็อกที่มีตำแหน่งเริ่มต้นเป็น 7800 โดยหลังจากงานที่ 1 คืนพื้นที่หน่วยความจำหลักแล้ว ตำแหน่งเริ่มต้นจะมีค่าเท่ากับตำแหน่งที่น้อยที่สุดจาก * กับ ** นำมาเปรียบเทียบกัน ซึ่งในที่นี้ตำแหน่งเริ่มต้นของ * เท่ากับ 7600 และตำแหน่งเริ่มต้นของ ** เท่ากับ 7800 ดังนั้น หลังจากงานที่ 1 คืนหน่วยความจำหลักให้กับระบบแล้ว ตำแหน่งเริ่มต้นจะมีค่าเป็น 7600 ส่วนค่าขนาดบล็อกของหน่วยความจำหลักจะมีค่าเป็น 205 ซึ่งมาจากขนาดบล็อกของหน่วยความจำหลักในตำแหน่ง * บวกกับขนาดบล็อกของหน่วยความจำหลักในตำแหน่ง ** ดังนั้นการทำงานจะได้ผลดังตารางที่ 6.5

ตารางที่ 6.5 แสดงผลการจัดบล็อกในหน่วยความจำหลัก

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7560	20	Free
7580	20	Busy
7600	205	Free
7805	5	Busy
7810	50	Busy
7860	5	Busy
7865	5	Free

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 128)

6.5.2 กรณีที่ 2 เมื่อบล็อกของงานที่ต้องการคืนพื้นที่หน่วยความจำหลักให้กับระบบอยู่ติดกับบล็อกว่าง 2 บล็อก คือ บล็อกที่อยู่เหนือบล็อกนี้ก็ว่างและบล็อกที่อยู่ใต้บล็อกนี้ก็ว่างเพื่อให้ง่ายต่อการอธิบาย จึงขอยกตัวอย่าง ดังตารางที่ 6.6

ตารางที่ 6.6 แสดงการสมมติสถานะของบล็อกของหน่วยความจำหลัก

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
*7560	20	Free
**7580	20	Busy
***7600	205	Free
7805	5	Busy
7810	50	Busy
7860	5	Busy
7865	5	Free

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 128)

ต่อมางานที่ 2 ซึ่งมีตำแหน่งเริ่มต้นในหน่วยความจำหลักเป็น 7580 ได้รับการประมวลผลเสร็จแล้ว และต้องการคืนหน่วยความจำหลักให้กับระบบ จากการพิจารณาพบว่ามีบล็อกว่างที่อยู่เหนือและใต้บล็อกนี้ ได้แก่ บล็อกที่มีตำแหน่งเริ่มต้นเป็น 7560 และ 7600 เมื่องานที่ 2 คืนพื้นที่หน่วยความจำหลักแล้ว ตำแหน่งเริ่มต้นจะมีค่าเท่ากับตำแหน่งที่น้อยที่สุดจาก *, ** และ *** นำมาเปรียบเทียบกัน ซึ่งในที่นี้ตำแหน่งเริ่มต้นของ * เท่ากับ 7560, ตำแหน่งเริ่มต้นของ ** เท่ากับ 7580 และตำแหน่งเริ่มต้นของ *** เท่ากับ 7600 ดังนั้นหลังจากคืนหน่วยความจำหลักให้กับระบบแล้ว ตำแหน่งเริ่มต้นจะมีค่าเป็น 7560

ส่วนค่าขนาดบล็อกของหน่วยความจำหลักจะมีค่าเป็น 245 ซึ่งมาจากขนาดบล็อกของหน่วยความจำหลักในตำแหน่ง * บวกกับขนาดบล็อกของหน่วยความจำหลักในตำแหน่ง ** และ *** ดังนั้นการทำงานจะได้ผลดังตารางที่ 6.7

ตารางที่ 6.7 แสดงผลการจัดบล็อกในหน่วยความจำหลัก

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7560	245	Free
7805	5	Busy
7810	50	Busy
7860	5	Busy
7865	5	Free

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 129)

ซึ่งจากที่กล่าวมาแล้วข้างต้นว่า การทำงานของระบบปฏิบัติการจะแยกรายการแสดงสถานะ Free และ Busy ของบล็อกของหน่วยความจำหลักออกจากกัน ดังนั้น สามารถแสดงรายการสถานะ Free และ Busy ได้ตารางที่ 6.8

ตารางที่ 6.8 แสดงรายการแสดงสถานะ Free ของบล็อกของหน่วยความจำหลัก

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7560	245	Free
		Null
7865	5	Free

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 129)

ตารางที่ 6.9 แสดงรายการแสดงสถานะ Busy ของบล็อกของหน่วยความจำหลัก

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7805	5	Busy
7810	50	Busy
7860	5	Busy

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 130)

จากรายการแสดงสถานะของ Free จะพบว่าบรรทัดที่ 2 ของรายการไม่มีตำแหน่งเริ่มต้น ไม่มีขนาดบล็อกของหน่วยความจำหลัก แต่เซตสถานะของบล็อกให้เป็น Null เหตุผลที่ทำเช่นนี้เนื่องจากตำแหน่งของหน่วยความจำหลักตั้งแต่ 7805 ถึง 7864 มีสถานะของบล็อกไม่ใช่ Free ดังนั้นจึงเซตเป็น Null ไว้ เพราะหากต่อมามีบล็อกใดๆ ของช่วงตำแหน่ง 7805 ถึง 7864 เปลี่ยนสถานะเป็น Free จะได้ไม่ต้องเสียเวลาจัดเรียงรายการแสดงสถานะ Free ใหม่

6.5.3 กรณีที่ 3 เมื่อบล็อกของงานที่ต้องการคืนพื้นที่หน่วยความจำหลักให้กับระบบไม่ได้อยู่ติดกับบล็อกว่างใดๆ เลย สมมติรายการแสดงสถานะ Free ของบล็อกของหน่วยความจำหลักเป็น ดังตารางที่ 6.10

ตารางที่ 6.10 แสดงรายการแสดงสถานะ Free ของบล็อกของหน่วยความจำหลัก

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7560	245	Free
		Null
7865	5	Free

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 130)

ตารางที่ 6.11 แสดงรายการแสดงสถานะ Busy ของบล็อกของหน่วยความจำหลัก

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7805	5	Busy
7810	50	Busy
7860	5	Busy

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 131)

ต่อมางานที่ 3 ซึ่งมีตำแหน่งเริ่มต้นของหน่วยความจำหลัก 7810 ได้รับการประมวลผลเสร็จแล้ว และต้องการคืนหน่วยความจำหลักให้กับระบบ ดังนั้นจึงเปลี่ยนสถานะของบล็อกที่ตำแหน่งตั้งแต่ 7810-7859 จาก Busy ไปเป็น Free และนำบล็อกนี้ไปใส่ไว้ในรายการแสดงสถานะ Free ดังตารางที่ 6.12

ตารางที่ 6.12 แสดงรายการแสดงสถานะ Free

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7560	245	Free
7810	50	Free
7865	5	Free

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 131)

จากนั้นเซตบรรทัดของรายการเมื่อสักครู่ที่มีอยู่ในรายการแสดงสถานะ Busy ให้เป็น Null เพื่อที่หากต่อมามีบล็อกใดๆ ของช่วงตำแหน่ง 7810 ถึง 7859 เปลี่ยนสถานะเป็น Busy จะได้ไม่ต้องเสียเวลาจัดเรียงรายการแสดงสถานะ Busy ใหม่

ตารางที่ 6.13 แสดงรายการแสดงสถานะ Busy ใหม่

ตำแหน่งเริ่มต้น	ขนาดบล็อกของหน่วยความจำหลัก	สถานะของบล็อก
7805	5	Busy
		Null
7860	5	Busy

ที่มา : อรพิน ประวัติบริสุทธิ์ (2551: 126-135)

6.6 บทสรุป

หน่วยความจำหลัก หมายถึง พื้นที่สำหรับจัดเก็บข้อมูลแบบชั่วคราวในขณะการประมวลผล มีหน่วยในการเก็บข้อมูลเป็นไบต์ (Byte) โดยแต่ละไบต์จะมีหมายเลขบอกตำแหน่ง (Address) ของหน่วยความจำในการทำงานจะมีซีพียูเป็นตัวควบคุมการทำงานของหน่วยความจำ คือ อ่านและเขียนข้อมูลลงหน่วยความจำหลักผ่านเส้นทางการรับ-ส่งข้อมูลที่เรียกว่า “บัส”(Bus) หน่วยความจำหลักมี 2 ชนิด แบบลบเลือนได้เมื่อไม่มีกระแสไฟฟ้ามาหล่อเลี้ยง เรียกว่า RAM และแบบลบเลือนไม่ได้แม้ไม่มีกระแสไฟฟ้ามาหล่อเลี้ยงเรียกว่า ROM การศึกษาหน่วยความจำหลักจะต้องทำความเข้าใจกับโครงสร้างการทำงานของหน่วยความจำ ได้แก่ ลำดับชั้นของหน่วยความจำ ความสามารถด้านฮาร์ดแวร์ การเชื่อมโยงแอดเดรส

ในการทำงานของหน่วยความจำหลักในกรณีที่มีหลายๆ โปรเซสเข้ามาจะต้องทำการวิเคราะห์การจัดการหน่วยความจำหลักเพื่อหารูปแบบที่สามารถนำมาใช้จัดการกับหน่วยความจำหลักได้หลายรูปแบบ ซึ่งรูปแบบการจองพื้นที่หน่วยความจำหลัก ได้แก่ 1) Single-User Contiguous Memory Allocation 2) Fixed Partitions 3) Dynamic Partitions 4) Reloadable Dynamic Partitions เมื่อโปรเซสนั้นดำเนินการเสร็จเป็นที่เรียบร้อยแล้ว ระบบปฏิบัติการจะต้องทำการคืนพื้นที่หน่วยความจำหลักเพื่อให้งานในโปรเซสอื่นๆ ได้เข้ามาให้บริการหน่วยความจำหลักต่อไป

6.7 แบบฝึกหัด

ตอนที่ 1 ให้ผู้เรียนตอบคำถามจากโจทย์ที่กำหนดต่อไปนี้ด้วยตนเอง

1. อธิบายความหมายของหน่วยความจำหลัก
2. อธิบายหลักการในการวิเคราะห์การจัดการหน่วยความจำหลัก
3. อธิบายสาระสำคัญของขั้นตอนการจัดการหน่วยความจำหลักแบบ First-Fit
4. อธิบายสาระสำคัญของขั้นตอนการจัดการหน่วยความจำหลักแบบ Best-Fit
5. อธิบายสาระสำคัญของขั้นตอนการจัดการหน่วยความจำหลักแบบ Worst-Fit

ตอนที่ 2 ให้ผู้เรียนแบ่งกลุ่มๆ ละ 4 คน เพื่อแสดงวิธีการแก้ปัญหาจากโจทย์ที่กำหนดต่อไปนี้แล้วจัดทำรายงานส่งในสัปดาห์ถัดไป

1. สมมติว่าระบบมีการจองพื้นที่หน่วยความจำหลักในรูปแบบ Fixed Partitions และหน่วยความจำหลักของระบบถูกแบ่งออกเป็น 5 พาร์ทิชัน ดังนี้

ขนาดพาร์ทิชัน	ที่อยู่ของพาร์ทิชันในหน่วยความจำหลัก	งานที่ครอบครองพาร์ทิชัน	สถานะของพาร์ทิชัน
10K	10K	-	Free
100K	20K	-	Free
30K	120K	-	Free
20K	150K	-	Free
10K	170K	-	Free

1.1 จงวาดรูปแสดงว่าหน่วยความจำหลักของระบบมีหน้าตาเป็นอย่างไร

1.2 เมื่อเกิดกรณีต่างๆ ดังต่อไปนี้ขึ้นจงวาดรูปแสดงว่าหน่วยความจำหลักของระบบมีหน้าตาเป็นอย่างไร

1.2.1 งานที่ 1 มีขนาด 10K ถูกส่งเข้ามา

1.2.2 งานที่ 2 มีขนาด 20K ถูกส่งเข้ามา

1.2.3 งานที่ 3 มีขนาด 30K ถูกส่งเข้ามา

1.2.4 งานที่ 4 มีขนาด 40K ถูกส่งเข้ามา

1.2.5 งานที่ 5 มีขนาด 10K ถูกส่งเข้ามา

1.3 เมื่อเกิดกรณีต่างๆ ดังต่อไปนี้ขึ้นจงวาดรูปแสดงว่าหน่วยความจำหลักของระบบมีหน้าตาเป็นอย่างไร

1.3.1 งานที่ 1,2,3 และ 5 ประมวลผลเสร็จ

1.3.2 งานที่ 6 มีขนาด 60K ถูกส่งเข้ามา

1.4 เมื่อเกิดกรณีต่างๆ ดังต่อไปนี้ขึ้นจงวาดรูปแสดงว่าหน่วยความจำหลักของระบบมีหน้าตาเป็นอย่างไร

1.4.1 งานที่ 4 ประมวลผลเสร็จ

1.4.2 งานที่ 7 มีขนาด 10K ถูกส่งเข้ามา

2. สมมติว่าระบบมีการจองพื้นที่หน่วยความจำหลักในรูปแบบ Dynamic Partitions เมื่อเกิดกรณีต่างๆ ดังต่อไปนี้ขึ้น จงวาดรูปแสดงว่าหน่วยความจำหลักของระบบมีหน้าตาเป็นอย่างไร

2.1 ในตอนเริ่มต้น

2.1.1 งานที่ 1 มีขนาด 10K ถูกส่งเข้ามา

2.1.2 งานที่ 3 มีขนาด 30K ถูกส่งเข้ามา

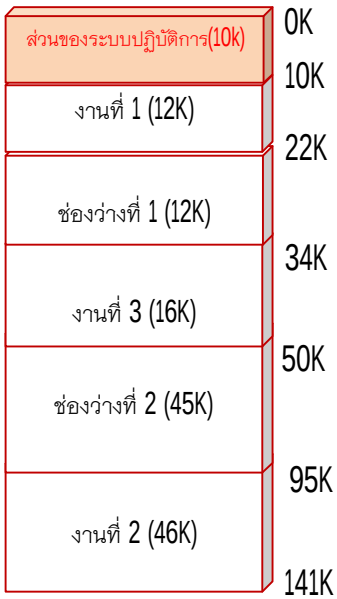
2.1.3 งานที่ 4 มีขนาด 40K ถูกส่งเข้ามา

2.2 ต่อมางานที่ 5 ประมวลผลเสร็จ และงานที่ 6 มีขนาด 25K ถูกส่งเข้ามา และงานที่ 7 มีขนาด 25K ถูกส่งเข้ามา

2.3 ต่อมางานที่ 1 และ 6 ประมวลผลเสร็จ และงานที่ 8 ขนาด 35K ถูกส่งเข้ามา

2.4 ต่อมางานที่ 7 ประมวลผลเสร็จ

3. สมมติขณะนี้พื้นที่หน่วยความจำหลักเก็บงานดังต่อไปนี้ ให้ตอบคำถามต่อไปนี้



3.1 จงแสดงว่าหลังทำ Relocation และ Compaction หน่วยความจำหลักของระบบจะมีหน้าตาเป็นอย่างไร

3.2 Relocation Registers ของงานที่ 2 มีค่าเท่าใด เพราะอะไร

ตอนที่ 3 ให้ผู้เรียนสะท้อนคิดถึงสิ่งที่ได้เรียนรู้ในบทที่ 6 แล้วนำมาเขียนในลักษณะความเรียงจัดเก็บไว้ในแฟ้มสะสมผลงานเพื่อส่งผู้สอนตอนสิ้นสุดกิจกรรมการเรียนการสอนในภาคเรียนนี้