แผนการบริหารการสอนประจำบทที่ 7

เนื้อหาประจำบท

บทที่ 7 การจัดการหน่วยความจำ

1. กระบวนการในการจัดการหน่วยความจำ
2. ประเภทและวิธีการจัดการกับหน่วยความจำหลัก
3. ปัญหาของการจัดการหน่วยความจำและการวิธีการแก้ไข

จุดประสงค์เชิงพฤติกรรม

เมื่อศึกษาบทที่ 7 แล้วนักศึกษาสามารถ

1. สามารถอธิบายวิธีการที่แตกต่างกันของโครงสร้างทางฮาร์ดแวร์ของหน่วยความจำ
2. สามารถพิจารณาการใช้เทคนิคการจัดสรรและการจัดการหน่วยความจำหลักประเภทต่าง ๆ
3. เพื่อวิเคราะห์เปรียบเทียบข้อดีข้อเสียของวิธีการจัดการวิธีต่าง ๆ

กิจกรรมการเรียนการสอนประจำบท

1. ผู้สอนอธิบายหลักการทำงานของระบบปฏิบัติการ พร้อมยกตัวอย่างประกอบการบรรยาย
2. ให้ผู้เรียนศึกษาเอกสารประกอบการเรียนการสอน ศึกษาทำความเข้าใจและซักถาม
3. ให้ผู้เรียนทำแบบฝึกหัดและงานที่ได้รับมอบหมาย
4. ทดสอบย่อยหลังจบบทเรียน

สื่อการเรียนการสอน

1. สื่ออิเล็กทรอนิกส์ประกอบการสอนวิชาระบบปฏิบัติการ
2. เอกสารประกอบการสอนวิชาระบบปฏิบัติการ
3. หนังสืออ่านประกอบค้นคว้าเพิ่มเติม

การวัดผลและประเมินผล

1. สังเกตจากการซักถามในระหว่างการเรียน
2. สังเกตจากความสนใจและความตั้งใจ
3. ประเมินจากการอภิปรายกลุ่มย่อย และการทำแบบฝึกหัด
4. ประเมินจากการสอบระหว่างภาคและปลายภาค

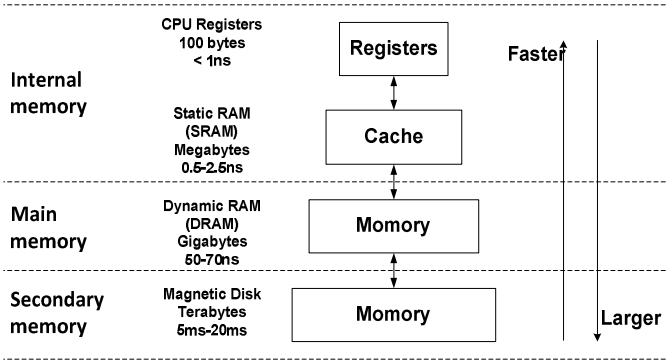
บทที่ 7
การจัดการหน่วยความจำ

การจัดการหน่วยความจำ เป็นงานอย่างหนึ่งของระบบปฏิบัติการ ถ้าหน่วยความจำมีปริมาณมาก ขีดความสามารถในการทำงานก็จะเพิ่มขึ้นด้วย โปรแกรมที่มีความสลับซับซ้อนและมีความสามารถมากก็ต้องการหน่วยความจำมาก ความต้องการหน่วยความจำของโปรแกรมและของโครงสร้างข้อมูลที่ต้องทำงานร่วมกับโพรเซสหรือระบบปฏิบัติการก็ยังเป็นที่ต้องการอยู่อีกมาก ดังนั้นจึงเป็นหน้าที่หลักของระบบปฏิบัติการที่จะต้องทำการจัดการหน่วยความจำที่มีอยู่ให้เกิดประโยชน์สูงสุด

หน่วยความจำจึงเป็นส่วนที่สำคัญที่สุดในระบบคอมพิวเตอร์ ถือเป็นศูนย์กลางการดำเนินการด้านต่าง ๆ ในระบบคอมพิวเตอร์ให้เป็นไปอย่างราบรื่นและมีประสิทธิภาพสูงสุด โดยภายในหน่วยความจำจะมีการทำงานหลายส่วน เช่น การทำงานของโปรแกรมจำนวนมาก ซึ่งต้องมีการแบ่งพื้นที่การใช้งานและวิธีการจัดการด้านต่าง ๆ และถ้าเราสามารถประหยัดเนื้อที่ของหน่วยความแม้นเพียงจำนวนไม่กี่พันไบต์อาจจะสร้างความแตกต่างระหว่างการทำงานของแต่ละโปรแกรม

7.1 ประเภทของหน่วยความจำ

หน่วยความจำ เป็นทรัพยากรสำคัญของระบบคอมพิวเตอร์ ทำหน้าที่เก็บข้อมูลต่าง ๆ รวมทั้งคำสั่ง และผลลัพธ์ที่ได้จากการทำงาน โดยลำดับชั้นของหน่วยความจำแบ่งออกเป็น 3 กลุ่ม คือ



ภาพที่ 7.1 จัดลำดับหน่วยความจำ

1) หน่วยความจำภายใน (Internal memory) หน่วยความจำภายใน หรือเรียกว่าหน่วยความจำแคช (Cache memory) ประกอบด้วยรีจีสเตอร์ความเร็วสูง โดยหน่วยความจำส่วนนี้ถูกใช้สำหรับเก็บคำสั่ง และข้อมูลที่ต้องการทำงานด้วยความเร็วสูงมาก และเป็นหน่วยความจำที่ซีพียูสามารถเข้าถึงได้โดยตรงและรวดเร็ว

2) หน่วยความจำหลัก (Main memory) หน่วยความจำหลัก เป็นหน่วยความจำความเร็วสูงใช้สำหรับเก็บคำสั่งและข้อมูลระหว่างการทำงาน โดยเป็นหน่วยความจำที่ซีพียูสามารถเข้าถึงได้โดยตรงและรวดเร็ว

3) หน่วยความจำสำรอง (Secondary memory) หน่วยความจำสำรองเป็นหน่วยความจำที่มีความเร็วน้อยกว่าหน่วยความจำสองประเภทแรก ใช้สำหรับเก็บข้อมูลที่มีขนาดใหญ่ และเป็นข้อมูลที่ยังไม่ต้องการนำมาประมวลผล

7.2 แนวคิดพื้นฐานการจัดการหน่วยความจำหลัก

หน้าที่ของระบบปฏิบัติการในการจัดการกับหน่วยความจำหลักมี 4 ประการคือ

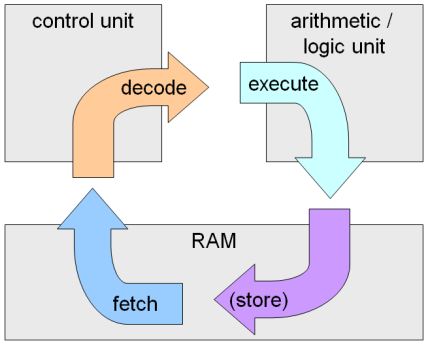
1) ควบคุมดูแลสถานะของแต่ละตำแหน่งของหน่วยความจำหลัก

2) ตัดสินว่าควรจัดสรรหน่วยความจำหลักขนาดเท่าไร ให้กับงานใด ณ ตำแหน่งใดของหน่วยความจำหลัก

3) จัดสรรหน่วยความจำหลักให้งานที่ได้เลือกแล้ว

4) ปลดปล่อยหน่วยความจำหลักให้ว่าง เมื่อทำงานเสร็จแล้ว

คำสั่งที่จะถูกดำเนินการได้โดยซีพียู จะต้องถูกดึงมาและเก็บที่ตำแหน่งในหน่วยความจำ รูปแบบการทำงานของรอบคำสั่งเครื่องและการปฏิบัติงานตามคำสั่ง (Instruction-execution cycle) จะมีขั้นตอนการทำงาน ดังนี้



ภาพที่ 7.2 แสดงไดอะแกรมการทำงานของรอบคำสั่งเครื่องและการปฏิบัติงานตามคำสั่ง
ที่มา: Personal blog of Shane Preece, Occasional tech minddump. Retrieved June 25, 2014 from http://blog.shamess.info/2008/12/03/computer-tech-lecture-three/

1) ขั้นตอนที่ 1 ไปนำมา (Fetch) คือ การเริ่มต้นการทำงานซึ่งระบบจะทำการดึงคำสั่งแรกจากหน่วยความจำ

2) ขั้นตอนที่ 2 ถอดรหัส (Decode) คือ การทำงานต่อจากขั้นตอนที่ 1 โดยนำคำสั่งนี้ไปทำการถอดรหัส ซึ่งอาจจะได้ตัวดำเนินการหรือข้อมูล เพื่อใช้กับคำสั่งถัดไป

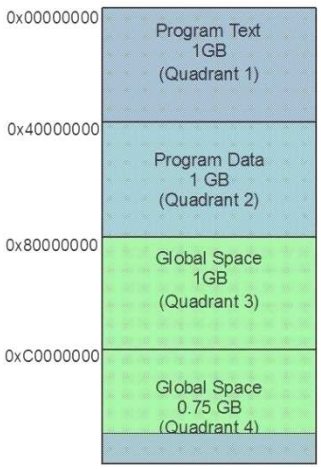
3) ขั้นตอนที่ 3 กระทำการ (Execution) คือ การทำงานต่อจากขั้นตอนที่ 2 ซึ่งหลังจากนั้นคำสั่งจะทำงานตามตัวดำเนินการที่ได้

4) ขั้นตอนที่ 4 จัดเก็บ (Store) ผลลัพธ์จะถูกเก็บกลับไปในหน่วยความจำหลักต่อไป

7.3 หน่วยความจำหลัก

หน่วยความจำหลัก เป็นศูนย์กลางของการทำงานต่าง ๆ ของระบบคอมพิวเตอร์ในปัจจุบัน ประกอบไปด้วยอาร์เรย์ขนาดใหญ่ (Large array) ซึ่งภายในประกอบไปด้วยเวิร์ด (Words) และไบต์ (Bytes) โดยแต่ละไบต์จะมีแอดเดรส (Address) บอกตำแหน่งของตัวเอง นอกจากนี้หน่วยความจำหลักยังทำหน้าที่เก็บชนิดกระบวนการในการประมวลผลคำสั่ง (A Typical Instruction-Execution Cycle) เพื่อให้หน่วยประมวลผลกลาง (Central Processing Unit : CPU) นำไปใช้ในการประมวลแล้วจึงทำการส่งผลลัพธ์ของคำสั่งนั้น ๆ กลับมาจัดเก็บไว้ในหน่วยความจำหลักอีกที

และโดยทั่วไปแล้วนักเขียนโปรแกรมคอมพิวเตอร์ (Programmer) ล้วนแต่ต้องการระบบที่มีหน่วยความจำหลักแบบไม่จำกัด มีความเร็วสูง และมีหน่วยความจำที่มีความเสถียรสูงหรือเป็นแบบไม่ถูกลบเลือน ซึ่งหมายความว่าแม้ระบบไฟฟ้าจะขัดข้องข้อมูลต่าง ๆ ในหน่วยความจำหลักไม่สูญหายไปด้วย



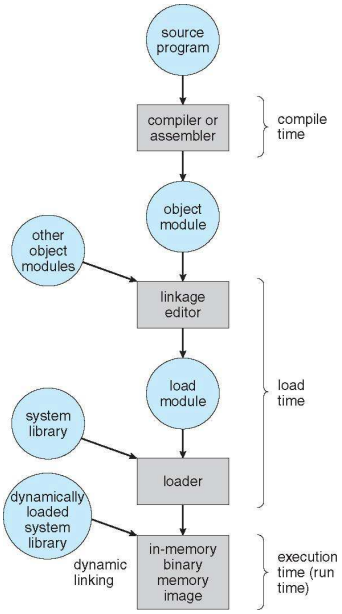
ภาพที่ 7.3 แสดง physical address ของหน่วยความจำหลัก
ที่มา: IBM, Developer Works. Retrieved June 25, 2014 from
http://www.ibm.com/developerworks/data/library/techarticle/dm-0406qi/

7.3.1 การเชื่อมโยงตำแหน่ง (Address Binding)

การเขียนโปรแกรมระยะแรก มีการกำหนดตำแหน่งของหน่วยความจำหลักด้วยเลขที่อยู่สัมบูรณ์ (Absolute address) คือ ตำแหน่งที่แน่นอนและมีอยู่จริงในหน่วยความจำหลัก มีการระบุตำแหน่งของหน่วยความจำที่โปรแกรมจะต้องถูกโหลดเพื่อนำไปใช้งาน ที่ตัวโปรแกรมหรือตัวแปลภาษาจะต้องทราบตำแหน่งที่แน่นอน ซึ่งจะทำให้เกิดปัญหาในกรณีที่ตำแหน่งที่ระบุไว้ไม่ว่าง จะทำให้ไม่สามารถใช้งานโปรแกรมได้ ทำให้ขาดความยืดหยุ่นในการใช้งาน และขาดคุณสมบัติการย้ายที่อยู่ (Relocate)

จึงได้มีการพัฒนาการเขียนโปรแกรมให้มีการใช้เลขที่อยู่เชิงสัญลักษณ์ ด้วยการกำหนดตัวระบุ เช่น ชื่อตัวแปร หรือซับรูทีน และมีการอ้างถึงหน่วยความจำหลักด้วยเลขที่อยู่สัมพัทธ์ (Relative

address) ซึ่งช่วยแก้ปัญหาของการเขียนโปรแกรม โดยโปรแกรมสามารถโหลดลงตำแหน่งใดก็ได้ในหน่วยความจำหลัก



ภาพที่ 7.4 ขั้นตอนต่าง ๆ ในการเรียกใช้งานของโปรแกรม
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.355)

การจำแนกการเชื่อมโยงของคำสั่งและตำแหน่งของข้อมูล การกำหนดเลขที่อยู่ของโปรแกรมเพื่อแปลงไปเป็นเลขที่อยู่ในหน่วยความจำหลัก สามารถทำได้ในช่วงเวลาต่าง ๆ ของการทำงานของโปรแกรม มีขั้นตอนดังนี้

1) เวลาแปลโปรแกรม (Compile time) เมื่อนำโปรแกรมต้นฉบับที่มีการระบุตำแหน่งของหน่วยความจำหลักด้วยเลขที่อยู่สัมบูรณ์มาผ่านการแปลด้วยคอมไพเลอร์ จะได้ออบเจ็กโปรแกรมที่มีการอ้างถึงตำแหน่งของหน่วยความจำหลักด้วยเลขที่อยู่สัมบูรณ์ เมื่อต้องการให้โปรแกรมส่วนนี้ทำงาน ระบบสามารถเรียกโหลดเดอร์สัมบูรณ์ (Absolute loader) เพื่อทำหน้าที่โหลดโปรแกรมขึ้นมาทำงานเพียงอย่างเดียว

2) เวลาโหลดโปรแกรม (Load time) เมื่อนำโปรแกรมต้นฉบับที่มีการระบุตำแหน่งของหน่วยความจำหลักด้วยเลขที่อยู่สัมพัทธ์มาผ่านการแปลด้วยคอมไพเลอร์ จะได้ออบเจ็กของโปรแกรมที่มีการอ้างถึงตำแหน่งของหน่วยความจำหลักด้วยเลขที่อยู่สัมพัทธ์ เป็นหน้าที่ของโหลดเดอร์ ซึ่งเป็นซอฟต์แวร์ระบบที่จะนำโปรแกรมที่ต้องการเข้ามาบรรจุบนหน่วยความจำหลัก ดังนั้นทุกครั้งที่ทำการโหลดโปรแกรมมาใช้งาน โหลดเดอร์จะต้องหาตำแหน่งของหน่วยความจำหลักที่จะโหลดโปรแกรมเข้ามา โดยโหลดเดอร์ต้องติดต่อกับระบบปฏิบัติการ เพื่อให้ระบบปฏิบัติการหาพื้นที่ว่างของหน่วยความจำหลักที่มีขนาดมากพอที่จะบรรจุโปรแกรมได้ หากมีการเรียกใช้โปรแกรมหลายครั้ง โหลดเดอร์จำเป็นต้องหาตำแหน่งของหน่วยความจำหลักทุกครั้ง โดยไม่จำเป็นต้องเป็นตำแหน่งเดิม ทำให้โปรแกรมมีความยืดหยุ่นในการใช้งาน และมีคุณสมบัติของการย้ายที่อยู่ แต่ระบบจำเป็นต้องมีการกำหนดตำแหน่ง เพื่อเป็นการ

เชื่อมโยงระหว่างตัวแปรกับตำแหน่งที่แท้จริงของหน่วยความจำหลัก ดังนั้นการกำหนดตำแหน่งจะถูกกระทำในช่วงเวลาโหลด คือ ช่วงเวลาที่ทำการโหลดโปรแกรม โดยใช้โหลดเดอร์ย้ายที่อยู่ ทำหน้าที่สำคัญ คือ ทำการแปลจากเลขที่อยู่เชิงสัมพัทธ์ให้เป็นเลขที่อยู่เชิงสัมบูรณ์

3) เวลากระทำการ (Execution time) โปรแกรมต้นฉบับถูกแบ่งออกเป็นส่วน ๆ โดยแต่ละส่วนจะถูกใช้ไม่พร้อมกัน เช่น ส่วนที่รับข้อมูล ส่วนประมวลผลข้อมูล และส่วนแสดงผลลัพธ์ เมื่อนำโปรแกรมผ่านตัวคอมไพเลอร์ จะได้ออบเจ็กโปรแกรมที่อ้างถึงตำแหน่งที่อยู่ในหน่วยความจำหลักด้วยเลขที่อยู่เชิงสัมพัทธ์ และเก็บโปรแกรมไว้ในหน่วยความจำสำรอง เมื่อต้องการใช้งานจะมีโหลดเดอร์แบบไดนามิก ที่ทำการโหลดเฉพาะส่วนของโปรแกรมที่ต้องการใช้งานเท่านั้นเข้ามาในหน่วยความจำหลัก ดังนั้นการกำหนดตำแหน่งจะเกิดขึ้นในช่วงเวลากระทำการจองแอดเดรสของหน่วยความจำภายในโปรแกรมของผู้ใช้อาจแสดงได้หลายแบบแตกต่างกัน ตามขั้นตอนต่าง ๆ ในโปรแกรมต้นฉบับ

ตัวอย่าง ในโปรแกรมมีตัวแปร a และคอมไพเลอร์ แปลตัวแปรนี้เป็นแอดเดรสแบบย้ายได้ เช่น ให้ตัวแปรนี้อยู่ในแอดเดรสระยะห่าง 16 ไบต์จากจุดเริ่มต้นของโปรแกรม เมื่อโปรแกรมถูกโหลดเข้าไปในหน่วยความจำตำแหน่งที่ 84000 ตัวโหลดเดอร์ก็จะทำหน้าที่แปลงค่าแอดเดรสแบบย้ายได้ของตัวแปรนั้นไปเป็นแอดเดรสจริงทางกายภาพ ซึ่งคือตำแหน่ง 84016 นั่นเอง ดังนั้นค่าของที่อยู่นั้นจะแบ่งออกเป็น 2 ค่าคือ

1) Absolute address แอดเดรสแท้จริงของโพรเซสที่อยู่ในพาร์ติชั่นของหน่วยความจำ
2) Relative address แอดเดรสของคำสั่งหรือโปรแกรมของโพรเซสจากการคอมไพล์

7.3.2 Dynamic Loading

Routine จะไม่ถูกโหลด จนกระทั่งถูกเรียกใช้งาน เป็นการใช้หน่วยความจำเป็นประโยชน์มากกว่า โปรแกรมย่อยที่ไม่ได้ใช้ (Unused routine) จะไม่ถูกโหลดมาในหน่วยความจำหลักซึ่งมีประโยชน์สำหรับโปรแกรมขนาดใหญ่ ที่มีบางส่วนหรือบางกรณีที่เกิดขึ้นไม่บ่อยและไม่ต้องการการสนับสนุนจากระบบปฏิบัติการ การทำ Dynamic loading อาศัยการ Implement โดยการออกแบบโปรแกรมเอง

7.3.3 Dynamic Linking and Shared Libraries

Linking จะเลื่อนออกไปจนกระทั่งอยู่ในช่วง Execution time มีชุดคำสั่งเล็ก ๆ เรียกว่า Stub ใช้สำหรับ Locate ไลบรารีของโปรแกรมย่อยที่เหมาะสมในหน่วยความจำหลัก Stub จะแทนที่ตัวเองด้วยแอดเดรสของโปรแกรมย่อยและจะทำงานตามโปรแกรมย่อยนั้น ซึ่งระบบปฏิบัติการจำเป็นต้องตรวจดูว่าโปรแกรมย่อยนั้นต้องอยู่ภายในแอดเดรสของโพรเซสในหน่วยความจำ Dynamic linking มีประโยชน์โดยเฉพาะอย่างยิ่งสำหรับการใช้ระบบไลบรารีในการแบ่งปันไลบรารี

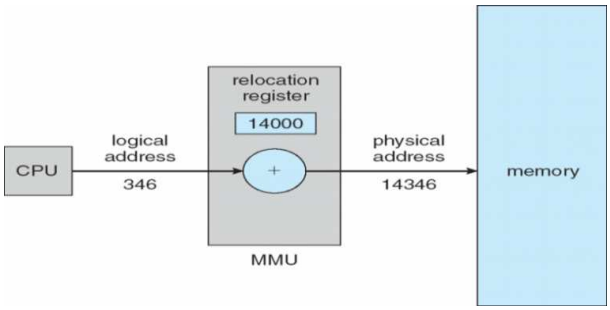
7.4 ตำแหน่งที่ว่างทางกายภาพกับตำแหน่งที่ว่างทางตรรกะ

ตำแหน่งที่ถูกสร้างโดยซีพียูมักหมายถึงตำแหน่งทางตรรกะ (Logical address) ส่วนตำแหน่งในหน่วยความจำจะหมายถึง Physical address เป็นตำแหน่งบนหน่วยความจำคือ ตำแหน่งทางกายภาพ

Logical และ Physical address เป็นตำแหน่งเดียวกันในช่วงของ Compile time และ Load time การเชื่อมโยง Logical address เข้ากับแต่ละ Physical address ถือเป็นหัวใจสำคัญในการจัดการหน่วยความจำหลัก

แต่ช่วง Execution time จะมีตำแหน่งทางตรรกะและทางกายภาพต่างกัน ด้วยเหตุนี้เราจึงมักอ้างถึงตำแหน่งทางตรรกะว่า ตำแหน่งเสมือน (Virtual address) กลุ่มของตำแหน่งทางตรรกะทั้งหมดที่ถูกสร้างโดยโปรแกรมจะถูกเรียกว่า ตำแหน่งที่ว่างทางตรรกะ (Logical address space) คือ ตำแหน่งเสมือนที่อ้างอิงโดยโปรแกรม ส่วนกลุ่มของตำแหน่งที่ว่างทางกายภาพที่เกี่ยวข้องกับตำแหน่งทางตรรกะเหล่านั้นถูกเรียกว่า ตำแหน่งที่ว่างทางกายภาพ (Physical address space) ดังนั้นช่วง Execution time ตำแหน่งที่ว่างทางตรรกะและทางกายภาพจึงต่างกัน

เมื่อบรรจุกระบวนการเข้ามาในหน่วยความจำ Logical address จะต้องถูกแปลงไปเป็น Physical address เรียกวิธีการนี้ว่า การย้ายเลขที่อยู่ (Relocation) ดังนั้นการจับคู่ (Mapping) ของตำแหน่งเสมือนกับตำแหน่งจริงทางกายภาพจะถูกทำโดยหน่วยจัดการหน่วยความจำ (Memory Management Unit : MMU) คือ ส่วนของฮาร์ดแวร์ที่รับผิดชอบในการจัดการหน่วยความจำแทนซีพียู โดยหน้าที่หลักของ MMU จะรับ Virtual address จากซีพียู และแปลงเป็น Physical address เพื่ออ้างอิงตำแหน่งจริงจากหน่วยความจำ ทำการป้องกันข้อมูล (Memory protection) ไม่ให้โปรแกรมไม่หวังดีเข้าถึงได้ และทำการจัดการหน่วยความจำแคช (Cache control) สำหรับดูแลการจัดส่งข้อมูล (Bus arbitration) อย่างถูกต้องครบถ้วนและปลอดภัย



ภาพที่ 7.5 ขั้นตอนต่าง ๆ ในการเรียกใช้งานของโปรแกรม
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.356)

จากภาพที่ 7.5 รีจีสเตอร์ฐานกลายเป็นรีจีสเตอร์สำหรับการย้ายตำแหน่ง (Relocation register) การอ้างอิงตำแหน่งในหน่วยความจำทุกครั้งต้องนำค่าอ้างอิงมาบวกกับค่ารีจีสเตอร์ฐานเสียก่อนเพื่อให้ได้ค่าตำแหน่งจริง เช่น ค่ารีจีสเตอร์ฐานเท่ากับ 14000 ผู้ใช้ต้องการอ่านค่าจากตำแหน่ง 346 ก็จะไปอ่านจากตำแหน่งจริงที่ 14000 + 346 = 14346 เป็นต้น

จะสังเกตได้ว่า โปรแกรมของผู้ใช้ไม่สามารถทราบค่าตำแหน่งที่แท้จริง (ทางกายภาพ) โปรแกรมอาจสร้างตัวชี้ไปยังตำแหน่ง 346 และทำการคำนวณ เก็บค่า อ่านค่า หรือเปรียบเทียบค่าในตำแหน่งอื่น ๆ กับค่าในตำแหน่ง 346 นี้ โปรแกรมของผู้ใช้ทำงานด้วยตำแหน่งทางตรรกะ ฮาร์ดแวร์ของเครื่องเป็นผู้จัดการจับคู่ตำแหน่งทางตรรกะนี้กับตำแหน่งจริง เมื่อมีการอ้างอิงตำแหน่งในหน่วยความจำ

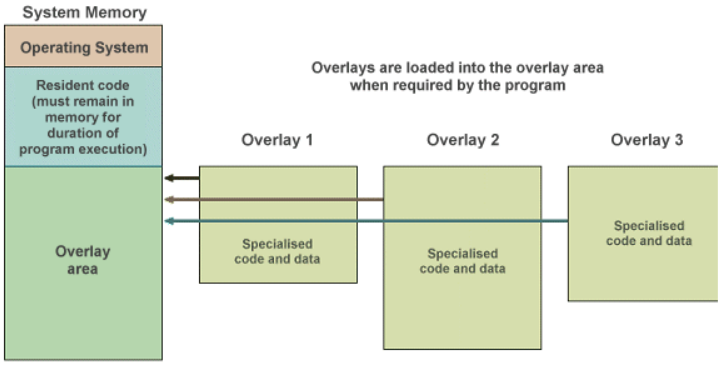
แนวคิดในเรื่องการใช้ตำแหน่งทางตรรกะเพื่อใช้ในการอ้างอิงแทนตำแหน่งจริงนี้เป็นหัวใจของการจัดการหน่วยความจำหลัก

7.5 การจัดการหน่วยความจำหลัก

หน้าที่ของระบบปฏิบัติการในการจัดการกับหน่วยความจำหลัก คือ การควบคุมดูแลสถานะของแต่ละตำแหน่งของหน่วยความจำหลัก พร้อมตัดสินว่าควรจัดสรรหน่วยความจำหลักขนาดเท่าไรให้กับโพรเซสใด ณ ตำแหน่งใดของหน่วยความจำหลัก และจัดสรรหน่วยความจำหลักให้โพรเซสที่ได้เลือกแล้ว ปลดปล่อยหน่วยความจำหลักให้ว่างเมื่อทำงานเสร็จแล้ว

7.5.1 วิธีการซ้อนทับ (Overlays)

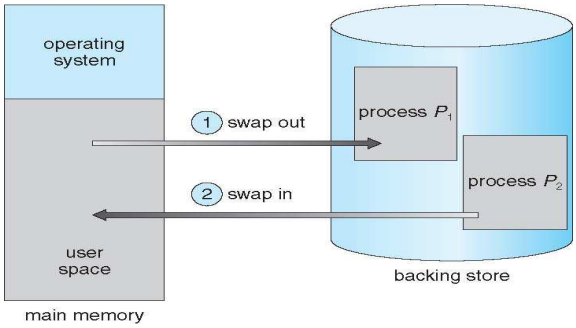
โดยทั่วไปโพรเซสที่ต้องการประมวลผลด้วยซีพียู จะต้องถูกบรรจุอยู่ในหน่วยความจำหลัก ซึ่งโพรเซสจะต้องมีขนาดน้อยกว่าหรือเท่ากับขนาดของหน่วยความจำหลักลบด้วยขนาดของระบบปฏิบัติการ ดังนั้นหากมีความจำเป็นที่ต้องใช้โพรเซสที่มีขนาดมากกว่าขนาดของหน่วยความจำหลัก สามารถทำได้โดยวิธีซ้อนทับ (Overlays) โพรเซสที่จะทำงานด้วยวิธีซ้อนทับ จะต้องถูกแบ่งออกเป็นส่วนย่อยที่อิสระต่อกัน คือแต่ละส่วนไม่จำเป็นต้องถูกเรียกใช้งานพร้อมกัน โดยระบบจะนำเฉพาะส่วนของโพรเซสที่ต้องการใช้เข้ามาบรรจุในหน่วยความจำหลัก ส่วนอื่น ๆ ของโพรเซสให้เก็บไว้ในหน่วยความจำสำรอง



ภาพที่ 7.6 แสดงวิธีซ้อนทับ
ที่มา: TechnologyUK. Retrieved June 26, 2014 from http://www.technologyuk.net/computing/operating_systems/memory_management.shtml

7.5.2 วิธีการสับเปลี่ยน (Swapping)

เนื่องจากหน่วยความจำหลักมีขนาดพื้นที่จำกัด บางครั้งระบบมีพื้นที่หน่วยความจำหลัก ไม่เพียงพอกับการใช้งาน ทำให้ระบบจำเป็นต้องนำบางโพรเซสออกจากหน่วยความจำหลักก่อน เพื่อให้หน่วยความจำหลักมีพื้นที่ว่างมากเพียงพอที่จะบรรจุโพรเซสอื่น เป็นการสับเปลี่ยน (Swap) ให้โพรเซสอื่นทำงาน



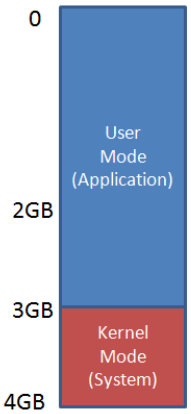
ภาพที่ 7.7 วิธีการสับเปลี่ยนการทำงานของโพรเซส
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.358)

ตัวอย่าง ระบบที่มีการจัดลำดับการทำงานของโพรเซส โดยใช้ลำดับความสำคัญที่กำหนดให้โพรเซสที่มีลำดับความสำคัญสูง สามารถทำงานก่อนโพรเซสที่มีลำดับความสำคัญต่ำกว่า

ดังนั้นถ้าโพรเซส P1 กำลังทำงาน (อยู่ในหน่วยความจำหลัก) แต่มีโพรเซส P2 ซึ่งมีลำดับความสำคัญสูงกว่าต้องการทำงาน ระบบปฏิบัติการต้องนำโพรเซส P1 ออกจากหน่วยความจำหลักไปเก็บไว้ในหน่วยความจำสำรอง เรียกว่า สับเปลี่ยนออก (Swap out) และนำเอาโพรเซส P2 จากหน่วยความจำสำรองเข้ามาบรรจุที่หน่วยความจำหลัก เรียกว่า สับเปลี่ยนเข้า (Swap in)

7.6 การจัดสรรหน่วยความจำแบบต่อเนื่อง

ทุกโปรแกรมมีความจำเป็นต้องใช้หน่วยความจำที่มีอยู่ในระบบ จะใช้มากหรือน้อยขึ้นอยู่กับการทำงานและขนาดของโปรแกรม โปรแกรมจะทำงานได้ก็ต่อเมื่อมันได้ถูกนำไปวางไว้ในหน่วยความจำแล้วเท่านั้น การที่โปรแกรมได้เข้าไปใช้หน่วยความจำของระบบเป็นเพราะการจัดสรรหน่วยความจำ (Memory allocation) ของระบบปฏิบัติการ



ภาพที่ 7.8 วิธีการอ้างอิงตำแหน่งจริงจากหน่วยความจำ
ที่มา: IBM, Developer Works. Retrieved June 25, 2014 from
http://www.ibm.com/developerworks/data/library/techarticle/dm-0406qi/

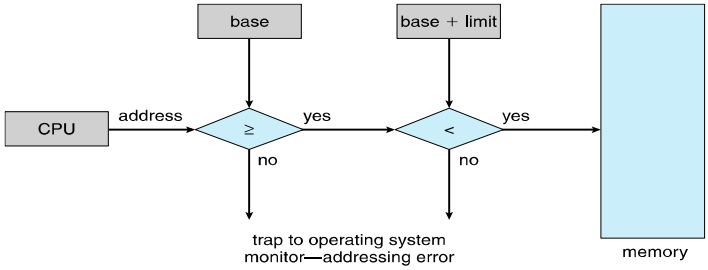
หน่วยความจำหลักจำเป็นจะต้องอำนวยความสะดวกให้กับระบบปฏิบัติการและความหลากหลายของผู้ใช้งานโพรเซส ดังนั้นในแต่ละโพรเซสจึงมีความต้องการใช้พื้นที่ในหน่วยความจำอย่างต่อเนื่อง จึงเป็นหน้าที่ของระบบปฏิบัติการในการจัดสรรพื้นที่หน่วยความจำในหน่วยความจำให้มีประสิทธิภาพสูงสุด ซึ่งโดยทั่วไปแล้วหน่วยความจำหลักจะถูกแบ่งออกเป็น 2 ส่วน คือ

1) หน่วยความจำระดับบน (High memory) ซึ่งเป็นส่วนที่ใช้ติดต่อกับส่วนของผู้ใช้

2) หน่วยความจำระดับล่าง (Low memory) ซึ่งเป็นส่วนของระบบปฏิบัติการ

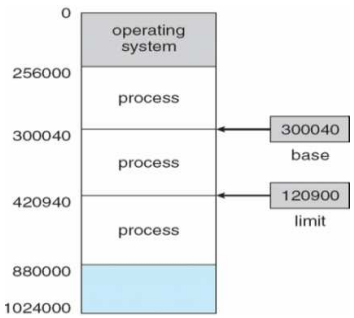
7.6.1 การจัดสรรพื้นที่แบบขนาดคงที่ (Fixed-Size Partition)

การจัดสรรพื้นที่แบบขนาดคงที่หรือแบบส่วนเดียว (Single-partition allocation) ชุดของรีจิสเตอร์ย้ายตำแหน่ง (Relocation-register scheme) จะใช้สำหรับป้องกันการทำงานโพรเซสของผู้ใช้จากโพรเซสอื่น และจากการเปลี่ยนรหัสของระบบปฏิบัติการ และข้อมูลชุดของรีจิสเตอร์ย้ายตำแหน่งบรรจุด้วยค่าเลขที่อยู่เชิงกายภาพที่เล็กที่สุด, Base register บางครั้งเรียก Offset บรรจุขอบเขตของที่อยู่เชิงตรรกะ ซึ่งแต่ละเลขที่อยู่จะต้องมีค่าน้อยกว่า Limit register



ภาพที่ 7.9 วิธีการอ้างอิงตำแหน่งจริงจากหน่วยความจำ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.353)

จากภาพที่ 7.9 ถ้า Base register จองพื้นที่ที่ตำแหน่ง 300040 และ Limit register กำหนดขอบเขตที่ตำแหน่ง 120900 ดังนั้นโปรแกรมสามารถเข้าถึงตำแหน่ง Address ได้ทั้งหมดตั้งแต่ตำแหน่ง 300040 จนถึงตำแหน่ง 420939



ภาพที่ 7.10 วิธีการอ้างตำแหน่ง address ในหน่วยความจำ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.353)

การจัดการหน่วยความจำหลักสำหรับงานเดียววิธีนี้เป็นวิธีที่ง่ายที่สุดเพราะกำหนดเพียง 1 โปรแกรมทำงาน ณ เวลาหนึ่ง ดังนั้นการใช้งานหน่วยความจำจะมีเพียง 1 โปรแกรมเท่านั้น คอมพิวเตอร์ในยุคแรก จะสามารถทำงานได้เพียงครั้งละ 1 งาน การจัดการหน่วยความจำหลักสามารถทำได้ง่ายดังแสดงจากภาพที่ 7.8 และ ภาพที่ 7.10 จะเห็นว่า หากโปรแกรมที่บรรจุในหน่วยความจำหลักมีขนาดเล็กจะทำให้มีพื้นที่ของหน่วยความจำหลักถูกทิ้งไปจำนวนมาก (เหลือว่างจำนวนมาก) แต่ถ้าโปรแกรมมีขนาดใหญ่เกินไป ระบบจะไม่สามารถบรรจุโปรแกรมนั้นในหน่วยความจำหลักได้ ดังนั้นการจัดการหน่วยความจำหลักวิธีนี้ โปรแกรมต้องมีขนาดน้อยกว่าหรือเท่ากับขนาดของหน่วยความจำหลัก

วิธีนี้หน่วยความจำจะไม่ถูกแบ่งพื้นที่ โดยโพรเซสในส่วนของระบบปฏิบัติการจะอยู่ในส่วนหน่วยความจำระดับล่าง และส่วนของโพรเซสของผู้ใช้อยู่ในส่วนหน่วยความจำระดับบน โดยเกี่ยวข้องกับรีจิสเตอร์ย้ายตำแหน่ง (Relocation register) ในการแยกโพรเซสในส่วนของผู้ใช้ออกจากส่วนของระบบปฏิบัติการ และรีจิสเตอร์ขอบเขตเป็นรีจิสเตอร์ที่ใช้ระบุขนาดของค่าตำแหน่งทางตรรกะ โดยค่าตำแหน่งทางตรรกะต้องน้อยกว่ารีจิสเตอร์ขอบเขตเสมอ ทุกครั้งที่โปรแกรมมีการอ้างถึงตำแหน่งใด ๆ ของหน่วยความจำหลัก ตำแหน่งนั้นจะต้องถูกตรวจสอบกับค่าที่บรรจุในรีจีสเตอร์ เพื่อป้องกันการเข้าไปยุ่งเกี่ยวกับเนื้อที่ของหน่วยความจำหลักที่บรรจุระบบปฏิบัติการ ถ้ามีโปรแกรมใดพยายามเข้าถึงตำแหน่งของหน่วยความจำหลักที่ถูกป้องกัน จะมีผลให้เกิดการขัดจังหวะเพื่อให้ระบบปฏิบัติการเข้ามาจัดการต่อไป

ข้อดีของการจัดการหน่วยความจำแบบนี้ คือ ความง่ายในการจัดสรรหน่วยความจำหลัก ทำให้ลดความซ้ำซ้อนของระบบปฏิบัติการลงและง่ายในการเขียนโปรแกรม ข้อเสียของการจัดการหน่วยความจำแบบนี้ คือ หน่วยความจำหลักไม่ได้ถูกใช้งานอย่างเต็มที่ เพราะแม้จะมีพื้นที่ว่างบางส่วนที่เหลือจากการใช้งานก็ต้องทิ้งไป ไม่สามารถนำไปจัดสรรให้กับงานอื่น ๆ ได้ นอกจากนี้คอมพิวเตอร์จะต้องทำงานเพียงครั้งละ 1 งาน ทำให้มีบางช่วงเวลาที่ซีพียูว่าง เพราะต้องรออุปกรณ์รับเข้า/ส่งออกทำงานรับส่งข้อมูล

7.6.2 การแบ่งหน่วยความจำออกเป็นพาร์ติชั่น

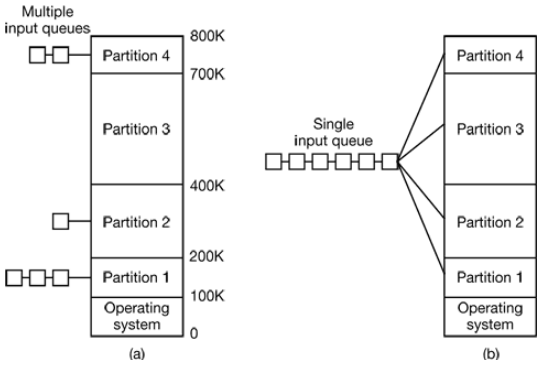
ข้อเสียของการจัดการหน่วยความจำหลักแบบงานเดียว คือ คอมพิวเตอร์สามารถทำงานได้ครั้งละ 1 งาน เพื่อเป็นการเพิ่มประสิทธิภาพการทำงานของซีพียู จึงพัฒนาให้ระบบคอมพิวเตอร์สามารถทำได้หลายงานพร้อมกัน เรียกว่า การทำงานแบบมัลติโปรแกรมมิ่ง ระบบคอมพิวเตอร์ส่วนใหญ่ในปัจจุบันจะอนุญาตให้มีหลายโพรเซสทำงานในเวลาเดียวกันได้ การที่โพรเซสหลายตัวทำงานพร้อมกันอธิบายได้ว่าขณะที่โพรเซสหนึ่งกำลังรอการนำเข้า/ส่งออกข้อมูลทำงานเสร็จนั้น โพรเซสอื่นสามารถใช้งานโพรเซสเซอร์ได้ จะสลับไปทำงานอื่นได้ทันที โดยไม่ต้องเสียเวลารอให้มีการเรียกงานอื่นจากหน่วยความจำสำรองเข้าไปในหน่วยความจำหลัก ดังนั้นการทำงานในลักษณะนี้จะเป็นการเพิ่มประสิทธิภาพการใช้งานของซีพียูและเป็นลักษณะการทำงานของเครื่องเซิร์ฟเวอร์ และลูกเครือข่ายในปัจจุบันอีกด้วย

โดยวิธีพื้นฐาน คือ การแบ่งหน่วยความจำหลักออกเป็นส่วนย่อย ๆ หรือแบ่งออกเป็นพาร์ติชั่น (Partition) แต่ละพาร์ติชั่นถูกใช้สำหรับงาน 1 งาน จำนวนงานที่สามารถทำงานได้พร้อมกันจะถูกกำหนดด้วยจำนวนพาร์ติชั่น การแบ่งหน่วยความจำออกเป็นพาร์ติชั่นทำได้ 2 วิธี คือ

7.6.2.1 การกำหนดขนาดพาร์ติชั่นคงที่ (Static Partition)

วิธีนี้ก่อนจะมีการทำงานใด ๆ หน่วยความจำหลักต้องถูกจัดสรรแบ่งออกเป็นส่วน ๆ ที่มีขนาดแน่นอน ซึ่งการแบ่งส่วนอาจกำหนดโดยโอเปอเรเตอร์ของระบบคอมพิวเตอร์ หรือกำหนดโดย

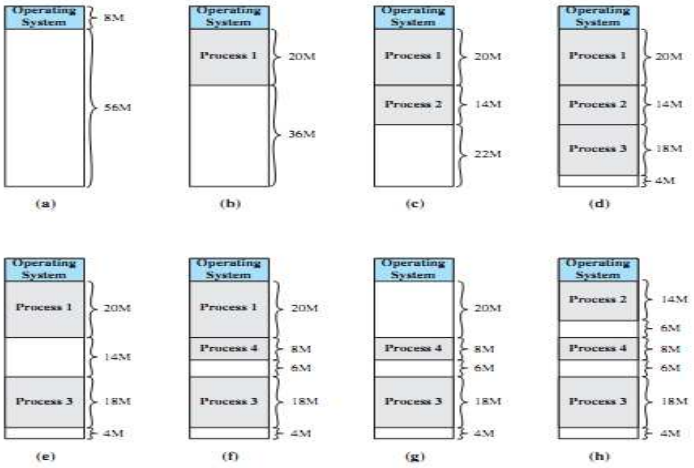
ระบบปฏิบัติการ ถ้ามีโพรเซสที่ต้องการทำงานและขอใช้พื้นที่ของหน่วยความจำหลัก ระบบปฏิบัติการจะต้องตรวจสอบดูขนาดของโพรเซสว่า ควรจะบรรจุลงพาร์ติชั่นใด ซึ่งเป็นไปได้ที่ขนาดของโพรเซสจะไม่พอดีกับขนาดของพาร์ติชั่น กรณีนี้มีผลให้มีพื้นที่บางส่วนของหน่วยความจำหลักที่ต้องสูญเสียไปโดยเปล่าประโยชน์ วิธีนี้เหมาะสำหรับกรณีที่ระบบรู้ขนาดของโพรเซสล่วงหน้า ทั้งนี้เพื่อระบบจะได้กำหนดขนาดของแต่ละส่วนของหน่วยความจำหลักที่เหมาะสมได้



ภาพที่ 7.11 การกำหนดขนาดพาร์ติชั่นคงที่ (a) แบบ Multiple in queues (b) Single input queue
ที่มา: Prime Interactive Ltd., operator of Host.sk. Retrieved June 27, 2014 from http://lovingod.host.sk/tanenbaum/BASIC-MEMORY-MANAGEMENT.html

7.6.2.2 การกำหนดขนาดของพาร์ติชั่นให้เปลี่ยนแปลงได้ (Dynamic Partition)

วิธีนี้เป็นการแบ่งส่วนของหน่วยความจำหลักแบบไดนามิก (Dynamic) ที่มีการจัดสรรหน่วยความจำหลักให้โพรเซสต่าง ๆ ตามขนาดที่ผู้ใช้ร้องขอเมื่อมีการขอใช้หน่วยความจำหลัก ระบบปฏิบัติการจะตรวจสอบขนาดของโพรเซส และจัดสรรหน่วยความจำหลักให้แก่โพรเซสเป็นขนาดเท่ากันกับขนาดของโพรเซสนั้น ๆ ซึ่งอาจจะมีขนาดพาร์ติชั่นที่เหมาะสม หรืออาจต้องแบ่งพาร์ติชั่นใหม่เมื่อโพรเซสทำงานเสร็จเรียบร้อย ระบบปฏิบัติการจะเรียกเอาหน่วยความจำทั้งหมดคืนจากโพรเซส



ภาพที่ 7.12 วิธีการสับเปลี่ยนการทำงานของโพรเซส
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.253)

ตัวอย่าง ถ้าเรามีหน่วยความจำหลักทั้งหมด 64 MB ดังภาพที่ 7.12 เริ่มในภาพที่ 7.12 (a) มีบางส่วนของหน่วยความจำหลักที่ใช้งานสำหรับระบบปฏิบัติการ นอกนั้นหน่วยความจำหลักยังคงว่างอยู่ เมื่อโพรเซส 3 งานโหลดเข้ามาใช้งาน ระบบจะจัดสรรหน่วยความจำเริ่มต้นจากตำแหน่งสิ้นสุดของหน่วยความจำหลักที่ระบบปฏิบัติการใช้งานอยู่ โดยจัดสรรพื้นที่ให้พอดีกับโพรเซสแต่ละตัวต้องการ ดังภาพที่ 7.12 (b) (c) และ (d)

ขณะที่โพรเซสทั้ง 3 กำลังทำงาน โพรเซสที่ 4 ที่มีขนาด 8 M โหลดเข้ามาจะสังเกตได้ว่าภาพที่ 7.12 (d) นั้นหน่วยความจำจะเหลือพื้นที่ว่าง แต่มีจำนวนขนาดน้อยกว่าพื้นที่ที่โพรเซสจะทำงานได้ ในภาพที่ 7.12 (e) ระบบปฏิบัติการทำการดึงโพรเซสที่ 2 ออกจากหน่วยความจำหลัก จึงทำให้มีพื้นที่เหลือเพียงพอที่จะทำให้โพรเซสที่ 4 เข้าไปใช้งานได้ ดังภาพที่ 7.12 (f)

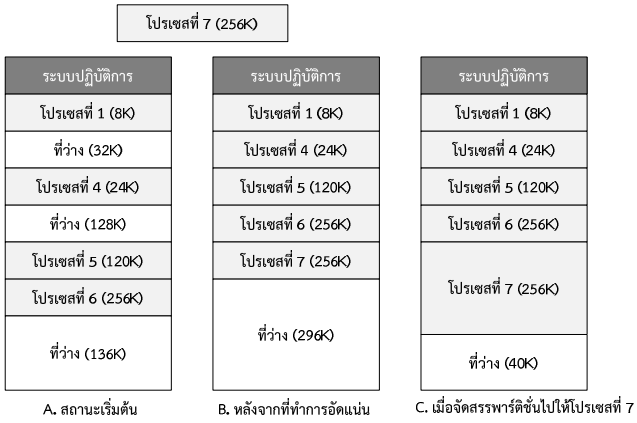
สมมติต่อมาโพรเซสที่อยู่ในหน่วยความจำหลักทั้งสาม ยังไม่สามารถทำงานได้แต่โพรเซสที่ 2 ต้องการที่จะทำงาน แต่พื้นที่ในหน่วยความจำหลักไม่เพียงพอให้โพรเซสที่ 2 ทำงาน ระบบจึงดึงโพรเซส 1 ออกมา ดังภาพที่ 7.12 (g)

จากนั้นระบบปฏิบัติการจึงโหลดโพรเซสที่ 2 เข้าไปใช้งานอีกครั้งหนึ่ง ดังแสดงในภาพที่ 7.12 (h) จากในตัวอย่าง เราอาจจะเห็นว่าระบบน่าจะทำงานได้ดีในระดับหนึ่ง แต่ในความเป็นจริงแล้ว เมื่อระบบทำงานได้ชักระยะหนึ่ง เราพบว่ามีช่องว่างเกิดขึ้นอย่างมากมายในหน่วยความจำหลัก และจะทำให้การใช้งานหน่วยความจำหลักนี้มีประสิทธิภาพที่น้อยลงไป เราเรียกช่องเล็ก ๆ ว่าการสูญเปล่าของพื้นที่ย่อยภายนอก เนื่องจากพื้นที่ภายนอกพาร์ติชั่นในหน่วยความจำหลักเกิดช่องว่างและไม่สามารถนำมาใช้ประโยชน์ได้

ข้อเสียของวิธีนี้ คือ เกิดปัญหาการแตกกระจาย (Fragmentation) เป็นปัญหาที่เกิดขึ้นหลังจากที่มีการใช้หน่วยความจำไประยะหนึ่งแล้ว (มีการจัดสรรพื้นที่หน่วยความจำหลักให้โพรเซส และโพรเซสส่งคืนพื้นที่หน่วยความจำหลักให้กับระบบ) ทำให้มีพื้นที่ว่างที่มีขนาดเล็กกระจายอยู่ทั่วไป โดยพื้นที่แต่ละส่วนนั้นมีขนาดไม่เพียงพอที่จะบรรจุโพรเซสได้ แต่เมื่อรวมหลาย ๆ พื้นที่ว่างเข้าด้วยกันแล้ว จะมีขนาดมากกว่าขนาดของโพรเซสที่ต้องการจัดสรร

7.6.3 การจัดการหน่วยความจำหลักแบบพาร์ติชั่นและย้ายที่อยู่

เนื่องจากวิธีการจัดการหน่วยความจำหลักแบบพาร์ติชั่นมีปัญหาเรื่องการแตกกระจาย ซึ่งสามารถแก้ปัญหาได้ด้วยการรวบรวมพื้นที่ว่างที่มีอยู่นั้นให้ติดกัน เรียกว่า การอัดแน่น (Compaction) แต่การที่ระบบเคลื่อนย้ายโพรเซสที่ถูกบรรจุในพาร์ติชั่นต่าง ๆ เข้าไปอยู่ในพื้นที่บริเวณที่ติดกัน ทำให้ตำแหน่งสำคัญต่าง ๆ ถูกเปลี่ยนแปลงไปด้วย เช่น เลขที่อยู่ฐาน เลขที่อยู่ของหน่วยความจำหลัก จึงจำเป็นต้องปรับเลขที่อยู่ของตำแหน่งดังกล่าวด้วยการย้ายที่อยู่ ซึ่งอาจทำได้ด้วยการนำเอาโปรแกรมไปผ่านโหลดเดอร์อีกครั้งหนึ่ง เพื่อให้โหลดเดอร์ทำการย้ายที่อยู่ วิธีนี้ระบบต้องเสียเวลาในการโหลดโปรแกรมใหม่ทั้งหมด ซึ่งอาจมีบางส่วนของโปรแกรมที่ไม่ได้รับผลกระทบจากการอัดแน่น และไม่จำเป็นต้องย้ายที่อยู่ แต่ต้องถูกย้ายที่อยู่ใหม่อีกครั้ง



ภาพที่ 7.13 วิธีการการจัดการหน่วยความจำหลักแบบพาร์ติชั่นและย้ายที่อยู่

จากภาพที่ 7.13 เป็นตัวอย่างของการอัดแน่น เพื่อแก้ปัญหาการแตกกระจาย ทำให้ได้พื้นที่มากพอที่จะบรรจุโพรเซส 7 แต่หลังจากที่ทำการอัดแน่นแล้วนั้น จำเป็นต้องเคลื่อนย้ายตำแหน่งต่าง ๆ ในหน่วยความจำหลักเป็นจำนวนมาก การอัดแน่นอาจไม่เกิดประโยชน์ ในกรณีที่พื้นที่ว่างที่ได้หลังจากทำการอัดแน่นมีขนาดไม่เพียงพอที่จะบรรจุโพรเซสที่ขอใช้หน่วยความจำหลัก

7.7 ปัญหาการจัดสรรหน่วยเก็บแบบพลวัต

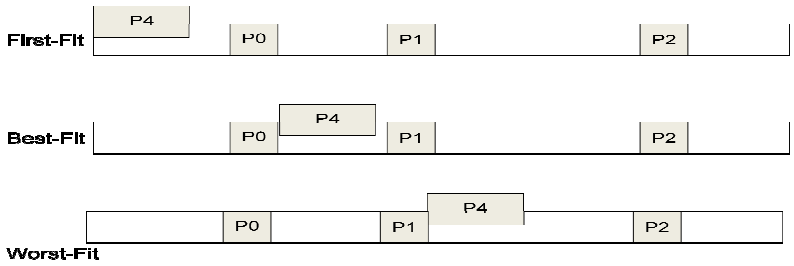
จากข้อจำกัดที่ว่า ทั้งโพรเซสต้องเข้าไปอยู่ในหน่วยความจำหลัก ทำให้มีบางส่วนของโพรเซสที่ไม่จำเป็นต้องใช้งาน แต่จำเป็นต้องถูกนำเข้าไปไว้ในหน่วยความจำหลัก ซึ่งเป็นการสิ้นเปลืองพื้นที่ของหน่วยความจำ ปัญหาข้างต้นเป็นปัญหาในระบบที่มีการจัดสรรพื้นที่แบบยืดหยุ่น (Dynamic storage-allocation problem) ระบบจะพยายามจัดหาพื้นที่ว่างตามขนาดที่ผู้ใช้ร้องขอ มีการจัดสรรหรือเลือกพื้นที่เหมาะสมที่สุดและมีประโยชน์สูงสุด ที่พบบ่อยคือการคืนพื้นที่หน่วยความจำให้กับระบบเมื่อโพรเซสได้ทำงานเสร็จแล้ว ทำให้เกิดพื้นที่ว่างในหน่วยความจำ “โฮล (Hold)” ดังนั้นเราจึงใช้วิธีการวาง (Placement) เพื่อใช้แก้ปัญหาดังกล่าว ซึ่งจะทำให้การจัดสรรพื้นที่ว่างในหน่วยความจำเกิดประโยชน์สูงสุดดังนี้

1) First-Fit คือ เลือกช่องโหว่แรกที่พบและมีขนาดใหญ่เพียงพอกับพื้นที่ที่ต้องการ เป็นวิธีที่ง่ายที่สุดและเสียเวลาน้อยที่สุด

2) Best-Fit คือ การเลือกช่องโหว่ที่เหมาะสมที่สุด ต้องหาเนื้อที่ว่างโดยต้องตรวจดูพื้นที่ว่างทั้งหมดในระบบ เพื่อหาเนื้อที่ว่างที่มีขนาดเท่ากัน หรือใกล้เคียงกับขนาดของโพรเซสจริง มีข้อเสียคือ ใช้เวลานานเพราะต้องนำพื้นที่ว่างทุกรายการเปรียบเทียบกับโพรเซสเพื่อหาขนาดที่เหมาะสม และทำให้เกิดช่องว่างเล็ก ๆ ภายในหน่วยความจำเป็นจำนวนมาก วิธีที่จะทำให้การค้นหาเร็วขึ้นคือ การเรียงรายการตามลำดับจากขนาดเล็กไปใหญ่ ถ้าพบว่าที่ว่างที่โพรเซสนั้นลงได้ ก็ให้ใช้ได้เลยเพราะว่าที่ว่างหลังจากนั้นจะมีค่าใหญ่กว่าแน่นอน จึงเป็นการประหยัดเวลาไม่ต้องนำโพรเซสไปเปรียบเทียบกับทุกพื้นที่ว่างและจะทำให้เหลือพื้นที่เล็กที่สุด

3) Worst-Fit คือ การเลือกช่องโหว่ที่ใหญ่ที่สุด เป็นวิธีป้องกันไม่ให้เกิดปัญหาการเกิดเนื้อที่ว่างเล็ก ๆ เป็นจำนวนมากอย่าง Best-Fit เริ่มต้นเนื้อที่ว่างโดยต้องตรวจดูพื้นที่ว่างทั้งหมดในระบบ และหาที่ว่างที่มีขนาดใหญ่ที่สุดเพื่อใส่โพรเซสใหม่เข้ามา การเลือกจะทำให้เกิดพื้นที่ว่างที่มีขนาดใหญ่ซึ่งอาจมากพอที่จะบรรจุโพรเซสอื่นได้อีก

วิธีการเลือกแบบ First-fit และแบบ Best-fit ดีกว่าแบบ Worst-fit ในแง่ของเวลาที่ลดลงและประสิทธิผลในการใช้ที่เก็บข้อมูล



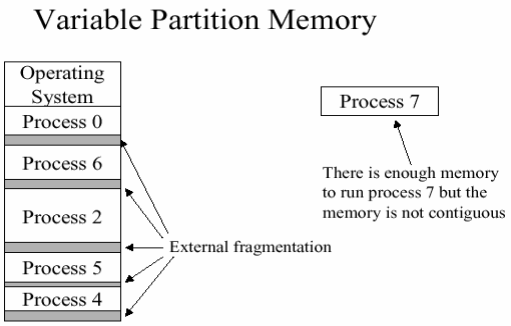
ภาพที่ 7.14 เปรียบเทียบการเลือกพื้นที่ด้วยวิธีต่าง ๆ

7.8 ปัญหาของการจัดการหน่วยความจำ

ปัญหาที่เกิดจากการจัดสรรพื้นที่ที่ตามมาคือ พื้นที่ของหน่วยความจำที่ว่างเป็นช่วง ๆ มีขนาดเล็กไปสำหรับงานที่รอคอยอยู่ประเภทปัญหาของการจัดการหน่วยความจำมีดังนี้

7.8.1 การสูญเปล่าของพื้นที่ย่อยภายนอก (External Fragmentation)

เป็นปัญหาที่เกิดจากการขอพื้นที่หน่วยความจำ เมื่อทำงานในการแจกพื้นที่และตามเก็บพื้นที่ไปสักช่วงเวลาหนึ่ง จะพบว่าพื้นที่ว่างถูกพื้นที่ใช้งานแบ่ง ทำให้พื้นที่ว่างต่อเนื่องยาว ๆ ไม่มีให้ใช้ การที่ระบบมีพื้นที่ว่างพอแต่ไม่สามารถนำมาใช้งานได้ เพราะพื้นที่ว่างถูกพื้นที่ใช้งานแบ่งเป็นพื้นที่ย่อย ๆ มากจนเกินไปจนไม่สามารถนำมาใช้งานได้

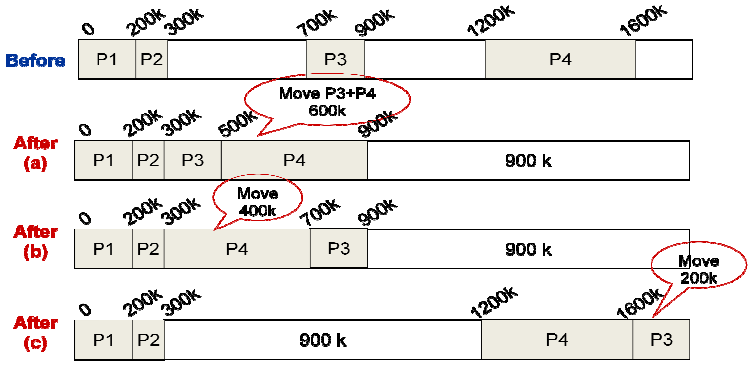


ภาพที่ 7.15 แสดงพื้นที่ว่างเริ่มมีการแตกเป็นส่วน (Fragmentation) เมื่อมีการคืนพื้นที่
ที่มา: B. Thomas Golisano College of Computing and Information Sciences. Retrieved June 27, 2014 from http://www.cs.rit.edu/~hpb/Lectures/99_440/all-inOne-11.html

จากภาพที่ 7.15 จะพบว่าพื้นที่ว่างเริ่มมีการแตกเป็นส่วน (Fragmentation) เมื่อมีการคืนพื้นที่และมีการนำพื้นที่มาใช้ต่อพื้นที่ว่างจะถูกหั่นเป็นส่วน ๆ มากยิ่งขึ้น การเกิดการสูญเปล่าของพื้นที่ย่อยภายนอกจะมีผลกระทบต่อระบบ ทำให้ไม่สามารถใช้งานทรัพยากรได้อย่างเต็มที่ จะเห็นว่ามีพื้นที่ว่างแต่โพรเซส 7 ไม่สามารถเข้าใช้ได้ถึงขนาดพื้นที่เมื่อรวมกันจะมีขนาดใหญ่กว่าโพรเซส 7 แต่เนื่องจากไม่มีพื้นที่ต่อเนื่องกันที่โพรเซส 7 สามารถจะใช้งานได้ ทำให้บางโอกาสที่ควรใช้ได้กลับไม่สามารถใช้ได้ ทั้งที่มีทรัพยากรเหลือเพียงพอ

7.8.2 การแก้ปัญหาการสูญเปล่าของพื้นที่ย่อยภายนอก

ทำได้โดยย้ายตำแหน่งของพื้นที่ใช้งานให้เลื่อนมาติดกัน ซึ่งเราเรียกว่า Compaction แต่การทำเช่นนี้ต้องแลกด้วยต้นทุน (Cost) คือ ค่าใช้จ่ายที่ระบบต้องเสียไปในการทำงาน เช่น เวลา พลังงาน ประสิทธิภาพ ซึ่งทั้งหมดนี้จะมีผลต่องานที่ทำอยู่ และการทำ Compaction คือ การกระชับหน่วยความจำเพื่อให้พื้นที่ว่างรวมกันเป็นผืนเดียว ซึ่งสามารถกระชับได้หลากหลายหนทาง และแต่ละหนทางก็จะมีค่าใช้จ่ายแตกต่างกัน



ภาพที่ 7.16 ตัวอย่างการกระชับหน่วยความจำแบบต่าง ๆ

จากภาพที่ 7.16 ระบบที่ดีต้องเลือกรูปแบบที่กระชับที่สุด เสียค่าใช้จ่ายน้อยที่สุด จึงเลือกแบบ C เป็นรูปแบบกระชับหน่วยความจำที่ดีที่สุด ในการกระชับหน่วยความจำเพื่อแก้ปัญหา การเกิดการสูญเปล่าของพื้นที่ย่อยภายนอกของพื้นที่ว่าง ข้อดี คือ ระบบไม่ซับซ้อน การออกแบบสร้างระบบทำได้ง่าย ข้อเสีย คือ เกิดการสูญเปล่าของพื้นที่ย่อยภายนอกและแก้ไขปัญหานี้ทำได้ยาก

7.8.3 การสูญเปล่าของพื้นที่ย่อยภายใน (Internal Fragmentation)

ปัญหาที่เกิดจากการจองพื้นที่หน่วยความจำมากกว่าขนาดข้อมูล พื้นที่ที่จองเกินนั้นจะเกิดเป็นช่องว่างของพื้นที่สูญเปล่าเพราะถูกจองไว้แต่ไม่ได้ใช้ เป็นปัญหาที่เกิดขึ้นภายในพื้นที่ที่จองไว้

ภาพที่ 7.17 แสดงพื้นที่ที่จองเกินนั้นจะเกิดเป็นช่องว่างของพื้นที่สูญเปล่า
ที่มา: Retrieved June 27, 2014 from http://solid-angle.blogspot.com/2011/02/virtual-addressing-101.html

7.9 การแบ่งพื้นที่เป็นหน้า

เพจจิ้ง คือ การแบ่งหน่วยความจำเป็นหน้า ๆ ขนาดเท่ากัน หลักการของเพจจิ้ง คือ หน่วยความจำจะถูกแบ่งออกเป็นพื้นที่เท่า ๆ กัน เรียงต่อกันไปเรื่อย ๆ ไม่มีช่องว่าง จนหมดพื้นที่หน่วยความจำ การจองหน่วยความจำจะจองเป็นตัวเลขลงตัวเสมอ เช่น ข้อมูลขนาด 3 หน้าครึ่ง จะจองพื้นที่ 4 หน้า เป็นต้น ดังนั้นการรับ Virtual address จากซีพียู และแปลงเป็น Physical address เพื่ออ้างอิงตำแหน่งจริงจากหน่วยความจำ จะเป็นหน้าที่ของ Memory Management Unit เพื่อช่วยลดบทบาทการทำงานของซีพียู

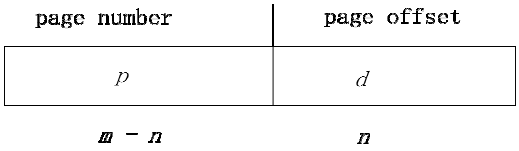
การแบ่งพื้นที่เป็นหน้า เป็นการจัดสรรพื้นที่ว่างบนหน่วยความจำ โดยทำให้โพรเซสที่อยู่บนหน่วยความจำได้โดยไม่ต้องเรียงต่อเนื่องกันทั้งโพรเซส เป็นการใช้พื้นที่ว่างที่อยู่กระจัดกระจายโดยไม่สูญเปล่า และไม่จำเป็นจะต้องบีบอัดพื้นที่ว่างในหน่วยความจำก่อน แบ่งออกเป็น 2 ประเภท ดังนี้

1) การจัดสรรหน่วยความจำทางกายภาพ (paging model of physical memory) เป็นวิธีการแบ่งพื้นที่ให้มีขนาดคงที่ (Fixed-size block) เรียกพื้นที่ส่วนนี้ว่า เฟรม (Frames) โดยที่ขนาดของเพจถูกกำหนดโดยฮาร์ดแวร์ ขนาดของเพจเป็นขนาดยกกำลัง 2 มีขนาดอยู่ระหว่าง 512 ไบต์ถึง 16 MB ต่อเพจ ขึ้นอยู่กับสถาปัตยกรรมของคอมพิวเตอร์

2) การจัดสรรหน่วยความจำทางตรรกะ (paging model of logical memory) เป็นวิธีการแบ่งพื้นที่แบบบล็อก (Block) เรียกพื้นที่ส่วนนี้ว่า เพจ (Pages) โดยกำหนดขนาดเพจเท่ากับขนาดเฟรมทุก ๆ ตำแหน่งจะถูกจัดสรรโดยหน่วยประมวลผลกลาง โดยแบ่งออกเป็นสองส่วน คือ

1) หมายเลขเพจ (Page number: p) โดยหมายเลขเพจจะใช้ดรรชนี (Index) เพื่อชี้ไปยังตารางเพจ (Page table) ซึ่งภายในบรรจุตำแหน่งเริ่มต้น (Base address) ของแต่ละเพจในหน่วยความจำทางกายภาพ

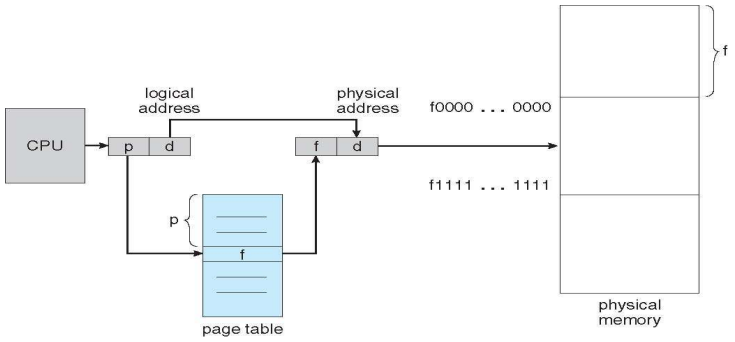
2) ขอบเขตเพจ (Page offset: d) คือ ตำแหน่งจริงในหน่วยความจำ ที่นำมารวมกับตำแหน่งเริ่มต้นที่ได้จากตารางเพจ เพื่อใช้คำนวณหาตำแหน่งของหน่วยความจำทางกายภาพ



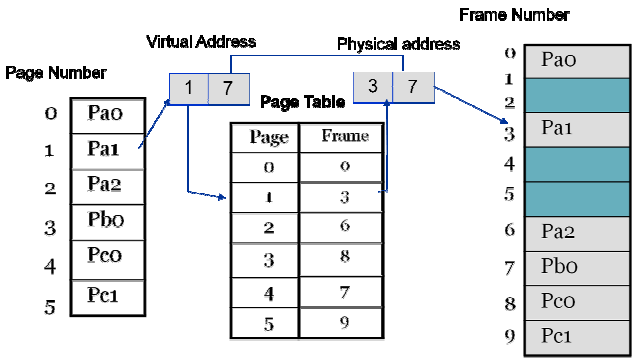
ภาพที่ 7.18 วิธีการอ้างอิงตำแหน่งจริงจากหน่วยความจำ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.369)

7.9.1 การทำงานของ Paging

Paging สามารถใช้จำนวนหน้าที่ไม่ติดกันได้ และในการจองพื้นที่ก็จะไม่เหลือพื้นที่ว่างคั่นไว้ จึงทำให้ไม่เกิดปัญหา External fragmentation



ภาพที่ 7.19 แสดงฮาร์ดแวร์การแบ่งตาราง (Hardware Paging)
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.368)



ภาพที่ 7.20 รูปแบบการแบ่งเพจในหน่วยความจำแบบตรรกะและหน่วยความจำแบบกายภาพ (Paging model of logical and physical memory)

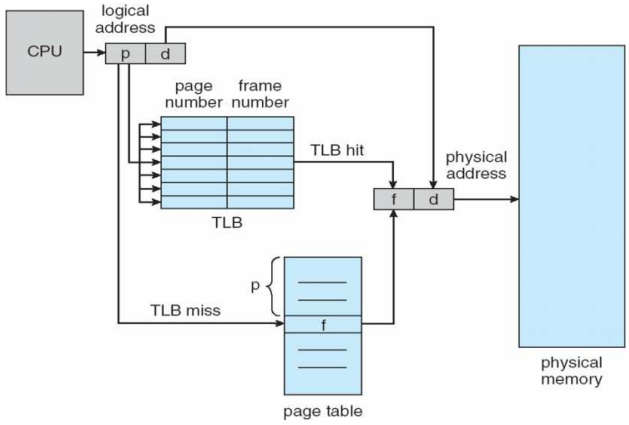
จากภาพที่ 7.20 สามารถอธิบายรูปแบบการแบ่งเพจในหน่วยความจำแบบตรรกะและหน่วยความจำแบบกายภาพ โดย Page table คือ ตารางที่ใช้จับคู่ตัวเลข Page number และ Frame number ซึ่งจะเก็บไว้ในหน่วยความจำ Page number คือ ตัวเลขหน้า เป็นตัวเลขเสมือน (Virtual address) ที่ใช้อ้างอิงตำแหน่งในโปรแกรม Frame number คือ ตัวเลขเฟรม เป็นตัวเลขจริง (Physical address) ที่ใช้อ้างอิงตำแหน่งในหน่วยความจำ

7.9.2 ฮาร์ดแวร์กับการสนับสนุนการแบ่งหน้า (Hardware Support)

การนำฮาร์ดแวร์กับการสนับสนุนการจัดการตารางเพจทำได้หลายทาง ซึ่งการเข้าถึงตำแหน่งในหน่วยความจำแต่ละครั้ง จะต้องอ่านข้อมูลจากหน่วยความจำทางกายภาพถึงสองครั้ง ครั้งที่หนึ่งจาก

ตารางเพจ และครั้งที่สองจากตำแหน่งข้อมูลทางกายภาพ ทำให้เสียเวลาและการเข้าถึงหน่วยความจำล่าช้า ดังนั้นวิธีมาตรฐานที่ใช้แก้ปัญหาจำเป็นต้องใช้ฮาร์ดแวร์ขนาดเล็กที่มีความเร็ว (Hardware cache) ในการอ่านและจัดเก็บข้อมูลสูง (Fast-lookup) ซึ่งเรียกฮาร์ดแวร์ชนิดนี้ว่า “Translation Look-aside Buffer (TLB)”

ข้อมูลจะถูกอ่านจากหน่วยความจำทางกายภาพเพียงครั้งเดียว แล้วจัดเก็บลง TLB ซึ่งเป็นหน่วยความจำที่มีความเร็วสูง (High speed memory) ประกอบไปด้วยสองส่วนด้วยกัน คือ ส่วนที่หนึ่งเก็บกุญแจ (Key) หรือเท็กซ์ (Tag) ส่วนที่สองเก็บค่า (Value) หากต้องการอ่านข้อมูลจากหน่วยความจำทำการเปรียบเทียบกุญแจว่าตรงกันหรือไม่ ถ้าพบรายการที่ค้นหาที่สอดคล้องกับค่าของฟิลด์ (Value field) ก็จะส่งคืนค่ากับไปให้ ประสิทธิภาพของการค้นหาจะรวดเร็ว แต่ฮาร์ดแวร์ชนิดนี้จะมีราคาแพง และค่าของจำนวนข้อมูลที่เข้ามาอยู่ใน TLB จะมีขนาดไม่ใหญ่มากนัก มักมีค่าอยู่ระหว่าง 64 ถึง 1024 ตัว



ภาพที่ 7.21 แสดงฮาร์ดแวร์กับการสนับสนุนการแบ่งหน้าด้วย TLB (Table look-aside buffer)
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.373)

ในกรณีที่ TLB เต็ม ระบบปฏิบัติการก็จะทำการเลือกหรือลบบางเพจออก (Flushed or erased) และนำเพจที่ใช้อยู่ล่าสุดไปแทนที่ แต่ในบาง TLB จะไม่อนุญาตให้ทำวิธีการแบบนี้ได้ ซึ่งจะทำให้เกิดการ “Wired Down” เช่น เพจที่เก็บคำสั่งในส่วนของแกนกลาง (Kernel code) เป็นต้น

ดังนั้นเปอร์เซ็นต์ของเวลาในการค้นหาหมายเลขเพจ (Page numbers) ที่พบใน TLB เราเรียกว่า “อัตราส่วนที่พบ (Hit ratio)” เช่น 80% อัตราส่วนที่พบหมายความว่า เราต้องการค้นหาหมายเลขเพจใน TLB แล้วเจอ 80% ของเวลาทั้งหมด

ถ้าเราใช้เวลา 20 นาโนวินาที ค้นหา ใน TLB และ 100 นาโนวินาที ในการเข้าถึงหน่วยความจำ (Access memory) ดังนั้นเวลาทั้งหมดที่ใช้ไป (Mapped-memory access) จะเท่ากับ 120 นาโนวินาที เมื่อหมายเลขเพจ (Page Numbers) อยู่ใน TLB แต่ถ้าไม่พบหมายเลขเพจใน TLB ซึ่งเสียเวลาไป 20 นาโนวินาที และเสียเวลาครั้งแรกไปในการเข้าถึงหน่วยความจำสำหรับการค้นหาในตารางเพจ 100 นาโนวินาที และเสียเวลาครั้งที่สองในการค้นหาหมายเลขเฟรมอีก 100 นาโนวินาที ซึ่งจะใช้เวลาทั้งหมดในการค้นหาเท่ากับ 200 นาโนวินาที การคำนวณประสิทธิภาพของเวลาในการเข้าถึงหน่วยความจำ (Effective memory-access time) สามารถทำได้ดังนี้

Effective Memory-Access Time = 0.80 X 120 + 0.20 X 220
= 140 Nanoseconds

สรุปได้ว่า เวลาจะช้าลงไปถึง 40% ในการเข้าถึงหน่วยความจำ (คือจาก 100 นาโนวินาที เป็น 140 นาโนวินาที) ถ้า 98% ของอัตราส่วนที่พบหมายความว่าอย่างไร อธิบายได้ดังนี้

Effective Memory-Access Time = 0.98 X 120 + 0.02 X 220
= 122 Nanoseconds

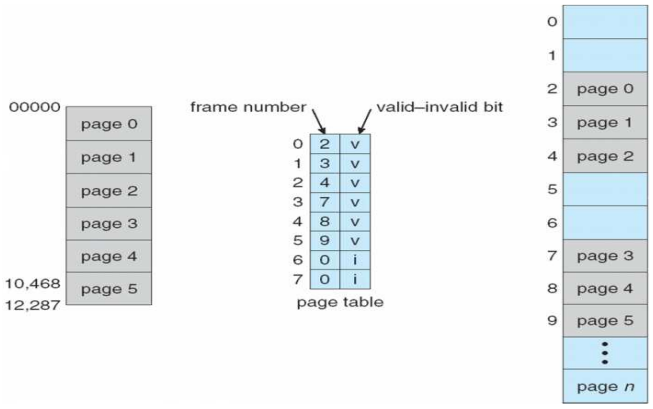
สรุปได้ว่า เวลาจะช้าลงไปถึง 22% ในการเข้าถึงหน่วยความจำ (คือจาก 100 นาโนวินาที เป็น 122 นาโนวินาที)

7.10 การป้องกันหน่วยความจำ

โดยปกติจำนวนบิตที่เก็บอยู่ในตารางเพจ สามารถกำหนดบิตเพื่อใช้ตรวจสอบและกำหนดเพจในการอ่าน-เขียนหรืออ่านข้อมูลเท่านั้น ซึ่งเรียกบิตพิเศษนี้ว่า “กลุ่มบิตป้องกัน (Associating protection bits)” ให้กับทุก ๆ เฟรมที่อยู่ในหน่วยความจำ ซึ่งแบ่งบิตสถานะออกเป็น 2 บิต คือ

1) บิตใช้งานได้ (Valid bit) เป็นบิตสถานะที่บอกว่าข้อมูลในเพจถูกอ่านเข้าสู่หน่วยความจำทางกายภาพแล้ว และสามารถนำไปใช้งานได้ทันที

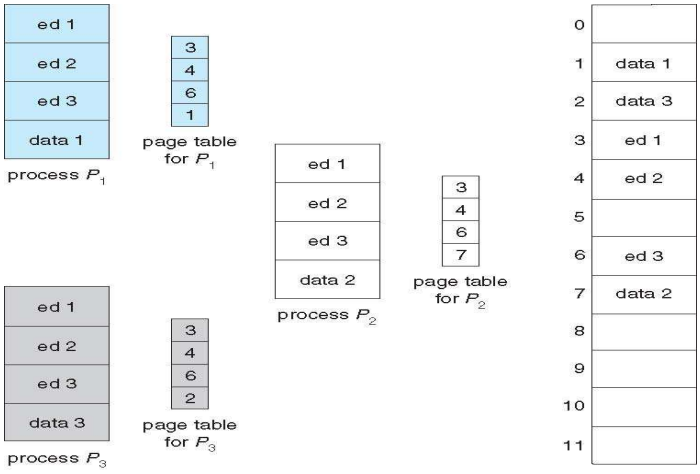
2) บิตใช้งานไม่ได้ (Invalid bit) เป็นบิตสถานะที่บอกว่าข้อมูลในเพจนั้นไม่มีอยู่ในหน่วยความจำทางกายภาพแล้ว และไม่สามารถนำไปใช้งานได้ อาจเกิดจากกรณีที่ระบบปฏิบัติการยังไม่ได้อ่านข้อมูลจากเพจเข้าสู่หน่วยความจำหลัก หรือข้อมูลของเพจที่ต้องการอ่านนั้นถูกสลับออกจากหน่วยความจำแล้ว ซึ่งจะทำให้เกิดข้อผิดพลาดขึ้นที่เรียกว่า “การผิดหน้า (Page fault)” จึงจำเป็นต้องโหลดเพจข้อมูลเข้าสู่หน่วยความจำก่อน แล้วจึงเปลี่ยนค่าของบิตใช้งานไม่ได้ให้เป็นบิตใช้งานได้ แล้วจึงทำการประมวลผลข้อมูลนั้นใหม่อีกครั้งหนึ่ง



ภาพที่ 7.22 ตารางเพจของบิตที่ใช้งานได้และไม่ได้ (Valid (v) or Invalid (i) Bit in a Page Table)
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.376)

7.11 การใช้เพจร่วมกัน

เป็นวิธีการแบ่งปันการใช้เพจร่วมกันในรูปแบบการแบ่งเวลา (Time sharing) โดยข้อมูลนั้นจะเป็นข้อมูลที่ใช้อยู่ในลักษณะไม่สามารถเปลี่ยนแปลงข้อมูลได้ (Non-self-modifying code) ในขณะที่มีการประมวลผลแต่ใช้งานร่วมกันได้ เรียกข้อมูลชนิดนี้ว่า “Reentrant Code” หรือ “Pure Code” ซึ่งจะเก็บไว้ในตำแหน่งทางตรรกะเดียวกัน แต่ละโพรเซสมีข้อมูลเพจ (own data page) เป็นของตัวเอง



ภาพที่ 7.23 แสดงภาพแวดล้อมของการใช้งานเพจร่วมกัน (Sharing of Code In a Paging Environment)
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.377)

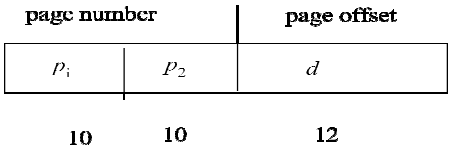
ภาพที่ 7.23 แสดงการใช้เพจร่วมกันระหว่างโพรเซส ซึ่งประกอบไปด้วย 3 โพรเซส คือ P1, P2, และ P3 ต้องการใช้งานโปรแกรม Text Editor ร่วมกันใน Page 0, Page 1 และ Page 3 ที่ถูกจัดเก็บไว้ในหมายเลขเฟรมที่ 3, 4 และ 6 ดังนั้นถ้าระบบปฏิบัติการมีผู้ต้องการใช้งาน 40 คน กำลังใช้งานโปรแกรม Text Editor ซึ่งใช้พื้นที่ 150 KB และใช้เพจสำหรับเก็บข้อมูล 50 KB ดังนั้นหากโพรเซสต่างใช้งานพื้นที่ในหน่วยความจำไม่ร่วมกันจะต้องใช้พื้นที่ในหน่วยความจำเท่ากับ 40 x (150 + 50) = 8000 KB

7.11.1 โครงสร้างของตารางเพจ (Memory Protection)

เทคนิคการสร้างตารางเพจแบ่งออกเป็น 3 รูปแบบ ได้แก่

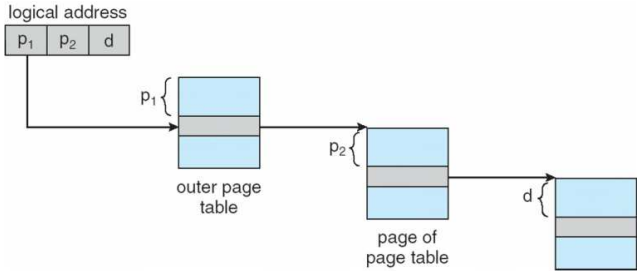
1) โครงสร้างแบบลำดับชั้น (Hierarchical paging) ระบบคอมพิวเตอร์สมัยใหม่จะสนับสนุนตำแหน่งพื้นที่ทางตรรกะ (Logical-address space) ขนาดใหญ่ (232 ถึง 264) ซึ่งโครงสร้างแบบลำดับชั้นเป็นการแบ่งพื้นที่ทางตรรกะออกเป็นตารางเพจหลายตาราง โดยใช้อัลกอริทึมในการแบ่งเพจแบบ 2 ระดับ (Two-level page table) ได้แก่

1.1 หมายเลขเพจ (page number)
1.2 ขอบเขตเพจ (page offset)



ภาพที่ 7.24 แสดงการแบ่งเพจแบบ 2 ระดับ หมายเลขเพจและขอบเขตเพจ

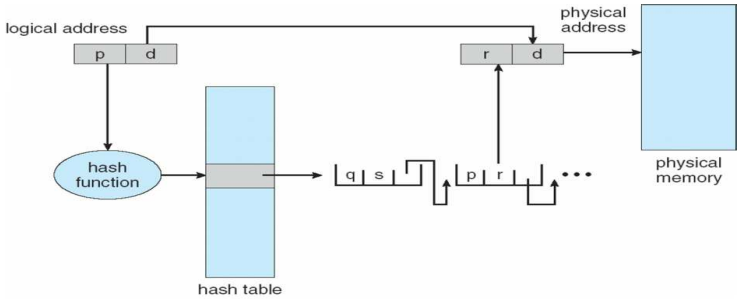
ตัวอย่าง ระบบคอมพิวเตอร์ขนาด 32 บิต ประกอบด้วยเพจขนาด (Page size) 4 KB ตำแหน่งทางตรรกะถูกแบ่งเป็น 2 ส่วน คือ ส่วนแรกขนาด 20 บิต เก็บหมายเลขเพจ (Page number) และส่วนที่สองขนาด 12 บิต เก็บขอบเขตเพจ (Page offset) ซึ่งแสดงโครงสร้างข้อมูลดังนี้



ภาพที่ 7.25 โครงสร้างการแปลงเลขที่อยู่สำหรับสถาปัตยกรรมการสลับหน้าแบบ two-level 32-bit
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.379)

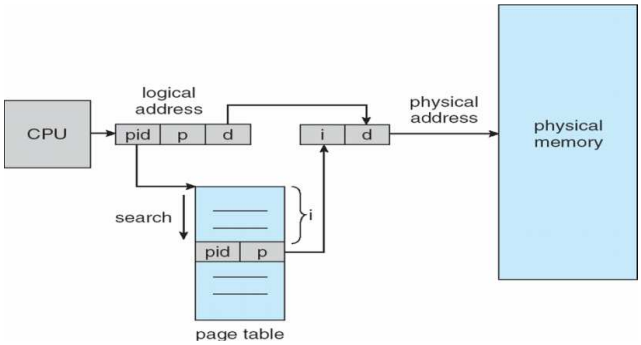
2) โครงสร้างแบบตารางเฮช (Hash page table) ระบบคอมพิวเตอร์ที่จะใช้โครงสร้างแบบนี้ ต้องมีตำแหน่งขนาดพื้นที่มากกว่า 32 บิต โดยค่าของเฮชจะอยู่ภายในหมายเลขเพจเสมือน (Virtual-page number) ซึ่งแต่ละหน่วย (Elements) ในตารางเฮชภายในจะมีลิงก์ลิสต์ (Link list) เชื่อมโยงไปยังหน่วยที่อยู่ในพื้นที่เดียวกัน โดยข้อมูลในแต่ละหน่วยจะประกอบไปด้วย

2.1 ค่าหมายเลขเพจเสมือน (Virtual-page number)
2.2 ค่าดรรชนีที่ชี้ไปยังเฟรมเพจ (Page frame)
2.3 ค่าของพอยเตอร์ ที่ชี้ไปยังหน่วยเชื่อมโยงในลิงก์ลิสต์



ภาพที่ 7.26 แสดงโครงสร้างแบบตารางแฮช (Hashed Page Table)
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.381)

3) โครงสร้างเพจแบบผกผัน (Inverted page table) ระบบคอมพิวเตอร์ที่จะใช้โครงสร้างแบบนี้ ต้องมีคุณสมบัติกำหนดโดยตารางเพจ จะมีเพียงหนึ่งโพรเซสใช้งานพื้นที่หน่วยความจำ โดยโปรแกรมหรือข้อมูลจะประกอบไปด้วยตำแหน่งของเพจเสมือน ที่เก็บอยู่ในหน่วยความจำจริง ดังภาพที่ 7.27 จะมีเพียงหนึ่งตารางเพจเท่านั้นที่อยู่ในหน่วยความจำ โดยแต่ละตำแหน่งของเพจเสมือนประกอบไปด้วย < process-id, page-number, offset >



ภาพที่ 7.27 แสดงโครงสร้างแบบผกผัน (Inverted Page Table)
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.382)

7.11.2 Paging และการแก้ไขปัญหา External Fragmentation

การจองพื้นที่ในเพจจิ้งจะจองเป็น nPage เสมอ โดยค่า n จะเป็นจำนวนเต็ม และ Page คือปริมาณข้อมูลใน 1 หน้ากระดาษ

ตัวอย่าง เรากำหนด ให้ 1 Page เก็บข้อมูล 1024 ไบต์ ดังนั้นถ้าเรามีข้อมูล 9555 ไบต์เราจะต้องจองพื้นที่เท่ากับ 10 Page การจองลักษณะนี้จะทำให้ปัญหา External Fragmentation ไม่เกิด เพราะการจองพื้นที่และคืนพื้นที่จะใช้ในอัตราส่วนเดียวกัน เช่น จอง 3 Page เมื่อใช้เสร็จคืนพื้นที่ ต่อมา 3 Page นั้นถูกนำไปใช้ต่อ 2 Page เหลือช่องว่าง 1 Page เราไม่นับช่องว่างนี้เป็น External Fragmentation เพราะช่องว่างนี้สามารถนำมาใช้งานต่อได้ (ช่องว่างที่เล็กที่สุดในเพจจิ้งจะมีขนาดเท่ากับ 1 Page แต่ในฟรีลิสต์ช่องว่างสามารถเล็กกว่านั้นได้อีก บางกรณีอาจเล็กเพียงแค่ 5-6 ไบต์เท่านั้น) แม้เพจจิ้งจะช่วยแก้ปัญหา External Fragmentation แต่ก็จะเกิดปัญหาใหม่ขึ้นมา คือ Internal Fragmentation

7.12 การแก้ปัญหา Internal Fragmentation

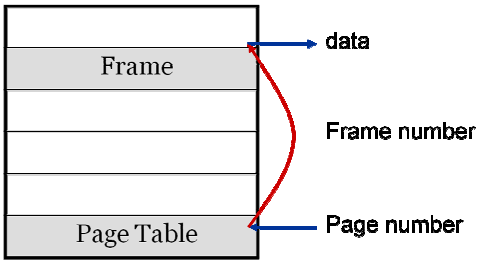
ปัญหา Internal fragmentation คือ การใช้งานพื้นที่ไม่คุ้มค่า จากหลักการของเพจจิ้ง จะมีน้อยครั้งมากที่ขนาดของข้อมูลกับขนาดหน้าจะเท่ากันพอดี ส่วนใหญ่จะเหลื่อม ทำให้ต้องเสียหน้าเพิ่มขึ้นอีกหน้า เกิดช่องว่างไม่ได้ใช้ภายในพื้นที่ที่จอง

ตัวอย่าง ใน 1 Page เรากำหนดให้เก็บข้อมูลได้ 1024 ไบต์ ตัวข้อมูลมี 9555 ไบต์ เพราะฉะนั้นต้องจอง 10 Page แต่ในความเป็นจริง 10 Page คือพื้นที่เท่ากับ 10240 ไบต์ เกิด Internal Fragmentation ขนาด 685 ไบต์

ภาพที่ 7.28 ปัญหา Internal Fragmentation แสดงการเก็บข้อมูลไม่เต็มหน้า เกิดพื้นที่สูญเปล่า

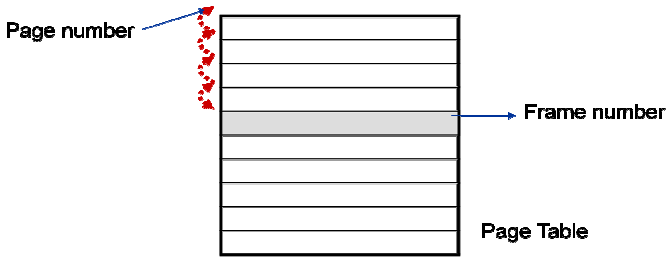
ปัญหา Internal fragmentation ในการใช้งานจริง โดยเฉลี่ยแล้วการจองพื้นที่แต่ละครั้งจะเสียพื้นที่สูญเปล่า (Wasted spaces) ไปประมาณครึ่งหนึ่งเสมอ ดังนั้นจึงต้องเลือกขนาดหน้าให้เล็กที่สุดเพื่อลดการสูญเสียพื้นที่เปล่า แต่ไม่ควรเลือกเล็กจนเกินไป เพราะถ้าแบ่งหน้าถี่ การจอง 1 ครั้งต้องใช้จำนวนหน้าเพิ่ม ข้อมูลในตาราง Page Table ก็มากแถวตาม ถ้าตารางมีข้อมูลมากขนาดใหญ่ การค้นหาข้อมูลเพื่อจับคู่ก็จะช้าตามไปด้วย

ปัญหาจากตารางเพจถือเป็นหัวใจหลักในการทำงานของเพจจิ้ง การเพิ่มตารางทำให้ระบบต้องมีตัวจัดการตารางที่ดี เพราะถ้าไม่ดีอาจกลายเป็นตารางปัญหาของระบบก็เป็นได้ ในระบบเพจจิ้งจึงมีการนำตารางเพจมาใช้ การทำงานของระบบจะช้าลง เพราะในการทำงานแต่ละครั้งระบบต้องอ่านหน่วยความจำสองหน หนแรกเพื่อหาตำแหน่งเฟรม หนที่สองเพื่อทำงานกับข้อมูลที่ต้องการในเฟรมนั้น



ภาพที่ 7.29 การหาตำแหน่งเฟรม เพื่อทำงานกับข้อมูลที่ต้องการในเฟรมนั้น

เมื่อมีการอ่านหน่วยความจำครั้งแรก จะเกิดอะไรขึ้น เมื่อ MMU นำค่าหมายเลขเพจมาเทียบในตารางเพจ การเปรียบเทียบจะเริ่มตั้งแต่ข้อมูลบรรทัดที่ 1 ไล่ไปเรื่อย ๆ จนพบเลขหน้าที่ต้องการ เปรียบเทียบเราจะเรียกว่า Liner Search หรือ Sequential Search มีความสะดวกในการออกแบบสร้างระบบ แต่ทำงานช้าเมื่อนำมาใช้กับข้อมูลปริมาณมาก



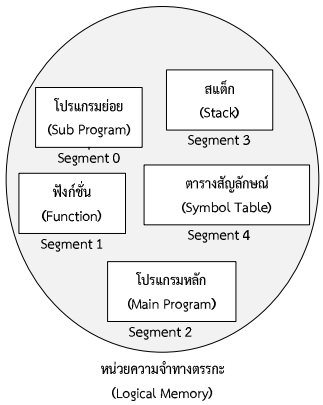
ภาพที่ 7.30 การเปรียบเทียบ (Page Number) มาเทียบในตาราง Page table แบบ Sequential Search

7.13 การแบ่งส่วน

การแบ่งหน่วยความจำเป็นหน้าทำให้เกิดหน่วยความจำทางตรรกะ (ซึ่งผู้ใช้มองเห็น) แตกต่างไปจากลักษณะของหน่วยความจำจริงอย่างหลีกเลี่ยงไม่ได้ โดยมีระบบแปลงตำแหน่งทางตรรกะให้เป็นตำแหน่งจริงทำให้ระบบทำงานได้ถูกต้อง แม้ว่าทั้งสองภาพจะมีความแตกต่างกันก็ตาม

7.13.1 วิธีพื้นฐาน (Basic Method)

วิธีการนี้เป็นการจัดสรรพื้นที่ในหน่วยความจำหลักออกเป็นส่วน ๆ ตามขนาดของโปรแกรมหรือโมดูลย่อย เรียกพื้นที่นี้ว่า Segment โดยแต่ละโมดูลจะถูกแบ่งออกเป็นส่วน ๆ (Segments) ที่มีขนาดไม่เท่ากัน เช่น โปรแกรมหลัก ฟังก์ชัน ตัวแปร บล็อก สแต็ก ตารางสัญลักษณ์ และอาร์เรย์ ซึ่งแต่ละโมดูลจะมีการทำงานสัมพันธ์กัน แสดงดังภาพที่ 7.31



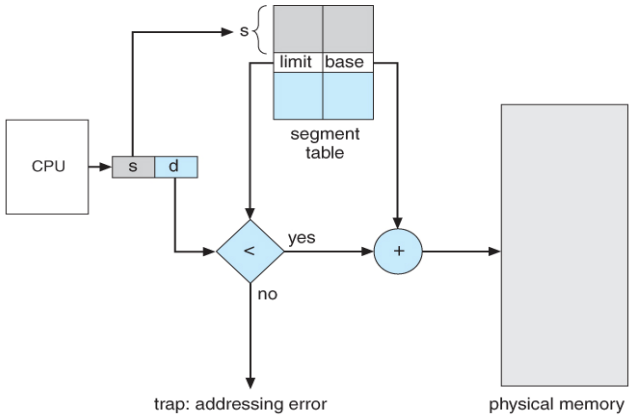
ภาพที่ 7.31 แสดงโครงสร้างแบบแบ่งส่วน (Segmentation)

การแบ่งเป็นตอน (Segmentation) เป็นการจัดการหน่วยความจำหลัก ตามมุมมองของผู้ใช้ หน่วยความจำทางตรรกะจะถูกแบ่งเป็นตอน ๆ แต่ละตอนจะมีชื่อ และขนาด ตำแหน่งอ้างอิง ก็จะมีชื่อตอนกับระยะห่างจากขอบ (Offset) ซึ่งต่างจากระบบแบ่งเป็นหน้า ที่ผู้ใช้อ้างอิงตำแหน่งเป็นค่าเดียว แต่ฮาร์ดแวร์ของระบบ จะแบ่งค่าตัวเลขเป็น 2 ส่วนเอง (เลขหน้าและระยะจากขอบ) โดยที่ผู้ใช้มองไม่เห็น เพื่อความสะดวกชื่อตอน มักใช้ตัวเลขแทนเรียกว่า เลขตอน (Segment-number) โดยปกติตัวแปลภาษา assembly หรือตัวแปลภาษาชั้นสูงอื่น ๆ จะแบ่งโปรแกรมเป็นตอน ๆ โดยอัตโนมัติ เช่น ตัวแปลภาษาปาสคาล อาจแบ่งตอนเป็น ตอนที่ 1 เก็บตัวแปรร่วม (Global variables) ตอนที่ 2 เป็นเนื้อที่สำหรับการเรียกโปรแกรมย่อย เพื่อใช้เก็บค่าตัวแปรต่าง ๆ และตำแหน่งในการเรียกกลับ ตอนที่ 3 เก็บคำสั่งของโปรแกรมย่อยต่าง ๆ ตอนที่ 4 เก็บตัวแปรภายใน (Local variables) สำหรับโปรแกรมย่อย

7.13.2 ฮาร์ดแวร์ (Hardware)

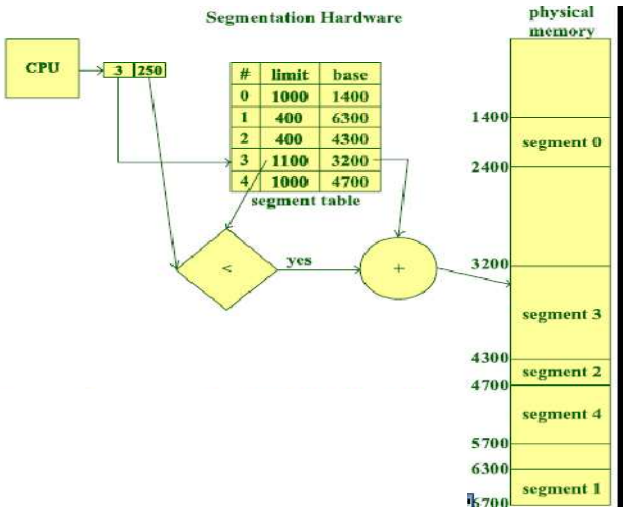
แม้ว่าผู้ใช้สามารถอ้างอิงส่วนต่าง ๆ ของโปรแกรมโดยใช้ตำแหน่งแบบ 2 มิติ แต่หน่วยความจำจริงยังคงเป็นแบบมิติเดียว คือเป็นแถวของคำเรียงต่อกันไป จึงต้องมีวิธีการจับคู่ตำแหน่ง

ทางตรรกะแบบสองมิติ ให้เป็นตำแหน่งจริงมิติเดียว โดยใช้ตารางเลขตอน (Segment table) รูปต่อไปแสดงวิธีใช้ ตารางเลขตอน



ภาพที่ 7.32 โครงสร้างการแบ่งส่วนหน่วยความจำด้วยฮาร์ดแวร์
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.366)

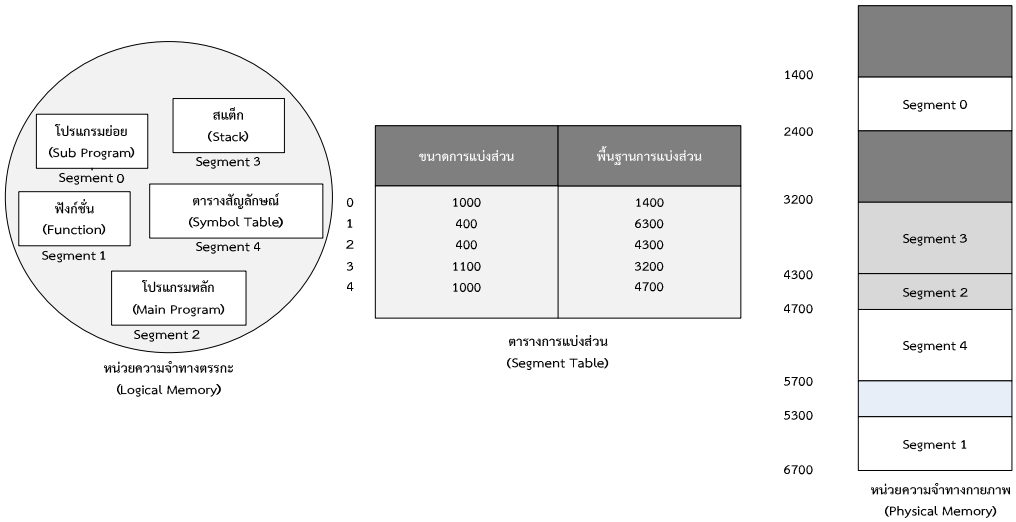
ตำแหน่งทางตรรกะแบ่งได้เป็นสองส่วน คือ หมายเลขตอน (Segment number) ใช้ตัวย่อ s และระยะจากขอบ (offset) ใช้ตัวย่อ d เราใช้หมายเลขตอนเป็นตัวชี้ไปยังข้อมูลในตารางเลขตอน ข้อมูลแต่ละช่องในตารางเลขตอนแ (base) และขอบเขตของตอน (limit) ระยะจากขอบ d จะมีค่าระหว่าง 0 จนถึงค่าขอบเขตของตอน ถ้า d มากกว่าขอบเขตของตอนแล้วจะเกิดข้อผิดพลาด รายงานไปยังระบบปฏิบัติการ (อ้างอิงตำแหน่งออกนอกตอน) ถ้าค่า d ไม่เกินค่าขอบเขตของตอน ฮาร์ดแวร์ก็จะนำค่า d ไปบอกกับฐานเป็นค่าตำแหน่งจริง จะเห็นได้ว่าตารางเลขตอน ก็คือ แถวลำดับ (array) ของรีจีสเตอร์ฐาน และขอบเขต



ภาพที่ 7.33 แสดงการทำงานการแบ่งส่วนหน่วยความจำด้วยฮาร์ดแวร์

จากภาพที่ 7.32 อธิบายได้ว่าหมายเลขตอนที่ 3 มีค่า offset เท่ากับ 250 มีค่าไม่เกินค่าขอบเขตของตอนคือ 1100 มีค่าฐานที่ชี้ไปยังตำแหน่งหน่วยความจำจริง 3200 ดังนั้น หมายเลขตอนที่ 3 นี้จะชี้ไปที่ตำแหน่งที่ 3200 + 250 = 3400

ตัวอย่างในรูปถัดไป มี 5 ตอน หมายเลข 0 จนถึง 4 แต่ละตอนเก็บอยู่ในหน่วยความจำทางกายภาพ ดังแสดงในรูป 7.33



ภาพที่ 7.34 แสดงโครงสร้างตารางการแบ่งส่วน (Segmentation Table)

ตัวอย่าง 1. จากภาพที่ 7.32 กำหนดให้มีการแบ่งส่วนออกเป็น 5 ส่วน มีหมายเลขตั้งแต่ 0 ถึง 4 โดยแต่ละส่วน ถูกจัดเก็บอยู่ในหน่วยความจำทางกายภาพ ภายในตารางการแบ่งส่วนมีข้อมูลอยู่ภายใน โดยกำหนดตำแหน่งเริ่มต้นเรียกว่า ตำแหน่งฐาน (Base) และขนาดของการแบ่งส่วนเรียกว่า Limit อธิบายได้ดังนี้

1) กำหนดให้ Segment 2 มีความยาว 400 ไบต์และตำแหน่งเริ่มต้น (Base of segment) ที่ 4300 ดังนั้นถ้ามีการอ้างอิงไปยังไบต์ที่ 53 ของ Segment

2) ก็จะทำการจับคู่ (Map) ไปยังตำแหน่งเริ่มต้นบวกกับตำแหน่งที่อ้างอิงถึงจะได้เท่ากับ 4300 + 53 = 4353

ตัวอย่าง 2. กำหนดให้ Segment 3 มีความยาว 1000 ไบต์และตำแหน่งเริ่มต้นที่ 3200 ดังนั้นถ้ามีการอ้างอิงไปยังไบต์ที่ 852 ของ Segment 3 ก็จะทำการจับคู่ไปยังตำแหน่งเริ่มต้นบวกกับตำแหน่งที่อ้างอิงถึงจะได้เท่ากับ 3200 + 852 = 4052

ตัวอย่าง 3. กำหนดให้ Segment 0 มีความยาว 1000 ไบต์และตำแหน่งเริ่มต้น (Base of segment) ที่ 1400 ดังนั้นถ้ามีการอ้างอิงไปยังไบต์ที่ 1222 ซึ่ง Segment 0 มีความยาวเพียง 1000 ไบต์ เป็นหน้าที่ของระบบปฏิบัติการในการเลื่อนไปยัง Segment อื่น ๆ ที่มีขนาดความยาว (หน่วยเป็นไบต์) ที่มีขนาดเพียงพอกับตำแหน่งที่ต้องการอ้างถึง

7.13.3 การสร้างตารางเลขตอน (Implementation of Segmentation Tables)

การแบ่งเป็นตอน คล้ายกับการแบ่งหน่วยความจำหลักออกเป็นส่วน ๆ ให้หัวข้อต้น ๆ มีข้อแตกต่างหลัก คือ โปรแกรมหนึ่ง ๆ อาจมีได้หลายส่วน (หรือหลายตอน) การแบ่งเป็นตอนจึงมีความซับซ้อนกว่า (ซึ่งเป็นเหตุผลให้ เราอธิบายเรื่องการแบ่งเป็นหน้าก่อน) เราอาจเก็บตารางเลขตอนไว้ในรีจีสเตอร์พิเศษ (ความเร็วสูง) หรือ ในหน่วยความจำหลัก เหมือนกับตารางเลขหน้า ฮาร์ดแวร์ของเครื่องสามารถเปรียบเทียบค่าขอบเขตกับค่าระยะห่างจากขอบไปพร้อม ๆ กับการบวกกับค่าฐานได้

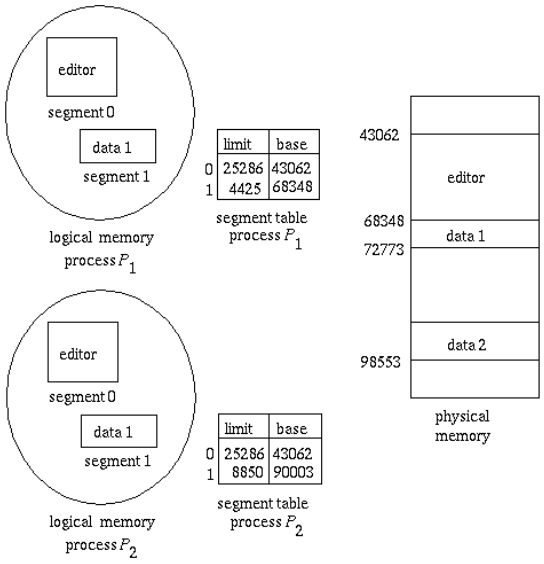
ถ้าโปรแกรมหนึ่งแบ่งเป็นตอน ๆ จำนวนมาก เราก็ไม่สามารถเก็บตารางเลขตอนในรีจีสเตอร์พิเศษได้ ต้องเก็บตารางเลขตอน(ขนาดใหญ่)ไว้ในหน่วยความจำหลักแทน แล้วใช้รีจีสเตอร์พิเศษเก็บตัวชี้ไปยังตารางเลขตอนอีกที (Segment Table Base Register STBR) เนื่องจากโปรแกรมต่าง ๆ มีจำนวนตอนไม่เท่ากัน เราจึงอาจเก็บค่าจำนวนตอนไว้ในรีจีสเตอร์พิเศษเรียกว่า **Segment Table Length Register (STLR)** เช่น ตำแหน่งทางตรรกะเป็น (s,d) เราต้องตรวจดูว่า s < STLR หรือไม่ (หมายเลขตอนไม่เกินจำนวนตอนที่มีจริง) แล้วอ่านค่าข้อมูลจากตารางเลขตอนในหน่วยความจำที่ตำแหน่ง (STBR + s) เมื่อได้ข้อมูลเลขฐานและขอบเขตแล้ว ก็ทำเหมือนเดิมคือ ตรวจว่า d < ขอบเขตหรือไม่ แล้วเอาค่า d บวกกับค่าฐานเป็นตำแหน่งจริงในหน่วยความจำหลัก

เหมือนกับการแบ่งเป็นหน้า การอ่านหน่วยความจำ 2 ครั้งต่อการอ้างอิงตำแหน่งครั้งหนึ่ง ทำให้ระบบทำงานช้าลง 2 เท่า จึงมีการใช้รีจีสเตอร์เสริมหลายตัวช่วยเก็บค่าที่เพิ่งจะใช้ไป และเช่นเดียวกัน เราใช้รีจีสเตอร์เสริมไม่มากนัก ก็สามารถลดเวลาเฉลี่ยในการอ้างอิงหน่วยความจำลงเป็นไม่เกิน 10 หรือ 15 % จากการอ้างอิงแบบโดยตรง

7.13.4 การป้องกันและการใช้ตอนร่วมกัน (Protection and Sharing)

ข้อดีหลักของการแบ่งเป็นตอน คือ สามารถผนวกการป้องกันไปกับแต่ละตอนได้ เพราะแต่ละตอนคือส่วนต่าง ๆ ของโปรแกรมที่มีลักษณะต่าง ๆ กัน ข้อมูลในตอนเดียวกันมักจะมีการใช้งานเหมือน ๆ กัน เช่น ตอนของโปรแกรม ตอนของข้อมูล เป็นต้น ในระบบทั่วไป เราอาจกำหนดตอนของโปรแกรมหรือคำสั่งให้เป็นแบบอ่านได้เท่านั้น (read-only) หรือใช้งานเท่านั้น (execute-only) โดยการใช้บิตป้องกันควบคู่กับแต่ละตอนในตารางเลขตอน เพื่อป้องกันการใช้ผิดประเภท (เช่น เขียนลงในตอนที่ใช้อ่านได้เท่านั้น หรือใช้ทำงานได้เท่านั้น) หรือใส่ข้อมูลประเภทแถวลำดับ (array) ไว้ในตอนเฉพาะ ฮาร์ดแวร์ของระบบก็จะสามารถช่วยตรวจดูได้ว่า มีการอ้างอิงข้อมูลออกนอกขอบเขตแถวลำดับหรือไม่ ดังนั้นฮาร์ดแวร์อาจช่วยตรวจจับข้อผิดพลาดเบื้องต้นต่าง ๆ ในโปรแกรมได้

ข้อดีอีกข้อหนึ่งก็คือ สามารถใช้ข้อมูลหรือโปรแกรมร่วมกันได้สะดวก กระบวนการแต่ละตัวจะมีตารางเลขตอนของตนเอง (เก็บอยู่ในตารางข้อมูลเฉพาะของกระบวนการ) กระบวนการหลายตัวอาจใช้ตอนร่วมกันได้ โดยให้ตัวชี้เลขตอนชี้ไปยังที่เดียวกันในหน่วยความจำจริง ดังภาพที่ 7.35



ภาพที่ 7.35 รูปแบบการป้องกันและการใช้ตอนร่วมกัน
ที่มา: Massey University, Computer science. Retrieved June 27, 2014 from http://www.massey.ac.nz/~mjjohnso/notes/59305/mod8.html

การใช้ข้อมูลร่วมกันนี้เกิดขึ้น ณ ระดับตอน (Segment level) ดังนั้นข้อมูลทุกชนิดที่กำหนดเป็นตอน สามารถใช้ร่วมกันได้โดยไม่จำกัดจำนวนตอนที่จะใช้ร่วมกัน กระบวนการหลายตัวจึงสามารถใช้โปรแกรมร่วมกันได้อย่างสะดวก เช่น การใช้โปรแกรม text editor ในระบบปันส่วน (Time-sharing) ซึ่งมีผู้ใช้หลายคนกำลังใช้โปรแกรมนี้อยู่ โปรแกรมนี้อาจประกอบด้วยหลาย ๆ ตอนซึ่งสามารถใช้ร่วมกันได้ ทำให้ความต้องการเนื้อที่หน่วยความจำจริงโดยรวมลงลงได้อย่างมาก โดยผู้ใช้แต่ละคนใช้ text editor ร่วมกัน แต่มีตอนสำหรับเก็บข้อมูลภายในแยกกัน

7.13.5 การสูญเปล่าพื้นที่ย่อย (Fragmentation)

ตัวจัดตารางการทำงานระยะยาว มีหน้าที่จัดหาเนื้อที่ในหน่วยความจำหลักให้โปรแกรมของผู้ใช้ทุก ๆ คน ซึ่งก็เหมือนในระบบแบ่งเป็นหน้า ยกเว้นว่า การแบ่งเป็นตอน ขนาดของพื้นที่ แต่ละตอนไม่เท่ากัน (ขนาดของหน้า เท่ากันหมดทุกหน้า) ดังนั้น การจัดสรรพื้นที่จะเหมือนกับการจัดสรรพื้นที่แบบขนาดไม่เท่ากัน ซึ่งนิยมใช้แบบ First-fit หรือ Best-fit

การแบ่งเป็นตอนมักทำให้เกิดการสูญเปล่าพื้นที่ย่อยภายนอก ซึ่งเกิดจากพื้นที่ว่างอยู่กระจัดกระจายกัน และแต่ละพื้นที่มีขนาดเล็กไป สำหรับกระบวนการหนึ่ง ๆ กระบวนการอาจรอจนมีพื้นที่ว่างพอหรือระบบอาจบีบอัดหน่วยความจำเพื่อให้เกิดพื้นที่ว่างต่อเนื่องขนาดใหญ่พอเพียง ในกรณีที่พื้นที่ว่างไม่พอ ตัวจัดตารางการทำงานหน่วยประมวลผลกลาง อาจรอหรือข้ามไปดูกระบวนการที่มีศักดิ์ต่ำกว่าก็ได้ ในระบบแบ่งเป็นตอนนี้ ปัญหาการสูญเปล่าพื้นที่ย่อยภายนอกเป็นปัญหามากเพียงไร และการใช้ตัวจัดตารางการทำงานระยะยาว กับการบีบอัดหน่วยความจำ จะช่วยได้หรือไม่ ทั้งสองประการนี้ขึ้นอยู่กับขนาดของตอนเป็นหลัก ถ้าแต่ละกระบวนการมีเพียง 1 ตอน จะทำให้ระบบกลายเป็นการจัดการหน่วยความจำ

แบบแบ่งส่วนไม่เท่ากัน หรือในทางตรงกันข้าม ถ้าแต่ละกระบวนการ ถูกแบ่งเป็นตอนย่อย ๆ ตอนละ 1 คำเท่านั้น ทุก ๆ คำ จะมีตอนเป็นของตัวเอง และสามารถย้ายไปไว้ที่ใดก็ได้ในหน่วยความจำ จะไม่มีการสูญเปล่าพื้นที่ย่อยภายนอกเลย แต่ทุก ๆ ตอน (ทุก ๆ คำ) จะต้องมีค่าเลขฐาน (base) ของตนเอง ทำให้สิ้นเปลืองเนื้อที่ในการเก็บเป็น 2 เท่า ถ้าเราขยายขนาดตอนให้โตขึ้นแต่มีขนาดเท่า ๆ กัน ก็จะกลายเป็นระบบแบ่งเป็นหน้า โดยทั่วไปแล้ว ถ้าตอนมีขนาดเฉลี่ยเล็ก พื้นที่ย่อยภายนอก ก็จะมีขนาดเล็กด้วย (โดยการเปรียบเทียบ สมมติว่า เราพยายามเรียงกระเป๋าเสื้อผ้าหลาย ๆ ใบลงในท้ายรถ แลดูท่าทางจะใส่ไม่หมด แต่ถ้าเราเปิดกระเป๋าแล้วเทข้าวของลงไปไว้ในท้ายรถแทน ก็จะสามารถใส่ลงได้หมด) เพราะว่าแบ่งเป็นหลาย ๆ ตอน แต่ละตอน ย่อมมีขนาดเล็กกว่า 1 กระบวนการ ซึ่งน่าจะจัดสรรลงในหน่วยความจำได้ง่ายกว่า

7.14 สรุป

มีการจัดการหน่วยความจำหลักอยู่หลายระบบ เริ่มจากแบบไม่ซับซ้อนไปถึงแบบซับซ้อน ในบทนี้จะเรียนรู้แบบไม่ซับซ้อน ซึ่งไม่ถูกนำมาใช้งานในระบบปฏิบัติการปัจจุบัน แต่อาจใช้ในคอมพิวเตอร์ขนาดเล็กอยู่ การเรียนรู้เรื่องนี้อาจนำไปประยุกต์ในการพัฒนางานซอฟต์แวร์อื่น ๆ ได้ ระบบโปรแกรมเดียว เป็นวิธีการจัดการที่ง่ายที่สุด โดยกำหนดเพียง 1 โปรแกรม ให้ทำงานในหน่วยความจำเพียงโปรแกรมเดียว ขณะที่ระบบหลายโปรแกรมมีการกำหนดขนาดพาร์ติชั่นคงที่ (Multiprogramming with fixed partition) เพื่อการทำงานของโปรแกรม

ปัจจุบันระบบปฏิบัติการยอมให้มีหลายโพรเซสทำงานพร้อมกันได้ หมายความว่า ขณะที่โพรเซสหนึ่งทำเสร็จ อีกโพรเซสที่รอก็เข้าใช้หน่วยความจำทันที โดยแบ่งหน่วยความจำออกเป็นพาร์ติชั่น การแบ่งเป็นแต่ละพาร์ติชั่นแบบตายตัวมีจุดบกพร่อง จึงมีการจัดการหน่วยความจำแบบมาก่อนได้ก่อน การจัดการแบบนี้ย่อมมีปัญหา จึงเป็นหน้าที่ของระบบปฏิบัติการที่ต้องจัดการ ระบบที่กำหนดขนาดของพาร์ติชั่นให้เปลี่ยนแปลงได้ (Dynamic partition) เพื่อแก้ปัญหาของการกำหนดแบบคงที่ จึงออกแบบการกำหนดพาร์ติชั่นแบบเปลี่ยนแปลงได้ ซึ่งมีความซับซ้อนมากขึ้นตามโพรเซสที่เข้าใช้หน่วยความจำ

กระบวนการจัดการหน่วยความจำประกอบด้วย การย้ายตำแหน่ง ระบบปฏิบัติการในปัจจุบันยอมให้โปรแกรมทำงานพร้อมกันได้หลายงานแบบ Multiprogramming ซึ่งโพรเซสต่าง ๆ เข้าใช้งานหน่วยความจำร่วมกัน จึงต้องมีการสลับโปรแกรมให้เข้าออกหน่วยความจำได้ รวมถึงการเปลี่ยนแปลงค่าตำแหน่งในหน่วยความจำที่อ้างอิงถึงในโปรแกรมให้ถูกต้องตามตำแหน่งจริง ในการป้องกันพื้นที่ระบบปฏิบัติการสามารถป้องกันโพรเซสจากการถูกรบกวนทั้งทางตรงและทางอ้อม

ดังนั้นก่อนให้โพรเซสใดเข้าครอบครองหน่วยความจำ จะต้องมีการตรวจสอบก่อน และใช้เวลาค้นหาเพื่อตรวจสอบตลอดเวลา การใช้พื้นที่ร่วมกันจำเป็นต้องมีการจัดสรรให้ใช้พื้นที่ของหน่วยความจำร่วมกันอย่างยืดหยุ่น การจัดการแบ่งทางกายภาพ หน่วยความจำแบ่งเป็น 2 ส่วนคือ หน่วยความจำหลักและหน่วยความจำสำรอง ลักษณะของหน่วยความจำหลักจะมีราคาแพง ทำงานได้เร็ว แต่เลือนหายได้ ในการทำงานจริงจึงต้องมีการเคลื่อนย้ายทางกายภาพระหว่างหน่วยความจำทั้งสองตลอดเวลา ซึ่งเป็นหน้าที่ของระบบที่ต้องจัดสรรให้สอดคล้องกับการทำงานแบบ Multiprogramming

แบบฝึกหัดท้ายบทที่ 7

จงตอบคำถามต่อไปนี้

1) จงอธิบายการทำงานระบบโปรแกรมเดียวมีวิธีการจัดการหน่วยความจำอย่างไร

2) จงอธิบาย Absolute Address กับ Relative Address แตกต่างกันอย่างไร

3) จงอธิบายMemory Management Unit : MMU มีหน้าที่อย่างไร

4) จงอธิบาย Static Partition กับ Dynamic Partition มีข้อดีข้อเสียแตกต่างกันอย่างไร

5) จงอธิบายถึงสาเหตุของการเกิด External Fragmentation เกิดจากอะไรและมีวิธีแก้ไขการเกิดได้อย่างไร

6) จงอธิบายสาเหตุของการเกิด Internal Fragmentation เกิดจากอะไรและมีวิธีแก้ไขการเกิดได้อย่างไร

7) ปัญหา Dynamic Storage-Allocation Problem มีวิธีการจัดสรรพื้นที่ว่างเมื่อมีการร้องขอขนาด n โดยอธิบายวิธีการทำงานของ First-Fit, Best-Fit และ Worst-Fit พร้อมทั้งเปรียบเทียบให้เห็นว่าวิธีแบบใดให้ผลดีกว่ากันในด้านใดบ้าง

8) จงอธิบายว่า อะไรคือวัตถุประสงค์หลักในการใช้ Paging และ Page Tables

9) พิจารณา Segment table ต่อไปนี้

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

จงตอบคำถามว่าอะไรคือ Physical Addresses ของ Logical Addresses ต่อไปนี้

1) 0,430
2) 1,100
3) 2,500
4) 3,400
5) 4,112

เอกสารอ้างอิง

พิเชษฐ์ ศิริรัตนไพศาลกุล. (2548). **ระบบปฏิบัติการ**. กรุงเทพฯ : ซีเอ็ดยูเคชั่น.

ยรรยง เต็งอำนวย. (2533). **ระบบปฏิบัติการ.** กรุงเทพฯ : ซีเอ็ดยูเคชั่น.

สุจิตรา อดุลย์เกษม. (2552). **ทฤษฎี ระบบปฏิบัติการ Operating Systems**. กรุงเทพฯ : โปรวิชั่น.

Abraham Silberschatz, Peter Baer Galvin, Greg Gagne. (2013). **Operating System Concepts. 9th ed.** Wiley & Sons, Inc.

Andrew S. Tanenbaum, Herbert Bos. (2014). **Modern Operating Systems. 4th ed.** Prentice Hall, Pearson Education International.

Andrew S. Tanenbaum, Maarten van Steen. (2002). **Distributed Systems Principles and Paradigms.** Prentice Hall, Pearson Education International.