บทที่ 2
การจัดการหน่วยความจำ
(Memory Management)

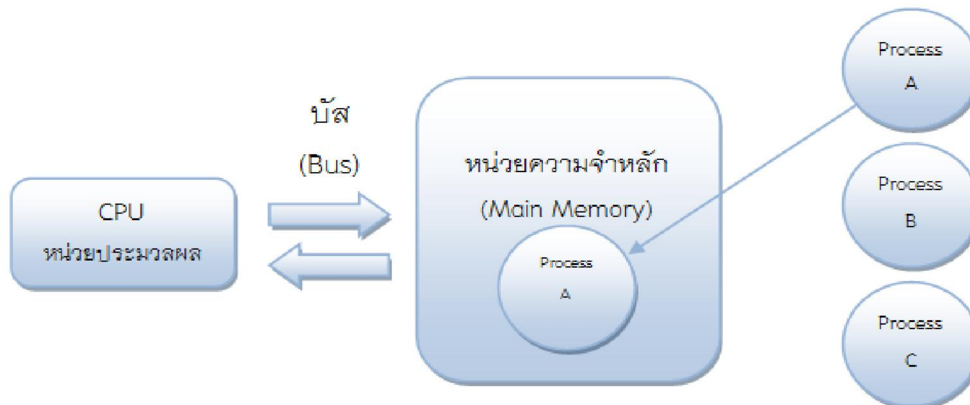
หน่วยความจำถือเป็นส่วนที่สำคัญที่สุดในระบบคอมพิวเตอร์ อีกทั้งยังเป็นศูนย์กลางให้การดำเนินการด้านต่าง ๆ ในระบบคอมพิวเตอร์เป็นไปอย่างราบรื่นและมีประสิทธิภาพสูงสุด โดยภายในหน่วยความจำจะมีการทำงานหลายส่วน เช่น การทำงานของชุดคำสั่งจำนวนมาก ซึ่งต้องมีการแบ่งพื้นที่การใช้งานและวิธีการจัดการด้านต่างๆ ให้กับงาน (Task) เพื่อให้เกิดประสิทธิภาพสูงสุด

ในบทนี้จะกล่าวถึงรายละเอียดของการจัดการหน่วยความจำ (Memory Management) เช่น การเชื่อมโยงตำแหน่ง (address binding) การบรรจุ (load) การใช้คลังชุดคำสั่ง (library) การแบ่งส่วน (Segmentation) การจัดสรรพื้นที่ในหน่วยความจำ (memory allocation) เป็นต้น (วิกิพีเดีย สารานุกรมเสรี, 2556)

2.1 การจัดการหน่วยความจำ (Memory Management)

การจัดการหน่วยความจำสามารถแบ่งออกได้เป็น 2 ส่วน ได้แก่ หน่วยความจำหลัก (Main Memory) และหน่วยความจำเสมือน (Virtual Memory) โดยแต่ละวิธีในการจัดเก็บข้อมูลทั้งสองส่วนมีข้อดีและข้อเสียต่างกันขึ้นอยู่กับซอฟต์แวร์และฮาร์ดแวร์ที่เลือกใช้ว่าสอดคล้องแลสนับสนุนการทำงานและวิธีการที่จัดเก็บในหน่วยความจำที่เลือกใช้มากน้อยเพียงใด แสดงได้ดังภาพที่ 2.1

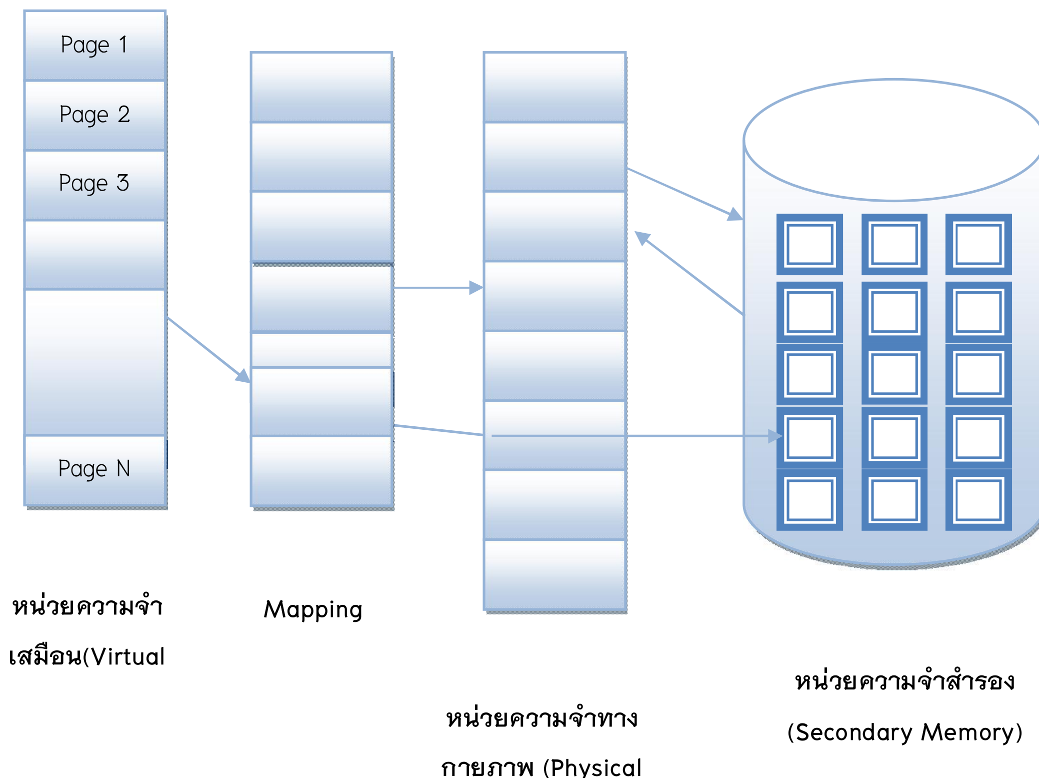
1. หน่วยความจำหลัก (Main Memory) ประกอบไปด้วยอาร์เรย์ขนาดใหญ่ (large array) ซึ่งภายในประกอบเวิร์ด (Words) และไบต์ (Bytes) ซึ่งแต่ละที่จะมีเลขตำแหน่ง (Address) เป็นของตัวเองนอกจากนี้หน่วยความจำหลักยังทำหน้าที่เก็บชนิดกระบวนการในการประมวลผลคำสั่งเพื่อให้หน่วยประมวลผลกลาง (Central Processing Unit: CPU) นำไปใช้ในการประมวลแล้วจึงส่งผลลัพธ์ของคำสั่งนั้น ๆ กลับมาจัดเก็บกลับไว้ในหน่วยความจำหลักอีกที



ภาพที่ 2.1 แสดงกระบวนการทำงานของหน่วยความจำหลัก (Main Memory)

จากภาพที่ 2.1 แสดงให้เห็นกระบวนการทำงานของหน่วยความจำหลัก (Main Memory) โดยที่กระบวนการ A (Process A) จะถูกบรรจุ (load) เข้าไปใช้งานหน่วยความจำหลัก ในขณะที่หน่วยประมวลผลกลาง (CPU) ก็จะบรรจุ (load) กระบวนการ (Process) ที่ต้องการจากหน่วยความจำหลัก ผ่านเส้นทางการรับและส่งข้อมูล (Bus) เพื่อนำไปใช้ในการประมวลแล้วจึงส่งผลลัพธ์ของคำสั่งนั้น ๆ กลับมาจัดเก็บกลับไว้ในหน่วยความจำหลักอีกที

2) หน่วยความจำเสมือน (Virtual Memory) เป็นเทคนิคที่อนุญาตให้กระบวนการ (Process) สามารถประมวลผลได้นอกหน่วยความจำหลักโดยไม่ต้องคำนึงถึงขนาดพื้นที่ใช้ในการประมวลผลว่าเพียงพอกับขนาดของชุดคำสั่งหรือไม่ นอกจากนี้ยังง่ายต่อการแชร์ไฟล์ (Share files) พื้นที่ว่าง (Address Space) และเพิ่มประสิทธิภาพให้กระบวนการทำงานได้เร็วขึ้น เพราะไม่ต้องคอยตรวจสอบขนาดของหน่วยความจำทางกายภาพ



ภาพที่ 2.2 แสดงความสัมพันธ์ระหว่างหน่วยความจำเสมือน (Virtual Memory และหน่วยความจำทางกายภาพ (Physical Memory)

จากภาพที่ 2.2 แสดงให้เห็นความสัมพันธ์ระหว่างหน่วยความจำเสมือน (Virtual Memory) และหน่วยความจำทางกายภาพ (Physical Memory) โดยหน่วยความจำเสมือนเกิดขึ้นจากการแบ่งหน่วยความจำตรรกะ (Logical Memory) ให้ทำงานเป็นอิสระจากหน่วยความจำกายภาพ โดยระบบปฏิบัติการจะทำการบรรจุ (load) ชุดคำสั่งเฉพาะส่วนที่ใช้ประมวลผลเข้าสู่หน่วยความจำตรรกะก่อน ส่วนที่ไม่จำเป็นต้องประมวลผลจะไม่ถูกบรรจุ (load) เข้าสู่หน่วยความจำ ทำให้พื้นที่หน่วยความจำทางตรรกะมีขนาดใหญ่กว่าพื้นที่หน่วยความจำทางกายภาพ โดยการทำงานลักษณะนี้อาศัยการสลับกระบวนการ (Process) เข้าและออกจากพื้นที่หน่วยความจำในขณะกำลังประมวลผล (พีรพร หมุนสนิท, สุธี พงศาสุกุลชัย, อัจจิมา เลี้ยงอยู่, 2553)

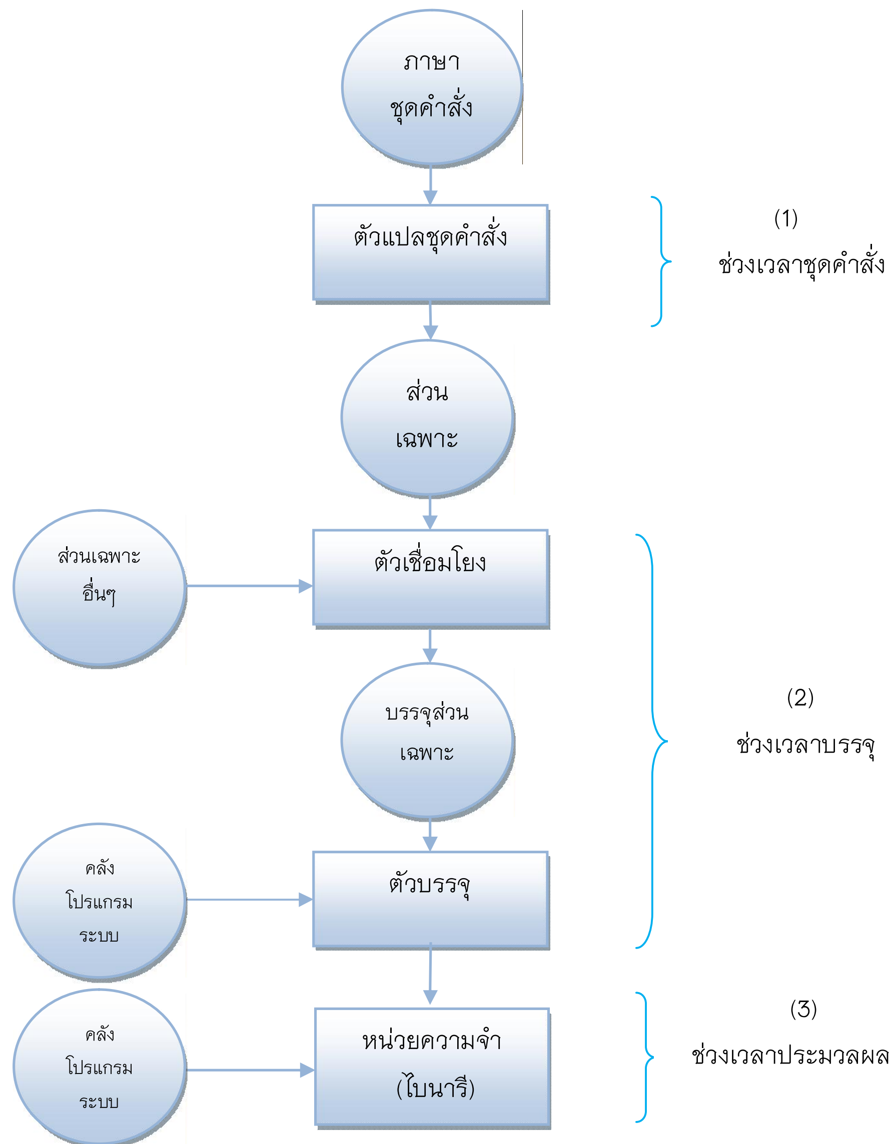
2.2 การเชื่อมโยงตำแหน่ง (Address Binding)

โดยทั่วไปกระบวนการ (Process) ที่จะถูกนำไปประมวลผลจะขึ้นอยู่กับการจัดการหน่วยความจำที่เลือกใช้ กระบวนการอาจจะถูกเคลื่อนย้ายกลับไปกลับมาระหว่างจานบันทึก (disk) และหน่วยความจำก่อนที่มันจะถูกประมวลผล โดยระบบปฏิบัติการจะมีการเก็บกระบวนการที่รอประมวลผลตามลำดับ (input queue) ของกระบวนการที่จะนำเข้ามาประมวลผลในหน่วยความจำรวมทั้งยังเชื่อมโยงกับค่าเริ่มต้นของตำแหน่ง ระบบคอมพิวเตอร์มักเริ่มต้นค่าตำแหน่งเชื่อมโยงที่ค่าเริ่มต้นที่ตำแหน่ง 0 และแบ่งค่าที่เชื่อมโยงตำแหน่ง ได้เป็นค่าจริง (Absolute address) ของกระบวนที่อยู่ในหน่วยความจำ (Memory) และค่าตำแหน่งที่สัมพันธ์กัน (Relative address) หรือชุดคำสั่งต่าง ๆ ที่ได้รับหลังจากการแปลชุดคำสั่ง (Compile) การจำแนกการเชื่อมโยงของคำสั่งและตำแหน่งของข้อมูลในหน่วยความจำสามารถแบ่งตามขั้นตอนแต่ละช่วงเวลา แสดงได้ดังภาพที่ 2.3 ดังนี้

1. ช่วงเวลาแปลชุดคำสั่ง (Compile time) คือ ช่วงเวลาที่ชุดคำสั่งหรือข้อมูลถูกแปลโดยตัวแปลชุดคำสั่ง (Compiler) โดยตัวแปลชุดคำสั่งจะค้นหาตำแหน่งจริง (Absolute Code) ในหน่วยความจำเมื่อพบแล้วจะสร้างชุดคำสั่งทำให้ระบบปฏิบัติการสามารถประมวลผลคำสั่งหรือข้อมูลนั้นได้ทันที แต่หากตำแหน่งจริงถูกเปลี่ยนแปลง ตัวแปลชุดคำสั่งก็จะทวนการแปลชุดคำสั่ง (Recompile) ชุดคำสั่งหรือข้อมูลนั้นขึ้นใหม่ทุกๆ ครั้ง

2. ช่วงเวลาบรรจุ (Load time) ช่วงเวลาที่ชุดคำสั่งหรือข้อมูลถูกบรรจุเข้าสู่หน่วยความจำโดยลำดับแรกตัวแปลชุดคำสั่งจะสร้างชุดคำสั่งที่สามารถประมวลผลได้ทันที (Relocation code) ขึ้นมาก่อน หลังจากชุดคำสั่งหรือข้อมูลถูกบรรจุเข้าสู่หน่วยความจำแล้วตัวแปลชุดคำสั่งจะทำการแปลตำแหน่งที่บรรจุเข้ามาให้เป็นตำแหน่งจริง (Absolute Code) เพื่อให้ระบบปฏิบัติการสามารถประมวลผลคำสั่งหรือข้อมูลนั้นได้โดยไม่ต้องเสียเวลาในการแปลชุดคำสั่งใหม่ทุกครั้ง แต่จะเสียเวลาเฉพาะตอนบรรจุคำสั่งหรือข้อมูลนั้นเข้ามาในหน่วยความจำ

3. ช่วงเวลาประมวลผล (Execution time) ช่วงเวลาที่คำสั่งหรือข้อมูลถูกประมวลผล โดยตัวแปลชุดคำสั่ง โดยจะทำการเชื่อมโยงตำแหน่งและแปลชุดคำสั่งคำสั่งหรือข้อมูลตรงตำแหน่งนั้นๆ เข้าไปเก็บไว้ในหน่วยความจำ ในขณะที่อยู่ในช่วงเวลาประมวลผลระบบปฏิบัติการจะเสียเวลาในการแปลตำแหน่งคำสั่งหรือข้อมูลต่างๆ ก่อนถูกนำมาเข้าสู่กระบวนการประมวลผลใหม่ทุกครั้ง (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)



ภาพที่ 2.3 แสดงช่วงเวลาเชื่อมโยงตำแหน่ง (Address Binding)

จากภาพที่ 2.3 แสดงช่วงเวลาเชื่อมโยงตำแหน่ง (Address Binding) ในแต่ละช่วง ดังนี้ ช่วงเวลาที่ 1 ช่วงเวลาแปลชุดคำสั่ง (Compile time) ใช้ในการแปลชุดคำสั่งในส่วนเฉพาะ (module) เพื่อส่งชุดคำสั่งหรือข้อมูลที่ถูกแปลแล้วเข้าสู่ช่วงที่ 2 ช่วงเวลาบรรจุ (Load time) ชุดคำสั่งทั้งหมดเข้าสู่ ช่วงเวลาที่ 3 ช่วงเวลาประมวลผล (Execution time) ช่วงเวลานี้ชุดคำสั่งหรือข้อมูลที่ถูกประมวลผลทั้งหมดจะถูกจัดเก็บลงในหน่วยความจำอีกครั้งหนึ่ง

2.3 การเชื่อมโยงระหว่างพื้นที่ทางกายภาพกับพื้นที่ทางตรรกะ (Logical– Versus Physical–Address Space)

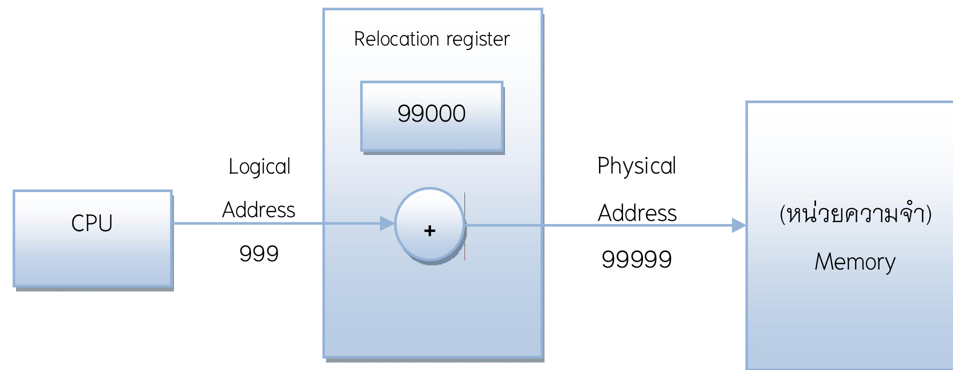
การเชื่อมโยงพื้นที่ในหน่วยความจำจะต้องมีการอ้างอิงถึงตำแหน่งที่เกี่ยวข้องอยู่ 2 ประเภท คือ

1. ตำแหน่งพื้นที่ทางตรรกะ (Logical Address Space)

บางครั้งเรียกตำแหน่งนี้ว่า ตำแหน่งเสมือน (Virtual Address) ซึ่งถูกสร้างขึ้น (Generate) โดยหน่วยประมวลผลกลาง (CPU) เพื่อใช้ในการแลกเปลี่ยนข้อมูล โดยกลุ่มของตำแหน่งพื้นที่ทางตรรกะ (Logical Address Space) ทั้งหมดถูกสร้างโดยชุดคำสั่ง

2. ตำแหน่งพื้นที่ทางกายภาพ (Physical Address Space)

คือตำแหน่งจริงที่ทำงานอยู่ในหน่วยความจำหลัก (Memory Unit) และทำงานโดยตอบสนอง (Corresponding) กับตำแหน่งทางตรรกะ (Logical Address) เสมอ โดยช่วงเวลาที่ชุดคำสั่งกำลังทำงานอยู่ (Run–Time) ตำแหน่งทางตรรกะ (Logical Address) และตำแหน่งทางกายภาพ (Physical Address) จะถูกจับคู่ (Mapping) โดยส่วนเครื่องหรืออุปกรณ์ (Hardware) ชนิดหนึ่งที่เรียกว่า หน่วยจัดการหน่วยความจำ (Memory Management Unit : MMU) ทำหน้าที่ในแปลงตำแหน่งทางตรรกะ (Logical Address) ไปเป็นตำแหน่งทางกายภาพ (Physical Address) โดยนำค่าที่ได้เก็บไว้ในรีจิสเตอร์พื้นฐาน (Base–Register) ส่วนการย้ายตำแหน่ง (Relocation Register) ก็จะนำค่าที่ได้เก็บไว้ในรีจิสเตอร์พื้นฐาน (Base–Register) มาบวกกับค่าตำแหน่งทางตรรกะ (Logical Address) เพื่อใช้ในการหาค่าของตำแหน่งทางกายภาพ (Physical Address) แล้วส่งผลลัพธ์ที่ได้เข้าสู่หน่วยความจำ (Memory) โดยกำหนดค่าเริ่มต้นตำแหน่งทางตรรกะ (Logical Address) ให้อยู่ในช่วงระหว่างต่ำสุด (ค่าเริ่มต้นที่ค่าศูนย์ (0)) ถึงค่าสูงสุด (max) และกำหนดค่าเริ่มต้นตำแหน่งทางกายภาพ (Physical Address) อยู่ในช่วงระหว่าง R + 0 ถึง R + max (R คือ Relocation Register เป็นรีจิสเตอร์สำหรับการย้ายตำแหน่ง) แสดงได้ดังภาพที่ 2.4



ภาพที่ 2.4 แสดงตัวอย่างของการทำงานของหน่วยจัดการหน่วยความจำ (Memory Management Unit: MMU)

จากภาพที่ 2.4 แสดงให้เห็นตัวอย่างของการทำงานของหน่วยจัดการหน่วยความจำ (Memory Management Unit: MMU) โดยที่หน่วยประมวลผลกลาง (CPU) จะไม่สามารถเข้าถึงหน่วยความจำ (Memory) ได้โดยตรงเป็นหน้าที่ของส่วนเครื่องหรืออุปกรณ์ (Hardware) ที่เรียกว่า หน่วยจัดการหน่วยความจำ (Memory Management Unit : MMU) โดย MMU จะทำการแปลงตำแหน่งทางตรรกะ (Logical Address) โดยนำเอาค่าเริ่มต้นที่กำหนดขึ้นมาคือค่า 999 จับคู่ (Mapping) หรือบวกเข้ากับค่ารีจิสเตอร์สำหรับการย้ายตำแหน่ง (Relocation Register) คือค่า 99000 ผลลัพธ์ที่ได้จะเป็นค่า 99999 คือค่าของตำแหน่งทางกายภาพ (Physical Address) หลังจากนั้น MMU ส่งผลลัพธ์ที่ได้เข้าสู่หน่วยความจำ (Memory) จึงจะทำให้หน่วยประมวลผลกลาง (CPU) เข้าถึงหน่วยความจำ (Memory) ได้นั่นเอง

2.4 การบรรจุแบบพลวัต (Dynamic Loading)

เนื่องจากพื้นที่ที่ใช้ในการประมวลผลในหน่วยความจำทางกายภาพ (Physical Memory) มีการจำกัดขนาดข้อมูล ดังนั้นชุดคำสั่งและข้อมูลทั้งหมดของกระบวนการ (Process) จะต้องมีขนาดเล็กกว่าหน่วยความจำทางกายภาพ (Physical Memory) ในขณะที่บรรจุข้อมูลทั้งหมดเข้าสู่พื้นที่ว่างในหน่วยความจำ โดยในแต่ละครั้งจะต้องมีการตรวจสอบขนาดของข้อมูลเพื่อจะช่วยให้การประมวลผลชุดคำสั่งและการบรรจุข้อมูลเข้าหน่วยความจำทำได้รวดเร็วยิ่งขึ้น ลักษณะการทำงานดังกล่าวนี้เรียกว่า การบรรจุแบบพลวัต (Dynamic Loading) ซึ่งวิธีการทำงานจะไม่บรรจุชุดคำสั่งย่อย (Routine) จนกว่าจะมีการถูกเรียกใช้งาน (Call) ทำให้ไม่สิ้นเปลืองเนื้อที่ในหน่วยความจำ โดยทุกครั้งที่ชุดคำสั่งหลัก (Main Program) ถูกบรรจุเข้าสู่

หน่วยความจำเพื่อทำการประมวลผลและมีการเรียกใช้ชุดคำสั่งย่อย (Routine) จะมีการตรวจสอบชุดคำสั่งย่อยตัวแรกก่อน (Routine First) ควบคู่กับการบรรจุชุดคำสั่งย่อยตัวอื่นพร้อมกันไปด้วย กรณีที่ไม่มีการบรรจุชุดคำสั่งย่อย ตัวเชื่อมโยงการย้ายตำแหน่ง (Relocation linking Loader) จะทำการบรรจุชุดคำสั่งย่อยไปเก็บไว้ในตารางเพื่อระบุตำแหน่ง (Address Table) ซึ่งตัวชุดคำสั่งจะทำการปรับเปลี่ยนตำแหน่งที่มีผลกระทบกับการเคลื่อนย้ายตำแหน่งโดยไม่ต้องอาศัยระบบปฏิบัติช่วยในการจัดการ ซึ่งตัวชุดคำสั่งจะเป็นตัวจัดการเหตุการณ์ดังกล่าวเอง

2.5 การใช้คลังโปรแกรมร่วมกันและการเชื่อมโยงแบบพลวัต (Dynamic Linking and Shared Libraries)

วิธีการเชื่อมโยงแบบพลวัต (Dynamic Linking) มีความคล้ายกันกับการบรรจุแบบพลวัต (Dynamic Loading) ต่างกันตรงเวลาที่ใช้ในการเชื่อมโยงที่มากกว่าเมื่อระบบต้องการเรียกใช้งานชุดคำสั่งย่อยในคลังโปรแกรม (Subroutine Libraries) จะมีชุดคำสั่งขนาดเล็กที่ชื่อว่าสตั๊บ (Stub) ทำการตรวจสอบว่ามีชุดคำสั่งย่อยในคลังโปรแกรมที่ต้องการใช้งานว่าอยู่ในหน่วยความจำแล้วหรือยัง ถ้ายังไม่มี ชุดคำสั่งสตั๊บ (Stub) จะมีหน้าที่ทำการบรรจุชุดคำสั่งย่อยใหม่เข้าสู่หน่วยความจำทันที พร้อมกับแทนที่ค่าตำแหน่งชุดคำสั่งย่อยใหม่ทับตำแหน่งชุดคำสั่งย่อยเดิมโดยเรียกกระบวนการทำงานนี้ว่า วิธีการเชื่อมโยงแบบพลวัต (Dynamic Linking) โดยที่ทุกๆ กระบวนการจะใช้ภาษาในคลังโปรแกรมทำการประมวลมลเพียงหนึ่งครั้งเท่านั้น พร้อมกับทำสำเนารหัส (Library code) ซึ่งเป็นไฟล์ที่มีนามสกุลเป็น .dll พร้อมกับทำการปรับปรุงไฟล์ในคลังข้อมูลและสร้างการเชื่อมโยงใหม่ทันที เพื่อให้ระบบสามารถเรียกใช้งานไฟล์ .dll ร่วมกันจากชุดคำสั่งย่อยในคลังโปรแกรมได้ (Shared Libraries)

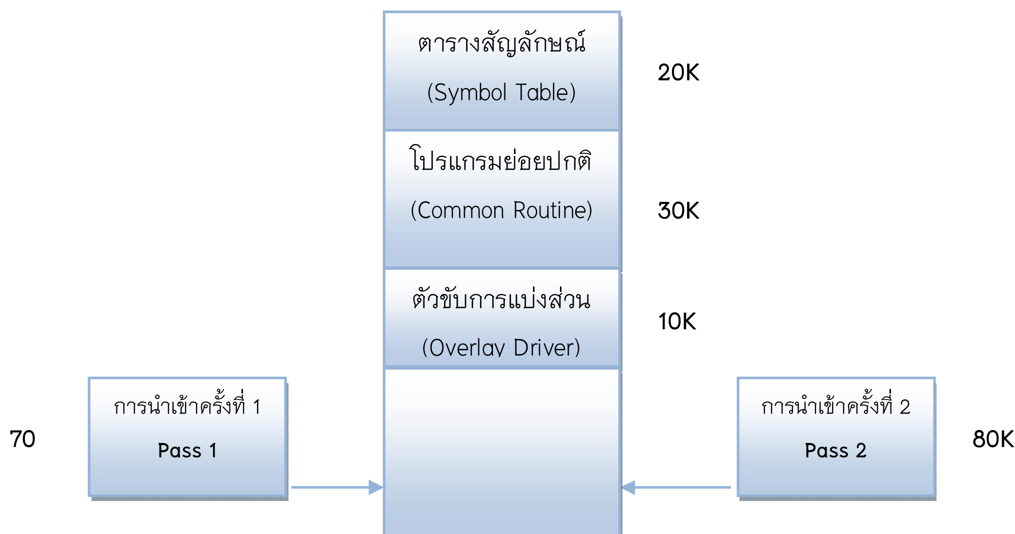
2.6 การแบ่งส่วน (Overlay)

เป็นวิธีที่ใช้ในกรณีที่กระบวนการมีขนาดใหญ่กว่าหน่วยความจำที่ต้องการจัดเก็บจำเป็นที่จะต้องจัดสรรหน่วยความจำให้เหมาะสมโดยการแบ่งส่วน (Overlay) หลักการทำงานของการแบ่งส่วนโดยที่คำสั่งและข้อมูลจะถูกเก็บอยู่ในหน่วยความจำตามขนาดของหน่วยความจำที่มีอยู่ ซึ่งจะต้องอาศัยการทำงานของตัวขับในการแบ่งส่วน (Overlay Driver) เพื่อบรรจุคำสั่งที่ต้องการใช้งานหน่วยความจำเข้ามาใช้งานจนเสร็จก่อนแล้วจึงสลับให้คำสั่งอื่นที่ต้องการใช้งานหน่วยความจำเข้ามาทำใช้งานหน่วยความจำครั้งต่อไป

ตัวอย่าง พิจารณาจากตัวแปลภาษาเอสเซมเบลอร์ (Assembler) ในการจัดการนำ 2 กระบวนการเข้าการทำงาน โดยที่กระบวนการ 1 ต้องการที่จะใช้ตารางสัญลักษณ์ (Symbol Table) ขณะที่กระบวนการ 2 ต้องการที่จะสร้างคำสั่งภาษาเครื่อง (Machine-language code) เราอาจจะต้องทำการแบ่งพาร์ทิชั่น (Partition) ให้ตัวแปลภาษาเอสเซมบลี ในการจัดการการนำเข้าการทำงานของคำสั่งในกระบวนการ 1 คำสั่งในกระบวนการ 2 การใช้งานตารางสัญลักษณ์ (Symbol Table) และชุดคำสั่งย่อยปกติ (Common Routine) ในหน่วยความจำร่วมกันทั้งสองกระบวนการ โดยกำหนดขนาดการใช้งานหน่วยความจำของแต่ละคำสั่งได้ดังนี้

การนำเข้าครั้งที่ 1 (Pass 1)	70 KB
การนำเข้าครั้งที่ 2 (Pass 2)	80 KB
ตารางสัญลักษณ์ (Symbol Table)	20 KB
ชุดคำสั่งย่อยปกติ (Common Routine)	30 KB

จากตัวอย่าง จะเห็นว่าการบรรจุทุกคำสั่งหนึ่งครั้งจะต้องใช้หน่วยความจำถึง 200KB ซึ่งในระบบมีขนาดของหน่วยความจำเพียง 150KB ทำให้ไม่สามารถประมวลผลกระบวนทั้ง 2 กระบวนการที่มีอยู่ได้ จากการสังเกตแล้วพบว่าการนำเข้ากระบวนครั้งที่ 1 และการนำเข้ากระบวนครั้งที่ 2 ไม่ต้องการใช้หน่วยความจำในเวลาเดียวกัน แสดงได้ดังภาพที่ 2.5



ภาพที่ 2.5 แสดงการแบ่งส่วนหน่วยความจำสำหรับการนำเข้าของกระบวนการครั้งที่ 1 และครั้งที่ 2 ของตัวแปลภาษาเอสเซมเบลอร์ (Overlays for two-pass assembler)

จากภาพที่ 2.5 แสดงวิธีการแก้ปัญหา การแบ่งส่วนหน่วยความจำสำหรับการนำเข้าของกระบวนการครั้งที่ 1 และกระบวนการครั้งที่ 2 ของตัวแปลภาษาเอสเซมบลี (Overlays for two-pass assembler) โดยระบบจะแบ่งการทำงานออกเป็นสองส่วน (Two Overlays) คือ Overlay A ทำงานกับตารางสัญลักษณ์ (Symbol Table) ชุดคำสั่งย่อยปกติ (Common Routine) และการนำเข้ากระบวนการครั้งที่ 1 (Pass 1) ส่วน Overlay B ก็ทำงาน (Symbol Table) ชุดคำสั่งย่อยปกติ (Common Routine) และการนำเข้ากระบวนการครั้งที่ 2 (Pass 2) ระบบจะทำการเพิ่มตัวขับในการแบ่งส่วน (Overlay Driver) ขนาด 10 KB เพื่อบรรจุคำสั่งใน Overlay A ที่ต้องการใช้งานหน่วยความจำ 120KB จนทำงานเสร็จ แล้วจึงกระโดดกลับไปที่ตัวขับในการแบ่งส่วนอีกครั้งเพื่อบรรจุคำสั่งใน Overlay B ที่ต้องการใช้งานหน่วยความจำ 130KB โดยเข้าไปแทนที่ (Overwriting) Overlay A ซึ่งจะทำให้ตัวแปลภาษาเอสเซมบลี สามารถที่จะประมวลบนพื้นที่หน่วยความจำขนาด 150KB ที่มีอยู่อยากจำกัดได้

2.7 กลยุทธ์ในการจัดการหน่วยความจำ (Memory Strategy)

การจัดการหน่วยความจำถือว่าเป็นหน้าที่หนึ่งของระบบปฏิบัติการ เพื่อจัดสรรพื้นที่หน่วยความจำให้กับชุดคำสั่งหรือข้อมูลต่างๆ ได้ถูกต้องเหมาะสม แบ่งออกเป็น 3 วิธี ดังนี้

1. กลยุทธ์การบรรจุ (Fetch Strategy) เป็นกลยุทธ์ที่ใช้ในการบรรจุคำสั่งหรือข้อมูลต่าง ๆ จากหน่วยเก็บข้อมูลสำรองเข้าสู่หน่วยความจำหลัก แบ่งออกเป็น 2 วิธี

1.1 Demand Fetch Strategy คือ วิธีการบรรจุเฉพาะคำสั่งหรือข้อมูลต่าง ๆ ที่ต้องการใช้งานเข้าสู่หน่วยความจำหลัก ซึ่งโดยปกติถือกันว่าคำสั่งหรือข้อมูลต่าง ๆ ของกระบวนการถูกนำเข้าหน่วยความจำหลักต่อเมื่อถูกอ้างอิงถึงเท่านั้น ซึ่งมีเหตุผลที่สำคัญอยู่หลายประการคือ

1.1.1 ในทางทฤษฎี เราไม่อาจคาดเดาพฤติกรรมหรือเส้นทางของการดำเนินโปรแกรมได้อย่างเฉพาะเจาะจง ดังนั้นการพยายามบรรจุคำสั่งหรือข้อมูลต่าง ๆ ที่คาดว่าจะได้ใช้นั้น อาจเป็นการบรรจุผิดคำสั่งหรือข้อมูลก็เป็นได้

1.1.2 การบรรจุคำสั่งหรือข้อมูลต่าง ๆ ลงในหน่วยความจำรับรองได้ว่าจะถูกเรียกใช้งานแน่นอน

1.1.3 ประหยัดเวลาและค่าใช้จ่ายในแง่การบรรจุเมื่อถูกเรียกใช้งานมีน้อยที่สุด

1.2 Anticipate Fetch Strategy คือ วิธีการคาดเดาว่าคำสั่งหรือข้อมูลใดจะถูกบรรจุเข้าสู่หน่วยความจำหลักก่อนที่จะถูกใช้งานจริง โดยอาศัยพิจารณาจากพฤติกรรมของการ

ดำเนินครั้งก่อน ๆ และถ้าบางครั้งเกิดการคาดเดาที่ผิดพลาดอาจจะทำให้คำสั่งหรือข้อมูลที่ถูกบรรจุเข้าไปล่วงหน้าไม่ได้ถูกใช้งานจริงก็จะทำให้เสียเวลา

2. กลยุทธ์การวาง (Placement Strategy) เป็นกลยุทธ์ที่ใช้ในการจัดวางคำสั่งหรือข้อมูลใหม่เข้าสู่หน่วยความจำหลักบนพื้นที่ว่างในหน่วยความจำที่เรียกว่า “โฮล (Hold)” เพื่อให้กับชุดคำสั่งหรือข้อมูลต่างๆ ไว้ให้เหมาะสมกับขนาดคำสั่งหรือข้อมูลนั้น

3. กลยุทธ์การแทนที่ (Replacement Strategy) โดยระบบปฏิบัติการจะเป็นส่วนที่ช่วยในการตัดสินใจว่าจะเลือกวิธีใดวิธีหนึ่งในการแทนที่คำสั่งข้อมูลลงบนหน่วยความจำหลัก (Main Memory) ให้เหมาะสมที่สุด ซึ่งมีทั้งหมด 5 วิธีดังนี้

3.1 วิธีสุ่ม (Random) คือ การสุ่มหาพื้นที่ว่างในหน่วยความจำ โดยคำสั่งหรือข้อมูลมีโอกาสถูกเลือกมาใช้งานเท่ากัน

3.2 วิธีมาก่อนได้ใช้ก่อน (First–in, First–out : FIFO) คือ คำสั่งหรือข้อมูลใดถูกบรรจุเข้าไปใช้งานหน่วยความจำหลักก่อนจะถูกเลือกออกไปก่อน

3.3 NFU (Not Frequency Use) คือ การเลือกคำสั่งหรือข้อมูลที่ถูกใช้งานน้อยที่สุดออกไปก่อนเนื่องจากคำสั่งหรือข้อมูลกลุ่มนี้ถูกใช้งานน้อยโอกาสจะถูกบรรจุเข้ามาใช้งานอีก มีโอกาศน้อยตาม ไปด้วย ดังนั้นจึงควรนำออกจากหน่วยความจำหลัก

3.4 LRU (Least Recently Use) คือ วิธีที่เก็บเวลาการใช้งานหน่วยความจำครั้งล่าสุด โดยคำสั่งหรือข้อมูลที่ไม่ได้ถูกนำมาใช้งานนานที่สุดจะถูกเลือกออกไปก่อน

3.5 NRU (Not Recently Use) คือ วิธีนี้แต่ละคำสั่งหรือข้อมูลจะเพิ่มบิตข้อมูลที่เกี่ยวข้อง 2 บิตด้วยกันคือ บิตอ้างอิง (Reference Bit) และบิตแก้ไข (Modify Bit) ซึ่งจะมีค่าเป็น 0 เมื่อมีการเข้าถึงคำสั่งหรือข้อมูลและจะมีค่าเป็น 1 เมื่อมีการแก้ไข

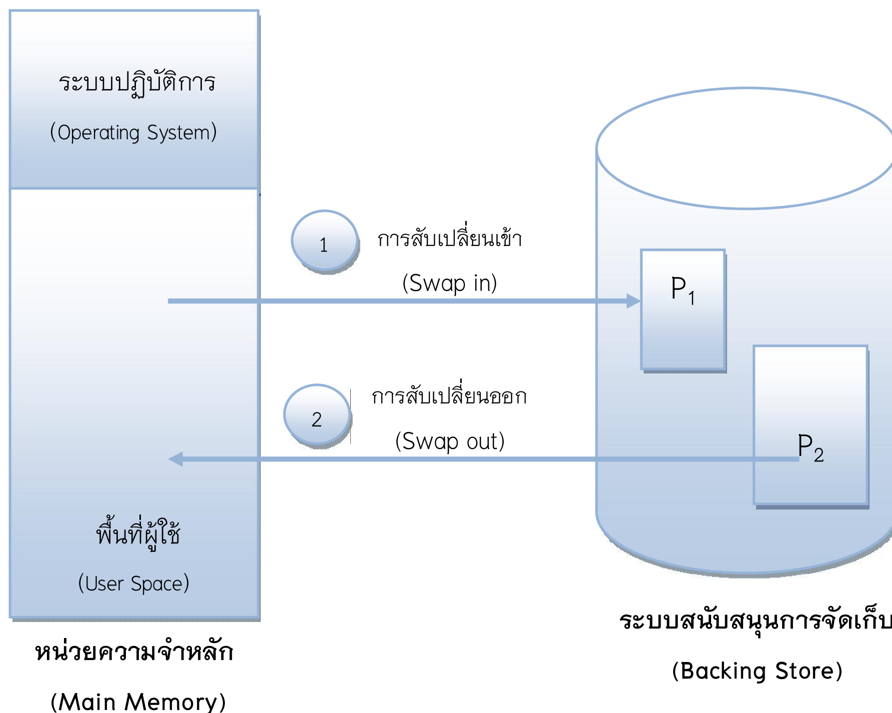
2.8 การสับเปลี่ยน (Swapping)

การสับเปลี่ยน (Swapping) เป็นวิธีที่ในการสลับกระบวนการที่ต้องการจะเข้าหรือออกเพื่อประมวลผลในหน่วยความจำ โดยหลักการจะต้องสลับกระบวนการเก่าออกมาก่อน แล้วจึงนำเข้ากระบวนการใหม่เข้าไปใช้งานหน่วยความจำโดยนำเข้าไปเก็บไว้ในระบบสนับสนุนการจัดเก็บ (Backing Store) แสดงได้ดังภาพที่ 2.6 โดยระบบปฏิบัติการจะเป็นส่วนจัดการเหตุการณ์ในการสลับกระบวนการเข้า–ออก จากหน่วยความจำดังนี้

1. กระบวนการ (Process) ร้องขอการใช้งานอุปกรณ์ I/O
2. กระบวนการ (Process) ดำเนินการเสร็จสิ้น

3. กระบวนการ (Process) หมดเวลาการทำงาน (Quantum Time)

ดังนั้นหลักการสำคัญในการสับเปลี่ยน (Swapping) จะต้องคำนึงถึงระยะเวลาในการเคลื่อนย้าย (Transfer) กระบวนการซึ่งเป็นสัดส่วนโดยตรงกับจำนวนขนาดหรือพื้นที่ในหน่วยความจำที่มีอยู่ด้วย



ภาพที่ 2.6 แสดงการสับเปลี่ยนระหว่าง 2 กระบวนการ ในการใช้งานพื้นที่จานบันทึก (disk) ในการจัดเก็บ (Swapping of two processes using a disk as a backing store)

จากภาพที่ 2.6 อธิบายการควบคุมการสับเปลี่ยน (Swapping) ของระบบปฏิบัติการ โดยใช้อัลกอริทึมการกำหนดตารางลำดับความสำคัญ (Priority-base Scheduling Algorithms) ถ้ากระบวนการ P_1 มีลำดับความสำคัญสูงกว่า (Higher-priority process) เข้ามาถึงและต้องการจะใช้บริการหน่วยความจำ ตัวจัดการหน่วยความจำสามารถที่จะทำการสลับเอากระบวนการ P_2 ที่มีลำดับความสำคัญต่ำกว่า (Lower-priority process) ออกไปก่อน (Swap out) และสลับกระบวนการ P_1 (Swap in) ซึ่งมีลำดับความสำคัญสูงกว่า (Higher-priority process) เข้ามาทำงานให้แล้วเสร็จก่อน กระบวนการ P_2 จึงจะสลับกลับมา

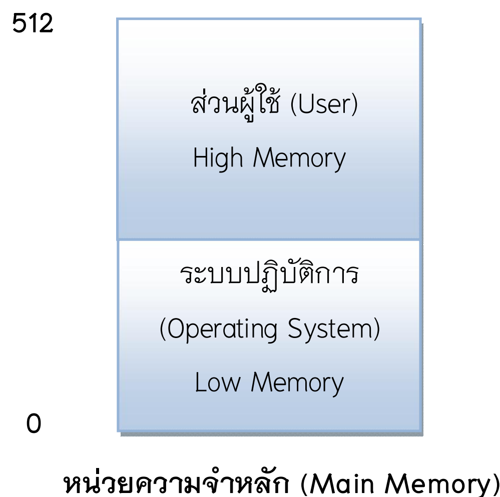
(Swap back) เพื่อทำงานในหน่วยความจำต่อไปได้ โดยบางครั้งอาจจะเรียกวิธีการสลับแบบนี้ว่าการหมุนเข้า (Roll in) หรือหมุนออก (Roll out)

2.9 การจัดสรรหน่วยความจำที่ต่อเนื่องกัน (Continues Memory Allocation)

หน่วยความจำหลักต้องจำเป็นจะต้องอำนวยความสะดวกให้กับระบบปฏิบัติการและความหลากของผู้ใช้งานกระบวนการ ดังนั้นในแต่ละกระบวนการ (Process) จึงมีความต้องการใช้พื้นที่ในหน่วยความจำอย่างต่อเนื่อง จึงเป็นหน้าที่ของระบบปฏิบัติการในการจัดสรรพื้นที่ว่างอย่างต่อเนื่องภายในหน่วยความจำให้เกิดประสิทธิภาพสูงสุด ซึ่งโดยทั่วไปแล้วหน่วยความจำหลัก (Main Memory) จะถูกแบ่งออกเป็น 2 ส่วน เพื่อให้ง่ายกับการจัดการ แบ่งออกเป็น

ส่วนที่ 1 หน่วยความจำระดับบน (High Memory) ซึ่งเป็นส่วนที่ใช้ติดต่อกับส่วนของผู้ใช้ (User)

ส่วนที่ 2 หน่วยความจำระดับล่าง (Low Memory) ซึ่งเป็นส่วนของระบบปฏิบัติการ (Operating System) แสดงได้ดังภาพที่ 2.7

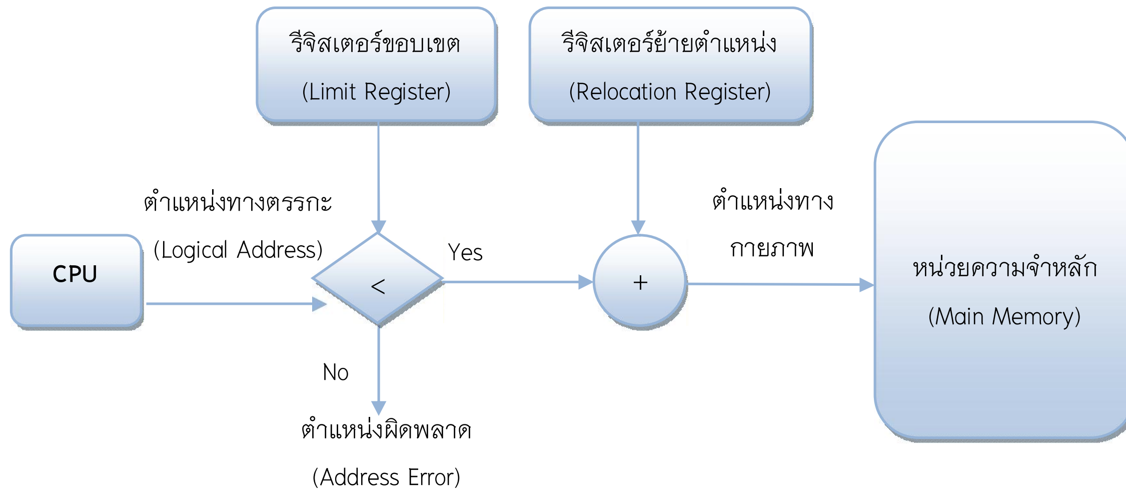


ภาพที่ 2.7 แสดงการแบ่งระดับของหน่วยความจำหลัก (The Level of Main Memory)

นอกจากนี้การจัดสรรพื้นที่ในหน่วยความจำหลัก (Main Memory) เกี่ยวข้องกับการป้องกัน ซึ่งจะต้องพิจารณาจากผู้ใช้งานกระบวนการ (User Processes) และผู้ที่เกี่ยวข้องกับการใช้งานกระบวนการอื่นด้วย โดยมีการตรวจสอบค่าตำแหน่งในรีจิสเตอร์ (Register) ต่างๆ เพื่อป้องกันการแก้ไขคำสั่งหรือข้อมูลในส่วนของระบบปฏิบัติการ (Operating System) และในส่วนของผู้ใช้ (User) ที่กำลังทำงานอยู่ เช่น

1. รีจิสเตอร์ย้ายตำแหน่ง (Relocation Register) เป็นรีจิสเตอร์ที่ภายในบรรจุค่าที่เล็กที่สุดของตำแหน่งทางกายภาพ (Smallest Physical)

2. รีจิสเตอร์ขอบเขต (Limit Register) เป็นรีจิสเตอร์ที่ภายในบรรจุขนาดหรือช่วงของตำแหน่งทางตรรกะ (Logical Address) เช่น รีจิสเตอร์ย้ายตำแหน่ง (Relocation Register) = 100040 และรีจิสเตอร์ขอบเขต (Limit Register) = 74600 เป็นต้น โดยค่าตำแหน่งทางตรรกะ (Logical Address) ต้องน้อยกว่ารีจิสเตอร์ขอบเขต (Limit Register) เสมอ โดยหน่วยจัดการหน่วยความจำ (Main Memory Unit: MMU) จะทำการจับคู่ (Map) ตำแหน่งทางตรรกะ (Logical Address) แบบพลวัต (Dynamically) โดยการเพิ่มค่าลงไปในรีจิสเตอร์ย้ายตำแหน่ง (Relocation Register) แล้วส่งค่าตำแหน่งไปยังหน่วยความจำ (Memory) อีกที แสดงได้ดังภาพที่ 2.8



รูปที่ 2.8 แสดงภาพฮาร์ดแวร์ที่สนับสนุนการทำงานของรีจิสเตอร์ย้ายตำแหน่งรีจิสเตอร์ขอบเขต (Hardware Support for relocation and limit registers)

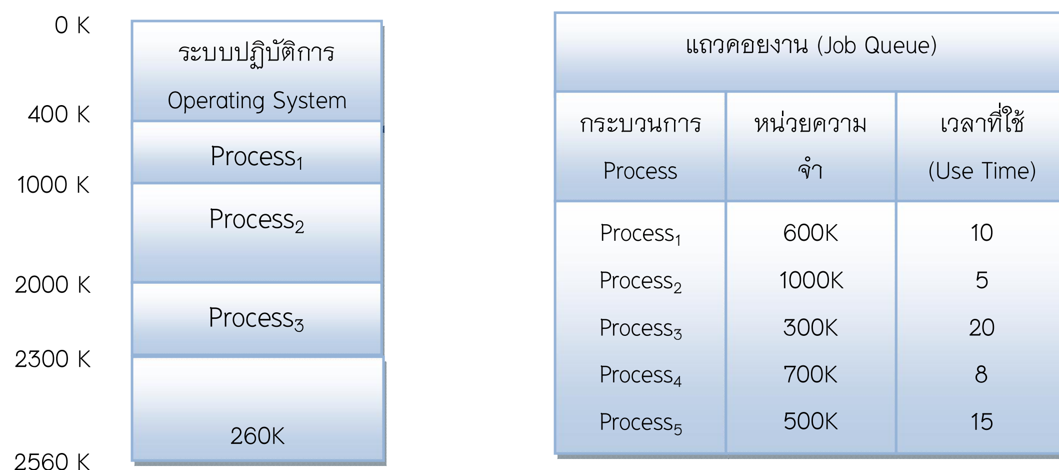
2.10 การจัดสรรพื้นที่ในหน่วยความจำ แบ่งออกเป็น 2 รูปแบบ ดังนี้

1. การจัดสรรหน่วยความจำแบบพื้นที่เดียว (Single-partition Allocation) วิธีนี้หน่วยความจำจะไม่ถูกแบ่งพื้นที่ โดยกระบวนการในส่วนของระบบปฏิบัติการจะอยู่ในส่วนหน่วยความจำระดับล่าง (Low Memory) และส่วนของกระบวนการของผู้ใช้อยู่ในส่วนหน่วยความจำระดับบน (High Memory) โดยเกี่ยวข้องกับรีจิสเตอร์ย้ายตำแหน่ง (Relocation Register) ในการแยกกระบวนการ (Process) ในส่วนของผู้ใช้ออกจากส่วนของระบบปฏิบัติการ และรีจิสเตอร์ขอบเขต (Limit Register) เป็นรีจิสเตอร์ที่ใช้ระบุขนาดของค่าตำแหน่งทางตรรกะ (Logical Address) โดยค่าตำแหน่งทางตรรกะต้องน้อยกว่ารีจิสเตอร์ขอบเขตเสมอ

2.การจัดสรรหน่วยความจำแบบหลายพื้นที่ (Multiple-partition Allocation) วิธีนี้หน่วยความจำจะถูกแบ่งตามจำนวนกระบวนการ (Process) โดยมีวิธีการจัดสรรพื้นที่ 2 รูปแบบ ดังนี้

2.1 การแบ่งแบบคงที่ (Fixed Partition) โดยการแบ่งพื้นที่หน่วยความจำออกเป็นหลาย ๆ พาร์ทิชั่น แต่ละพาร์ทิชั่นมีขนาดเท่ากันและบรรจุกระบวนการอยู่ภายในเพียงหนึ่งกระบวนการเท่านั้น โดยที่จำนวนพาร์ทิชั่นถูกกำหนดโดยจำนวนกระบวนการที่มีอยู่ในหน่วยความจำ ข้อเสียของวิธีนี้คือ หากกระบวนการที่บรรจุอยู่ในหน่วยความจำมีขนาดเล็กกว่าพาร์ทิชั่นที่กำหนดจะทำให้เหลือพื้นที่ในหน่วยความจำ (พาร์ทิชั่นมีพื้นที่ว่างเหลือ) เรียกพื้นที่ว่างที่เหลือนี้ว่า **“Internal Fragmentation”** กรณีที่ขนาดของกระบวนการมีขนาดใหญ่กว่าพาร์ทิชั่นที่กำหนดก็จะไม่สามารถนำกระบวนการนั้นเข้าไปใช้งานพื้นที่ในหน่วยความจำหลักได้

2.2 การแบ่งแบบพลวัต (Dynamic Partition) เป็นการแบ่งพื้นที่ตามขนาดของกระบวนการ โดยการใช้พื้นที่ว่างทางกายภาพที่เรียกว่า โฮล (Hold) เมื่อมีกระบวนการต้องการใช้งานพื้นที่ในหน่วยความจำ ระบบปฏิบัติการจะมีหน้าที่ในค้นหาพื้นที่โฮล (Hold) ที่มีขนาดใหญ่เท่ากับกระบวนการ บรรจุกระบวนการนั้นเข้าไปใช้งานหน่วยความจำ พร้อมกับเก็บข้อมูลของกระบวนการอื่นที่ยังไม่ได้นำเข้าไปใช้งานหน่วยความจำและพื้นที่ว่าง (Free Partition Hold) ที่ยังไม่ได้ถูกจัดสรรให้กับกระบวนการใดๆ ด้วยแถวคอยการทำงาน (Job Queue) ให้สัมพันธ์กับเวลาที่ใช้งานจริง (Use time) ของแต่ละกระบวนการด้วย แสดงได้ดังภาพที่ 2.9



แถวคอยงาน (Job Queue)		
กระบวนการ Process	หน่วยความจำ	เวลาที่ใช้ (Use Time)
$Process_1$	600K	10
$Process_2$	1000K	5
$Process_3$	300K	20
$Process_4$	700K	8
$Process_5$	500K	15

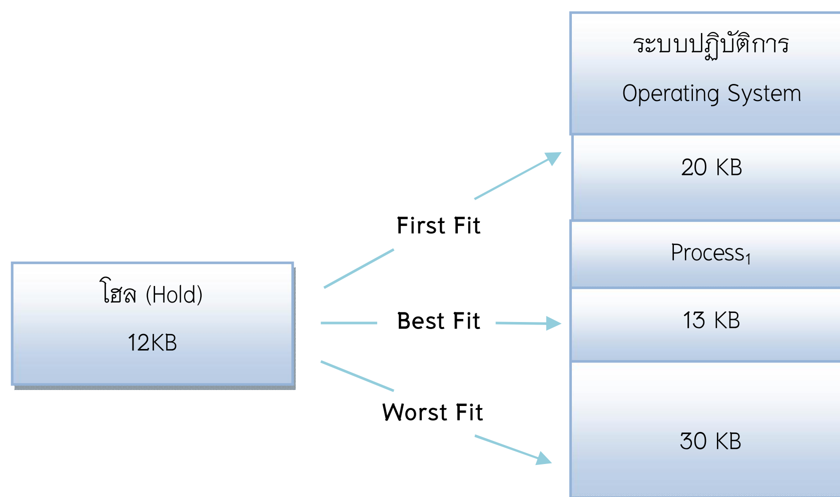
ภาพที่ 2.9 แสดงการแบ่งหน่วยความจำแบบพลวัต (Dynamic Partition)

ปัญหาการจัดสรรพื้นที่หน่วยความจำแบบพลวัต (Dynamic Storage-allocation Problem) ที่พบบ่อยคือการคืนพื้นที่หน่วยความจำให้กับระบบเมื่อกระบวนการได้ทำงานเสร็จแล้วทำให้เกิดพื้นที่ว่างในหน่วยความจำ ซึ่งเรียกพื้นที่ว่างนี้ว่า **"โฮล (Hold)"** ดังนั้นเราจึงใช้วิธีการวาง (Placement) เพื่อใช้แก้ปัญหาดังกล่าวซึ่งจะทำให้การจัดสรรพื้นที่ว่างในหน่วยความจำให้เกิดประโยชน์สูงสุด แสดงได้ดังภาพที่ 2.10 ดังนี้

1. วิธีเลือกพื้นที่ว่างแรกที่พบ (First fit) เป็นการเลือกโฮลตัวแรก (First Hold) ที่พบก่อนและมีขนาดใหญ่เพียงพอกับกระบวนการที่จะนำเข้าไปวางในหน่วยความจำ ซึ่งเป็นวิธีที่ใช้เวลาน้อยที่สุด

2. วิธีเลือกพื้นที่ว่างเหมาะสมที่สุด (Best fit) เป็นการเลือกโฮลที่มีขนาดเล็กที่สุด (smallest Hold) เพื่อให้เหมาะสมที่สุดกับ กระบวนการที่จะนำเข้าไปวางในหน่วยความจำ ซึ่งต้องทำการค้นหาทั้งรายการที่เก็บขนาดของโฮล (Hold) ที่ว่าง ซึ่งเป็นวิธีจัดสรรพื้นที่ว่างในหน่วยความจำให้เกิดประโยชน์สูงสุดเพราะเหลือพื้นที่ว่างโฮลน้อยที่สุด (smallest leftover hold) แต่ก็เสียเวลาการค้นหา

3. วิธีเลือกพื้นที่ว่างที่มีขนาดใหญ่ที่สุด (Worst fit) เป็นการเลือกโฮลขนาดที่ใหญ่ที่สุด (Largest Hold) เพื่อให้เหมาะสมที่สุดกับกระบวนการที่จะนำเข้าไปวางในหน่วยความจำ ซึ่งต้องทำการค้นหาทั้งรายการที่เก็บขนาดของโฮล (Hold) ที่ว่าง วิธีนี้อาจจะจัดสรรเนื้อที่ในหน่วยความจำได้ดีกว่าวิธี Best fit เพราะเหลือพื้นที่ว่างโฮลในหน่วยความจำน้อยกว่า



ภาพที่ 2.10 แสดงวิธีการวาง (Placement) แบบต่างๆ

วิธีการจัดสรรพื้นที่ในหน่วยความจำให้กับแต่ละกระบวนการ อาจเกิดการสูญเปล่าของเนื้อที่ว่างของโฮล (Hold) หรือพื้นที่ว่างที่ไม่สามารถนำไปใช้งานได้ทั้งภายใน (Internal Fragmentation) และภายนอก (External Fragmentation) ที่มีอยู่กระจัดกระจายอยู่เต็มไปหมดในหน่วยความจำหลัก (Main Memory) และมีขนาดใหญ่หรือเล็กเกินไปทำให้ไม่เหมาะสมกับขนาดของกระบวนการที่จะนำไปใช้งานได้ ซึ่งมีวิธีแก้ปัญหาการจัดสรรพื้นที่สูญเปล่าดังกล่าวอยู่ 2 วิธีดังนี้

1. การบีบอัด (Compression) หรือการจัดระเบียบพื้นที่ (Defragmentation) เป็นการจัดพื้นที่ว่างระหว่างกระบวนการใหม่ โดยการสับเปลี่ยนตำแหน่งของกระบวนการต่างๆ ในหน่วยความจำให้เรียงต่อกัน ทำให้มีพื้นที่เพิ่มขึ้นหรือมีขนาดใหญ่ขึ้นซึ่งเหมาะกับการจัดสรรพื้นที่แบบพลวัต (Dynamic Allocation) ระบบปฏิบัติการจะเป็นตัวจัดการ โดยที่กระบวนการต่าง ๆ จะถูกย้ายไปยังตำแหน่งต่าง ๆ โดยอัตโนมัติ

2. อนุญาตให้ตำแหน่งพื้นที่ทางตรรกะ (Logical Address) ของกระบวนการที่ไม่ได้อยู่ติดกัน (Noncontiguous) โดยที่กระบวนการจะถูกจัดสรรไว้บนพื้นที่หน่วยความจำทางกายภาพ (Allocated Physical Memory) ที่ใดก็ได้ที่ทุกตำแหน่งสามารถใช้งานได้

2.11 การแบ่งหน้า (Paging)

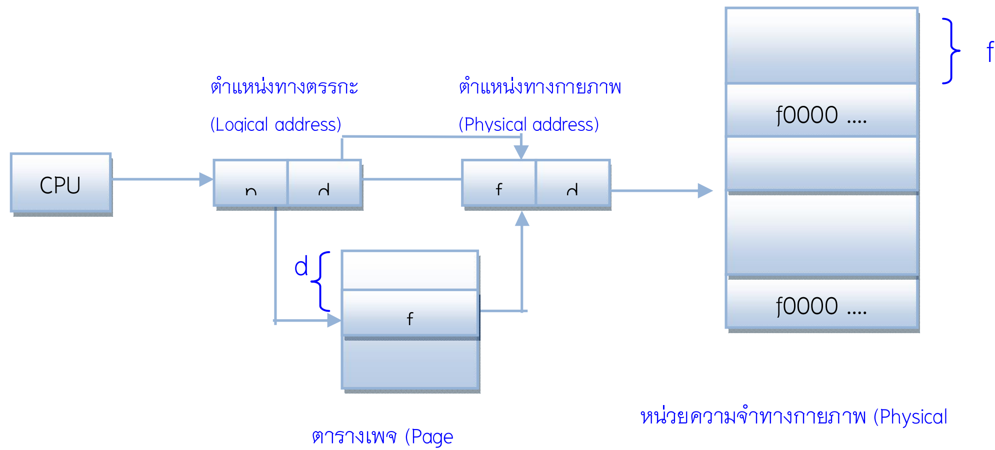
การแบ่งหน้า (Paging) เป็นการจัดสรรพื้นที่ว่างบนหน่วยความจำ โดยทำให้กระบวนการที่อยู่บนหน่วยความจำได้โดยไม่ต้องเรียงต่อเนื่องกันทั้งกระบวนการเป็นการใช้พื้นที่ว่างอยู่กระจัดกระจายโดยไม่สูญเปล่า และไม่จำเป็นจะต้องบีบอัดพื้นที่ว่างในหน่วยความจำก่อน แบ่งออกเป็น 2 ประเภท ดังนี้

1. การจัดสรรหน่วยความจำทางกายภาพ (Paging Model of Physical Memory) เป็นวิธีการแบ่งพื้นที่ให้มีขนาดคงที่ (Fixed–Size Block) เรียกพื้นที่ในส่วนนี้ว่า เฟรม (Frames) โดยที่ขนาดของเพจถูกกำหนดโดยฮาร์ดแวร์ ซึ่งขนาดของเพจเป็นเลขยกกำลัง 2 ที่มีค่าอยู่ระหว่าง 512 ไบต์ (Byte) ถึง 16 (เมกะไบต์) MB ต่อหนึ่งเพจ ขึ้นอยู่กับสถาปัตยกรรมของคอมพิวเตอร์แต่ละชนิด

2. การจัดสรรหน่วยความจำทางตรรกะ (Paging Model of Logical Memory) เป็นวิธีการแบ่งพื้นที่แบบบล็อค (Block) เรียกพื้นที่ส่วนนี้ว่า เพจ (Pages) โดยกำหนดขนาดเพจเท่ากับขนาดเฟรม แสดงได้ดังภาพที่ 2.11 แสดงภาพของฮาร์ดแวร์ซึ่งจะเป็นตัวจัดการแบ่งหน้า ทุกๆ ตำแหน่งจะถูกจัดสรรโดยหน่วยประมวลผลกลาง (CPU) โดยแบ่งออกเป็นสองส่วน คือ

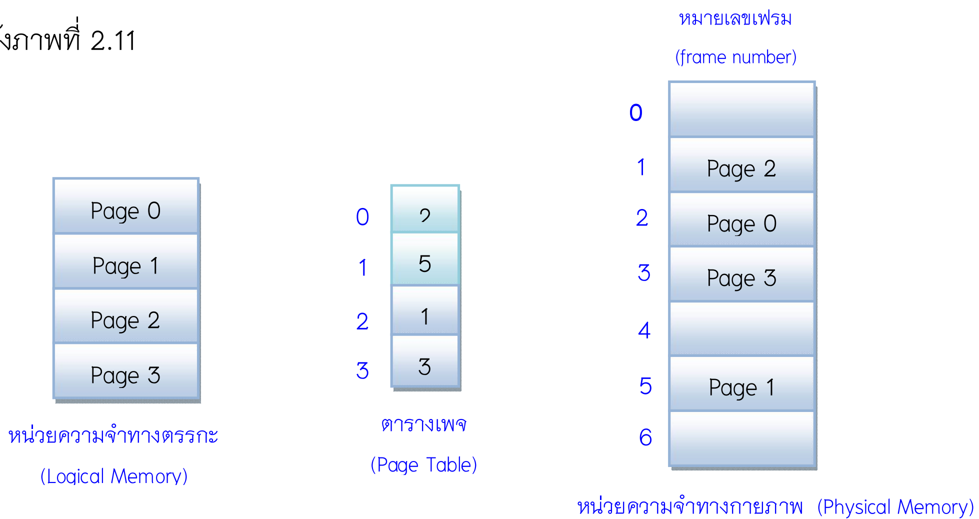
1. หมายเลขเพจ (Page Number: p) โดยหมายเลขเพจจะใช้ดรรชนี (Index) เพื่อชี้ไปยังตารางเพจ (Page Table) ซึ่งภายในบรรจุตำแหน่งเริ่มต้น (Base Address) ของแต่ละเพจในหน่วยความจำทางกายภาพ (Physical Memory)

2. ขอบเขตเพจ (Page Offset: d) คือ ตำแหน่งจริงในหน่วยความจำ ที่นำมารวมกับตำแหน่งเริ่มต้น (Base Address) ที่ได้จากตารางเพจ (Page Table) เพื่อใช้คำนวณหาตำแหน่งของหน่วยความจำทางกายภาพ (Physical Memory) แสดงได้ดังภาพที่ 2.11



ภาพที่ 2.11 แสดงฮาร์ดแวร์การแบ่งตาราง (Hardware Paging)

วิธีการจัดสรรพื้นที่ว่างบนหน่วยความจำ ระบบปฏิบัติการจะมีการแบ่งหน่วยความจำออกเป็น 2 ส่วน คือหน่วยความจำแบบตรรกะและหน่วยความจำแบบกายภาพ แสดงได้ดังภาพที่ 2.11



ภาพที่ 2.12 แสดงรูปแบบการแบ่งเพจในหน่วยความจำแบบตรรกะและหน่วยความจำแบบกายภาพ (Paging model of logical and physical memory)

2.12 ฮาร์ดแวร์กับการสนับสนุนการแบ่งหน้า (Hardware Support)

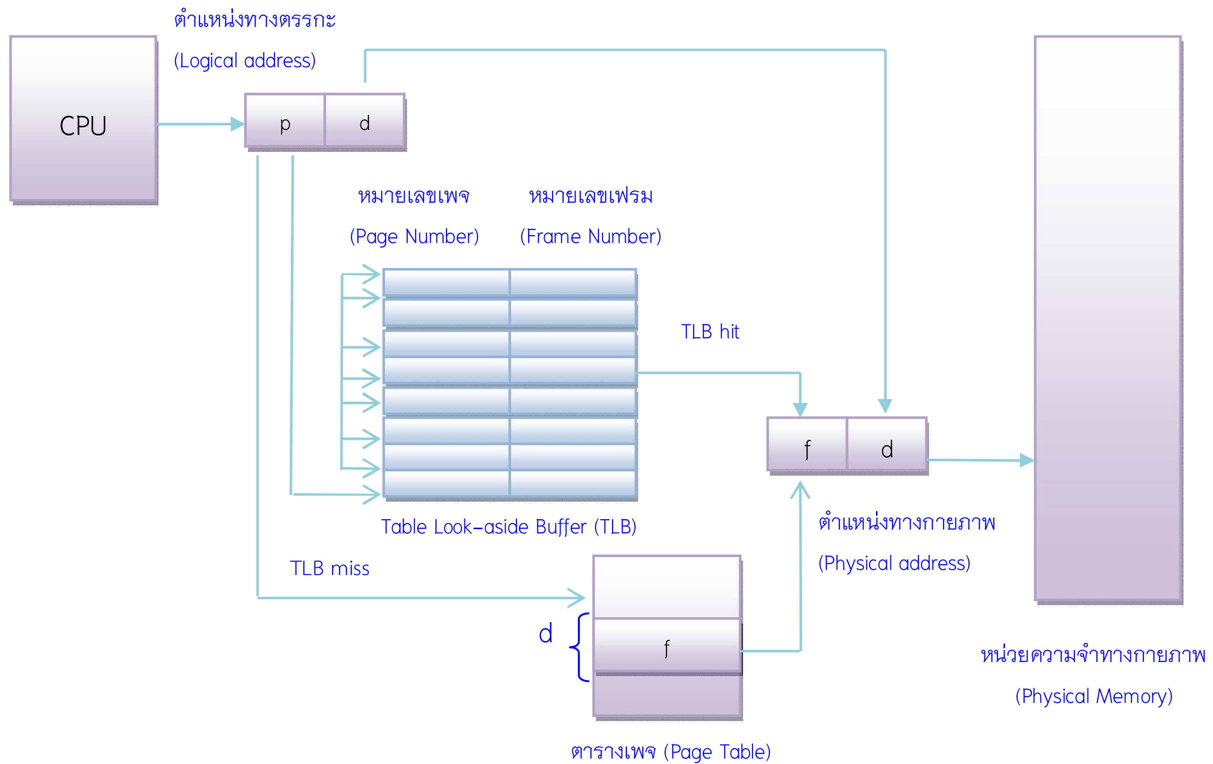
การนำฮาร์ดแวร์กับการสนับสนุนการจัดการตารางเพจ (Page Table) ทำได้หลายทาง ซึ่งการเข้าถึงตำแหน่งในหน่วยความจำแต่ละครั้ง จะต้องมีการอ่านข้อมูลจากหน่วยความจำทางกายภาพ (Physical Memory) ถึงสองครั้ง

ครั้งที่หนึ่ง จากตารางเพจ (Page Table)

ครั้งที่สอง จากตำแหน่งข้อมูลทางกายภาพ (Physical Address)

ทำให้เสียเวลาและการเข้าถึงหน่วยความจำล่าช้า ดังนั้นวิธีมาตรฐานที่ใช้แก้ปัญหาจำเป็นต้องใช้ฮาร์ดแวร์ขนาดเล็กที่มีความเร็ว (Hardware Cache) ในการอ่านและจัดเก็บข้อมูลสูง (Fast–Lookup) ซึ่งเรียกฮาร์ดแวร์ชนิดนี้ว่า **“Translation Look–aside Buffer (TLB)”** ซึ่งข้อมูลจะถูกอ่านจากหน่วยความจำทางกายภาพ (Physical Memory) เพียงครั้งเดียว แล้วจัดเก็บลง TLB ซึ่งเป็นหน่วยความจำที่มีความเร็วสูง (High Speed Memory) ซึ่งประกอบไปด้วยสองส่วนด้วยกัน คือ ส่วนที่หนึ่งเก็บกุญแจ (Key) หรือแท็กซ์ (Tag) ส่วนที่สองเก็บค่า (Value) หากต้องการอ่านข้อมูลจากหน่วยความจำก็จะทำการเปรียบเทียบกุญแจว่าตรงกันหรือไม่ ถ้าพบรายการที่ค้นหาที่สอดคล้องกับค่าของฟิลด์ (Value Field) ก็จะส่งคืนค่ากับไปให้ ซึ่งประสิทธิภาพของการค้นหาจะรวดเร็ว แต่ฮาร์ดแวร์ชนิดนี้จะมีราคาแพง และค่าของจำนวนข้อมูลที่เข้ามาอยู่ใน TLB จะมีขนาดไม่ใหญ่มากนักมักมีค่าอยู่ระหว่าง 64 ถึง 1024 ตัว

จากภาพที่ 2.13 เป็นการอ้างถึงตำแหน่งทางตรรกะ (Logical Address) ซึ่งในการค้นหาจะใช้การส่งหมายเลขเพจ (Page Number) ที่ต้องการค้นหาเข้าสู่ TLB และเปรียบเทียบกับหมายเลขเฟรม (Frame Number) ที่สัมพันธ์กับหมายเลขเพจ (Page Numbers) ทุกตัวพร้อมๆ กัน ซึ่งการค้นหาตำแหน่งใน TLB เรียกวิธีการ นี้ว่า **“ค้นหาแบบคู่ขนาน (Parallel Search)”** กรณีที่ไม่พบหมายเลขเพจที่ต้องการค้นหาใน TLB ก็จะทำ การค้นหาจากตารางเพจ (Page Table) ว่าอยู่เฟรมใดในหน่วยความจำหลักอีกที



ภาพที่ 2.13 แสดงฮาร์ดแวร์กับการสนับสนุนการแบ่งหน้าด้วย TLB
(Paging hardware with Table look-aside buffer)

ในกรณีที่ TLB เต็ม ระบบปฏิบัติการก็จะทำการเลือกหรือลบบางเพจออก (Flushed or Erased) และนำเพจที่ใช้อยู่ล่าสุดไปแทนที่ แต่ในบาง TLB จะไม่อนุญาตให้สามารถทำวิธีการแบบนี้ได้ ซึ่งจะทำให้เกิดการ “Wired Down” เช่น เพจที่เก็บคำสั่งในส่วนของแกนกลาง (Kernel Code) เป็นต้น ดังนั้นเปอร์เซ็นต์ของเวลาในการค้นหาหมายเลขเพจ (Page Numbers) ที่พบใน TLB เราเรียกว่า **“อัตราส่วนที่พบ (Hit Ratio”** เช่น 80% อัตราส่วนที่พบหมายความว่า เราต้องการค้นหาหมายเลขเพจใน TLB แล้วเจอ 80% ของเวลาทั้งหมดหรือถ้าเราใช้เวลา 20 Nanoseconds ในการค้นหา ใน TLB และ 100 Nanoseconds ในการเข้าถึงหน่วยความจำ (Access Memory) ดังนั้นเวลาทั้งหมดที่ใช้ไป (Mapped-Memory Access) จะเท่ากับ 120 Nanoseconds เมื่อหมายเลขเพจ (Page Numbers) อยู่ใน TLB แต่ถ้าไม่พบหมายเลขเพจ (Page Numbers) ใน TLB ซึ่งเสียเวลาไป 20 Nanoseconds และเสียเวลาครั้งแรกไปในการเข้าถึงหน่วยความจำสำหรับการค้นหาในตารางเพจ (Page Table) 100 Nanoseconds และเสียเวลาครั้งที่สองในการค้นหาหมายเลขเฟรม (Frame Numbers) อีกเป็นเวลา 100 Nanoseconds ซึ่งจะ

ใช้เวลาทั้งหมดในการค้นหาเท่ากับ 200 Nanoseconds ดังนั้น วิธีการคำนวณหาประสิทธิภาพของเวลาในการเข้าถึงหน่วยความจำ (Effective Memory–Access Time) สามารถคิดได้จากค่าถั่วเฉลี่ยไปตามเหตุการณ์ต่างที่เกิดขึ้น จึงสรุปได้ว่า

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

สรุปได้ว่า เวลาจะช้าลงไปถึง 40% ในการเข้าถึงหน่วยความจำ
(คือจาก 100 เป็น 140 นาโนวินาที)

ตัวอย่าง ถ้า 98% ของอัตราส่วนที่พบหมายความว่าอย่างไร อธิบายได้ดังนี้

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

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

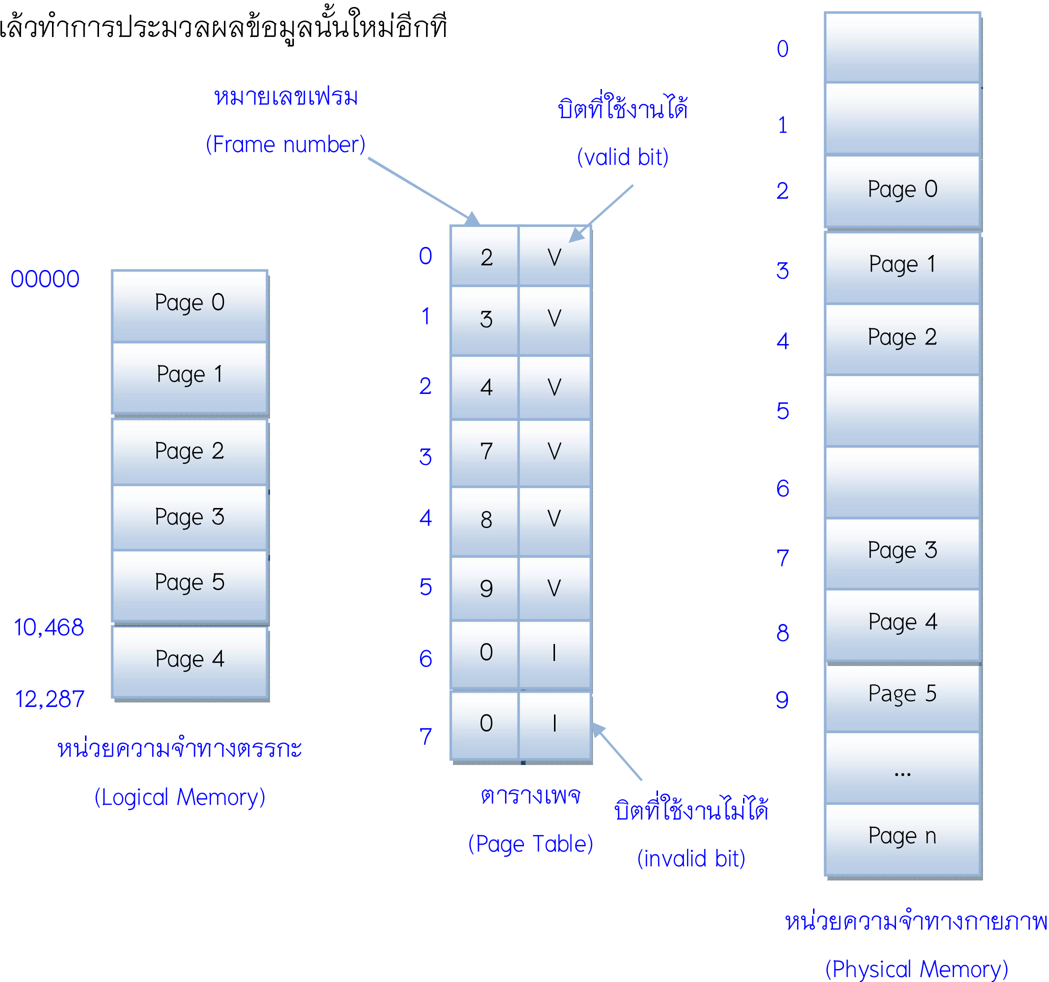
2.13 การป้องกันหน่วยความจำ (Memory Protection)

โดยปกติจำนวนบิต (Bits) ที่เก็บอยู่ในตารางเพจ (Page Table) เราสามารถกำหนดบิตเพื่อใช้การตรวจสอบและกำหนดเพจในการ อ่าน–เขียน (Read–Write) หรืออ่านข้อมูลเท่านั้น (Read–Only) ซึ่งเรียกบิตพิเศษนี้ว่า “กลุ่มบิตป้องกัน (Associating Protection Bits)” ให้กับทุกๆ เฟรม ที่อยู่ในหน่วยความจำ (Main Memory) ซึ่งแบ่งบิตสถานะออกเป็น 2 บิต คือ

1. บิตใช้งานได้ (Valid Bit) เป็นบิตสถานะที่บอกว่าข้อมูลในเพจถูกอ่านเข้าสู่หน่วยความจำทางกายภาพแล้ว และสามารถนำไปใช้งานได้ทันที

2. บิตใช้งานไม่ได้ (Invalid Bit) เป็นบิตสถานะที่บอกว่าข้อมูลในเพจนั้นไม่มีอยู่ในหน่วยความจำทางกายภาพแล้ว (Physical Memory) และไม่สามารถนำไปใช้งานได้ อาจเกิดจากกรณีที่ระบบปฏิบัติการยังไม่ได้อ่านข้อมูลจากเพจเข้าสู่หน่วยความจำหลัก (Main Memory) หรือข้อมูลของเพจที่ต้องการอ่านนั้นถูกสลับ (Swapped) ออกจากหน่วยความจำแล้ว ซึ่งจะทำให้เกิดข้อผิดพลาดขึ้นที่เรียกว่า “การผิดหน้า (Page Fault)” จึงจำเป็นต้องบรรจุเพจข้อมูลเข้าสู่หน่วยความจำก่อนแล้วจึงเปลี่ยนค่าของบิตใช้งานไม่ได้ (Invalid Bit) ให้เป็นบิตใช้งานได้ (Valid Bit) แล้วจึงทำการประมวลผลข้อมูลนั้นใหม่อีกครั้งหนึ่ง แสดงได้ดังภาพที่ 2.14

ภาพที่ 2.14 แสดงตารางเพจของบิตที่ใช้งานได้–บิตที่ใช้งานไม่ได้ (V–I Bit) ของระบบปฏิบัติการ 14 บิต (14–bit) ซึ่งมีพื้นที่ใช้งานได้ตั้งแต่ตำแหน่งที่ 0 ถึง 16383 และคำสั่งเพื่อใช้งานตำแหน่งตั้งแต่เริ่มต้นที่ 0 จนถึง 10468 และมีขนาดเพจ (Page Size) เท่ากับ 2 KB เริ่มตำแหน่งเพจเริ่มต้นที่ 0, 1, 2, 3, 4, 5 และทำการแม็บ (Map) ไปยังตารางเพจ (Page Table) เข้ากับหมายเลขตารางเฟรม (Frame Number) 2, 3, 4, 7, 8, 9 เพื่อบอกสถานการณ์การใช้งานพื้นที่บนหน่วยความจำทางกายภาพ (Physical Memory) ส่วนตารางเพจที่ 6 และ 7 บอกสถานะบิตเป็นบิตใช้งานไม่ได้ (Invalid Bit) ระบบปฏิบัติการมีหน้าที่ในการบรรจุเพจข้อมูลเข้าสู่หน่วยความจำเสร็จจึงเปลี่ยนค่าบิตที่ใช้งานไม่ได้ (Invalid Bit) ให้เป็นบิตที่ใช้งานได้ (Valid Bit) แล้วทำการประมวลผลข้อมูลนั้นใหม่อีกที

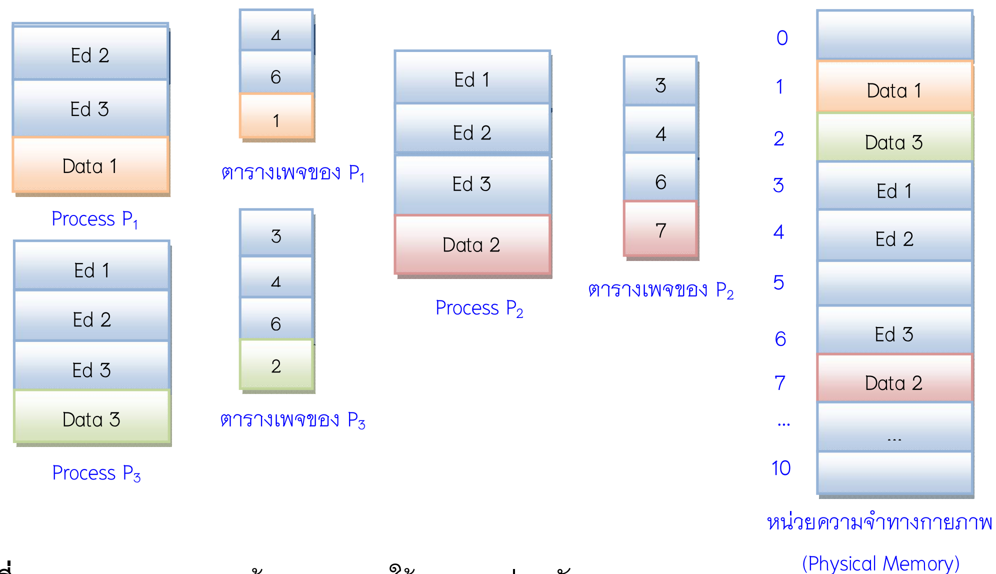


ภาพที่ 2.14 แสดงตารางเพจของบิตที่ใช้งานได้–บิตที่ใช้งานไม่ได้
(Valid (v) or Invalid (i) Bit in a Page Table)

2.14 การใช้เพจร่วมกัน (Shared Pages)

เป็นวิธีการแบ่งปันการใช้เพจร่วมกันในรูปแบบการแบ่งเวลา (Time–Sharing) โดยข้อมูลนั้นจะเป็นข้อมูลที่ใช้อยู่ในลักษณะไม่สามารถเปลี่ยนแปลงได้ (Non–Self–modifying Code) ในขณะที่มีการประมวลผล แต่ใช้งานร่วมกันได้ เรียกข้อมูลชนิดนี้ว่า “Reentrant Code” หรือ “Pure Code” ซึ่งจะเก็บไว้ในตำแหน่งทางตรรกะ (Logical Address) เดียวกัน แต่ละกระบวนการมีข้อมูลเพจ (Own Data Page) เป็นของตัวเอง แสดงได้ดังภาพที่ 2.15

ภาพที่ 2.15 แสดงการใช้เพจร่วมกันระหว่างกระบวนการ ซึ่งประกอบไปด้วย 3 กระบวนการ คือ P_1, P_2, และ P_3 ต้องการใช้งานชุดคำสั่ง Text Editor ร่วมกันใน Page 0, Page1 และ Page3 ที่ถูกจัดเก็บไว้ในหมายเลขเฟรมที่ 3, 4 และ 6 ดังนั้นถ้าระบบปฏิบัติการมีผู้ต้องการใช้งาน 40 คน กำลังใช้งานชุดคำสั่ง Text Editor ซึ่งใช้พื้นที่ 150 KB และใช้เพจสำหรับเก็บข้อมูล 50 KB ดังนั้นหากกระบวนการต่างใช้งานพื้นที่ในหน่วยความจำไม่ร่วมกันจะต้องใช้พื้นที่ในหน่วยความจำเท่ากับ 40 x (150 + 50) = 8000 KB แต่หากกระบวนการทั้งหมดใช้งานเพจร่วมกันจะใช้พื้นที่ในหน่วยความจำเท่ากับ 150 + (40 x 50) = 1250 KB ซึ่งจะช่วยลดการใช้พื้นที่ในหน่วยความจำลงได้ แต่ก็มีข้อเสียคือ จะไม่สามารถลบข้อมูลในเพจที่ใช้งานร่วมกัน (Shared Pages) ได้ ต้องรอจนกว่ากระบวนการอื่นจะใช้งานข้อมูลในเพจที่ใช้งานร่วมกันเสร็จก่อนจึงจะลบข้อมูลในเพจได้ การใช้งานของชุดคำสั่งร่วมกันแบบอื่น เช่น คอมไพเลอร์ (Compilers) การประมวลผลในส่วนไลบรารี (Run–Time Libraries) ระบบฐานข้อมูล (Database System) เป็นต้น



ภาพที่ 2.15 แสดงภาพแวดล้อมของการใช้งานเพจร่วมกัน

(Sharing of Code In a Paging Environment)

2.15 โครงสร้างของตารางเพจ (Memory Protection)

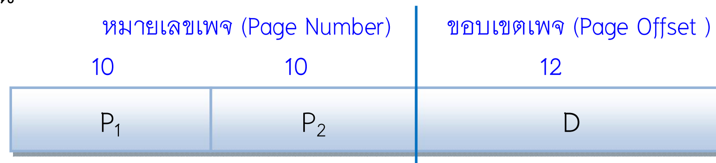
เทคนิคการสร้างตารางเพจแบ่งออกเป็น 3 รูปแบบ ได้แก่

1. โครงสร้างแบบลำดับชั้น (Hierarchical Paging) ระบบคอมพิวเตอร์สมัยใหม่จะสนับสนุนตำแหน่งพื้นที่ทางตรรกะ (Logical–Address Space) ขนาดใหญ่ (2^{32} ถึง 2^{64}) ซึ่งโครงสร้างแบบลำดับชั้นเป็นการแบ่งพื้นที่ทางตรรกะออกเป็นตารางเพจ (Page Table) หลายตาราง แสดงได้ดังภาพที่ 2.16 โดยใช้ขั้นตอนวิธีในการแบ่งเพจแบบ 2 ระดับ (Two–Level Page Table) ได้แก่

1.1 หมายเลขเพจ (Page Number)

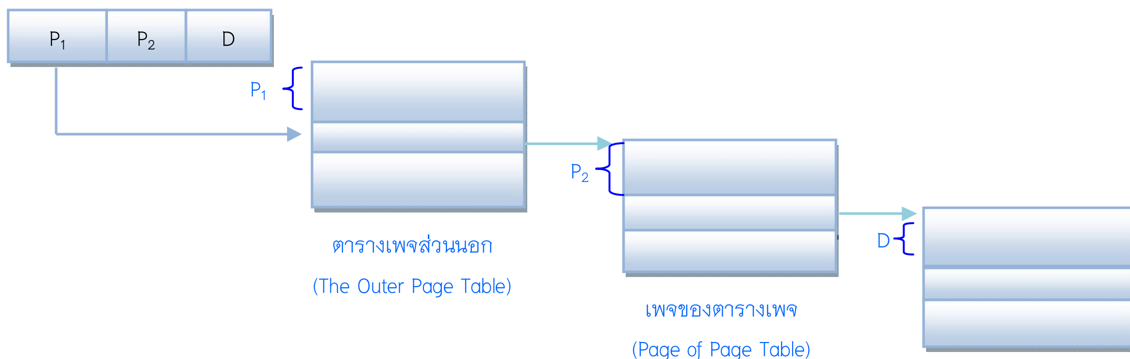
1.2 ขอบเขตเพจ (Page Offset)

ตัวอย่าง ระบบคอมพิวเตอร์ขนาด 32 บิต ประกอบด้วยเพจขนาด (Page Size) 4 KB ตำแหน่งทางตรรกะถูกแบ่งเป็น 2 ส่วน คือ ส่วนแรกขนาด 20 บิต (Bits) เก็บหมายเลขเพจ (Page Number) และส่วนที่สองขนาด 12 บิต (Bits) เก็บขอบเขตเพจ (Page Offset) ซึ่งแสดงโครงสร้างข้อมูลดังนี้



ภาพที่ 2.16 แสดงโครงสร้างตำแหน่งทางตรรกะแบบ 1 ระดับขนาด 32 บิต

ซึ่งประกอบไปด้วย กระบวนการ P_1 เป็นดัชนีที่อ้างอิงไปยังตารางเพจส่วนแรกที่เรียกว่า “ตารางเพจส่วนนอก (The Outer Page Table)” หลังจากนั้นก็จะอ้างอิงไปยังส่วนที่สองที่เรียกว่า “เพจของตารางเพจ (Page of Page Table)” โดยมีกระบวนการ P_2 อยู่ในเพจของตารางเพจ แสดงได้ดังภาพที่ 2.17



ภาพที่ 2.17 แสดงโครงสร้างตำแหน่งทางตรรกะแบบ 2 ระดับขนาด 32 บิต

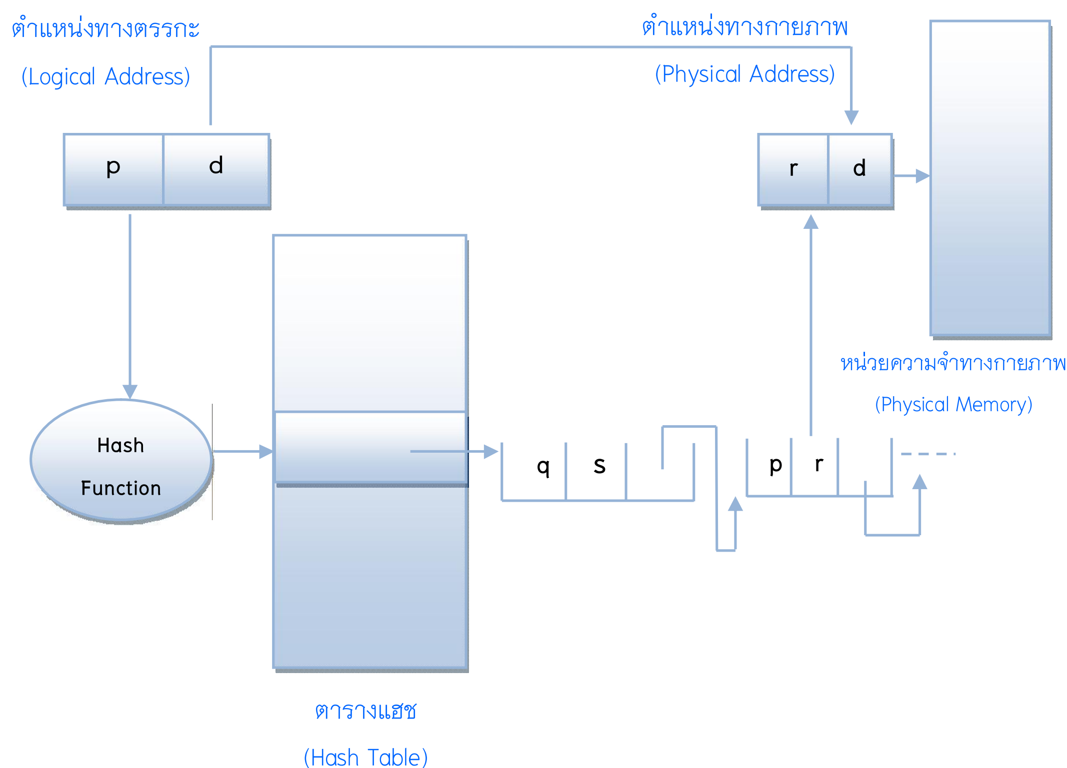
(Address Translation for a Two–Level 32–Bit Paging Architecture)

2. โครงสร้างแบบตารางเฮช (Hash Page Table) ระบบคอมพิวเตอร์ที่จะใช้โครงสร้างแบบนี้ต้องมีตำแหน่งขนาดพื้นที่ (Address Space) มากกว่า 32 บิต โดยค่าของเฮชจะอยู่ภายในหมายเลขเพจเสมือน (Virtual-Page Number) ซึ่งแต่ละหน่วย (Elements) ในตารางเฮชภายในจะมีลิงก์ลิสต์ (Link List) เชื่อมโยงไปยังหน่วยที่อยู่ในพื้นที่เดียวกัน แสดงได้ดังภาพที่ 2.18 โดยข้อมูลในแต่ละหน่วยจะประกอบไปด้วย

2.1 ค่าหมายเลขเพจเสมือน (Virtual-Page Number)

2.2 ค่าดรรชนีที่ชี้ไปยังเฟรมเพจ (Page Frame)

2.3 ค่าของตัวชี้ (Pointer) ที่ชี้ไปยังหน่วยที่เชื่อมโยงในลิงก์ลิสต์ (Link List)



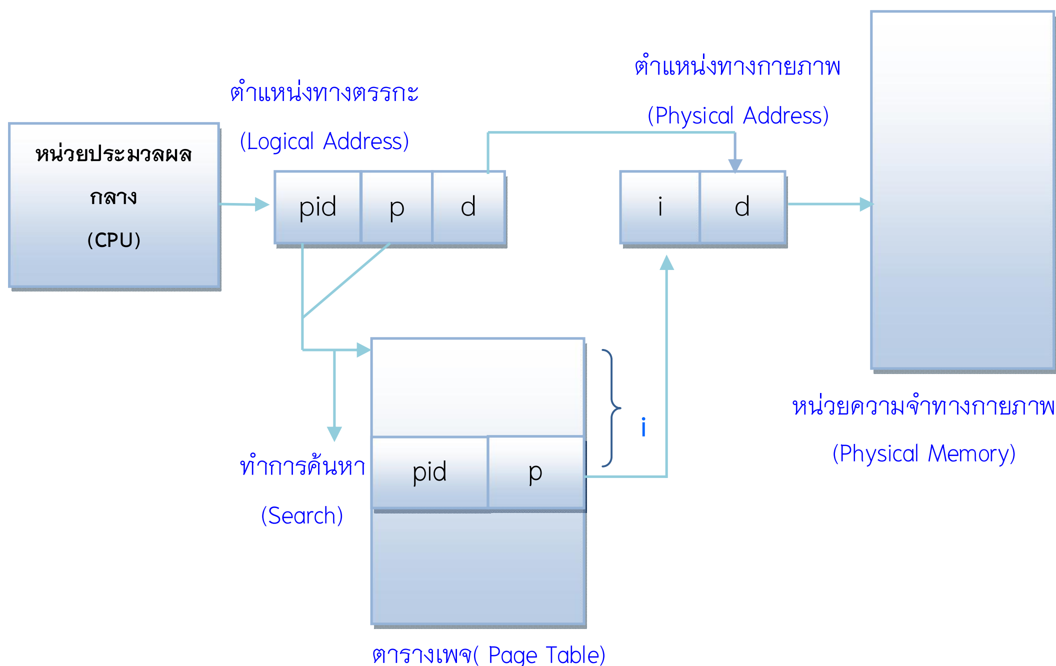
ภาพที่ 2.18 แสดงโครงสร้างแบบตารางแฮช (Hashed Page Table)

จากภาพที่ 2.18 แสดงอัลกอริทึมของหมายเลขเพจเสมือน (Virtual-Page Number) ในตารางเฮช (Hash Table) โดยจะทำการเปรียบเทียบค่าของฟิลด์ (Field) กับตำแหน่งแรกในลิงก์ลิสต์ (Link List) ถ้าเหมือนกัน (match) ก็จะส่งค่าของเฟรมเพจ (Page Frame) ที่ใช้เชื่อมโยงไปยังตำแหน่งทางกายภาพแต่ถ้าไม่เหมือนกัน (no match) ก็จะทำการค้นหาค่าที่เหมือนกันที่อยู่ในค่าหมายเลขเพจเสมือน (Virtual-Page Number) อีกที

3.โครงสร้างเพจแบบผกผัน (Inverted Page Table) ระบบคอมพิวเตอร์ที่จะใช้โครงสร้างแบบนี้ ต้องมีคุณสมบัติกำหนดโดยตารางเพจ (Page Table) จะมีเพียงหนึ่งกระบวนการใช้งานพื้นที่หน่วยความจำ โดยชุดคำสั่งหรือข้อมูลจะประกอบไปด้วยตำแหน่งของเพจเสมือน (Virtual-Address of Page) ที่เก็บอยู่ในหน่วยความจำจริง (Real Memory) แสดงได้ดังภาพที่ 2.19 จะมีเพียงหนึ่งตารางเพจ (One Page Table) เท่านั้นที่อยู่ในหน่วยความจำ โดยแต่ละตำแหน่งของเพจเสมือน (Virtual-Address of Page) ประกอบไปด้วย

<หมายเลขกระบวนการ (process-id), หมายเลขเพจ (page-number), ขอบเขต (offset)>

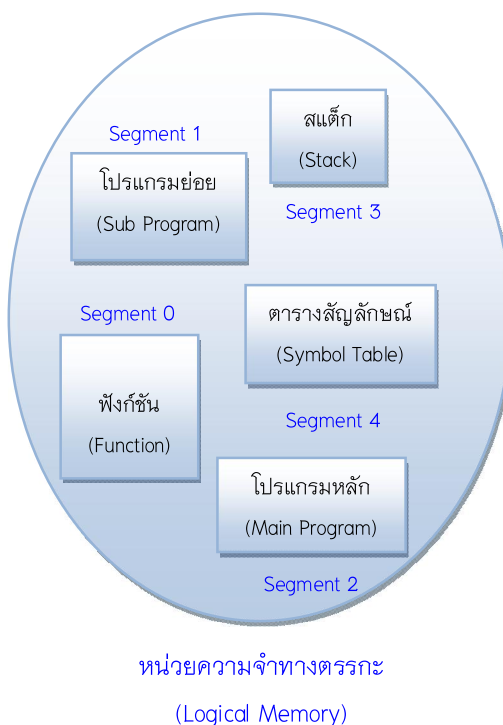
โดยแต่ละตารางเพจแบบผกผัน (Inverted Page Table) จะประกอบด้วย <process-id, page-number> ซึ่งจะทำการค้นหาโดยการจับคู่ตำแหน่งของเพจที่เหมือนกัน (Match) ในตารางเพจ (Page Table) ถ้าพบก็จะทำการ Map ค่าของ entry i กับตำแหน่งทางกายภาพ (Physical Address) เพื่อส่งเข้าไปประมวลผลในหน่วยความจำหลัก แต่ถ้าไม่พบก็จะทำการค้นต่อไปเรื่อยๆ ถึงแม้ว่าวิธีนี้จะใช้งานหน่วยความจำได้อย่างมีประสิทธิภาพ แต่ก็จะต้องเสียเวลาที่จะค้นหาตำแหน่งที่อยู่ในตารางอีกครั้งหนึ่ง กรณีที่มีการอ้างอิงถึงเหตุการณ์ที่เกิดขึ้นก่อนหน้า



ภาพที่ 2.19 แสดงโครงสร้างแบบผกผัน (Inverted Page Table)

2.17 การแบ่งส่วน (Segmentation)

วิธีการนี้เป็นการจัดสรรพื้นที่ในหน่วยความจำหลักออกเป็นส่วนๆ ตามขนาดของชุดคำสั่งหรือโมดูลย่อย เรียกพื้นที่นี้ว่า Segment โดยแต่ละโมดูลจะถูกแบ่งออกเป็นส่วนๆ ที่มีขนาดไม่เท่ากัน เช่น ชุดคำสั่งหลัก โปรแกรมย่อย ฟังก์ชัน ตัวแปร บล็อก สแต็ก ตารางสัญลักษณ์ และอาร์เรย์ ซึ่งแต่ละโมดูลจะมีการทำงานสัมพันธ์กัน แสดงได้ดังภาพที่ 2.20



หน่วยความจำทางตรรกะ
(Logical Memory)

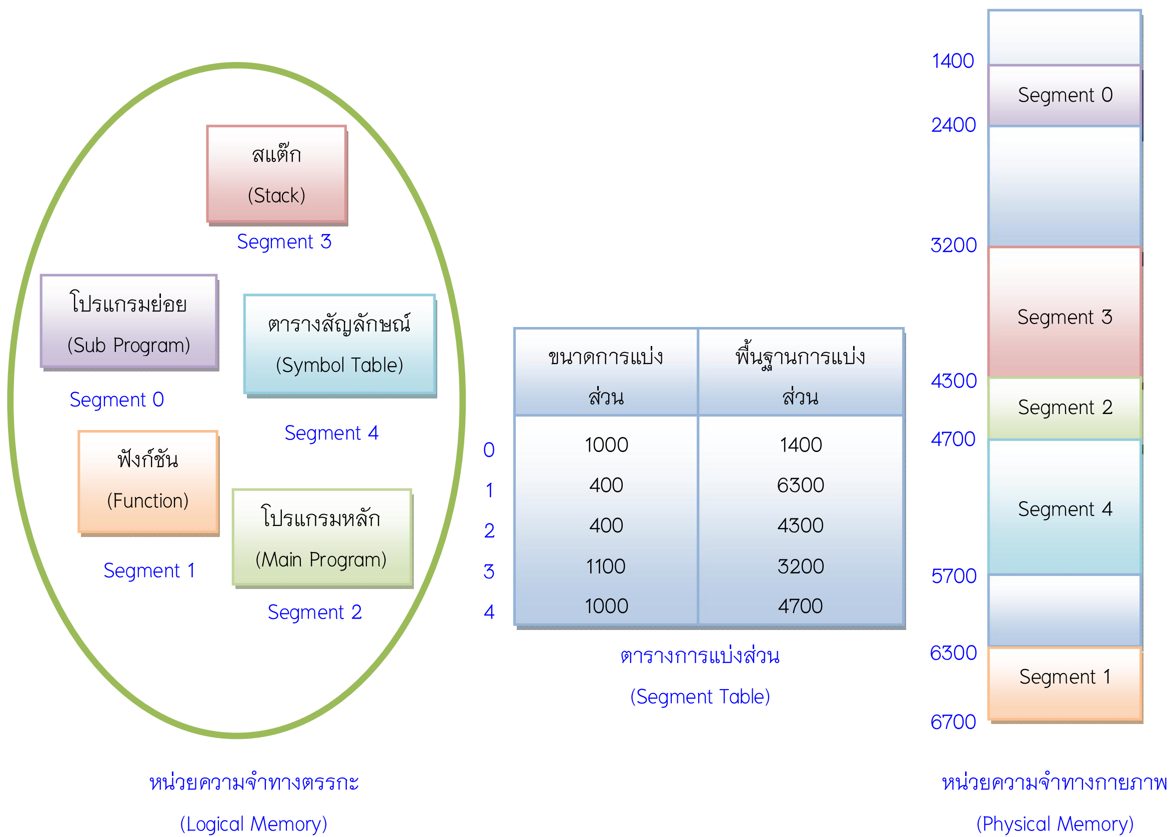
ภาพที่ 2.20 แสดงโครงสร้างแบบแบ่งส่วน (Segmentation)

ตัวอย่าง จากภาพที่ 2.21 กำหนดให้มีการแบ่งส่วน (Segment) ออกเป็น 5 ส่วน (Segment) มีหมายเลขตั้งแต่ 0 ถึง 4 โดยแต่ละส่วน (Segment) ถูกจัดเก็บอยู่ในหน่วยความจำทางกายภาพ (Physical memory) ภายในตารางการแบ่งส่วนมีข้อมูลอยู่ภายใน โดยกำหนดตำแหน่งเริ่มต้นเรียกว่าตำแหน่งฐาน (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

2. กำหนดให้ Segment 3 มีความยาว 1000 ไบต์และตำแหน่งเริ่มต้น (Base of Segment) ที่ 3200 ดังนั้นถ้ามีการอ้างอิงไปยังไบต์ที่ 852 ของ Segment 3 ก็จะทำการจับคู่ (Map) ไปยังตำแหน่งเริ่มต้นบวกกับตำแหน่งที่อ้างอิงถึงจะได้เท่ากับ 3200 + 852 = 4052

3. กำหนดให้ Segment 0 มีความยาว 1000 ไบต์และตำแหน่งเริ่มต้น (Base of Segment) ที่ 1400 ดังนั้นถ้ามีการอ้างอิงไปยังไบต์ที่ 1222 ซึ่งอยู่ที่ตำแหน่ง Segment 0 มีความยาวเพียง 1000 ไบต์ จึงเป็นหน้าที่ของระบบปฏิบัติการในการเลื่อน (Tab) ไปยัง Segment อื่นๆ ที่มีขนาดความยาวไบต์ (Bytes Long) พอดีกับไบต์ที่ต้องการ แสดงได้ดังภาพที่ 2.21



ภาพที่ 2.21 แสดงโครงสร้างตารางการแบ่งส่วน (Segmentation Table)

สรุป

การจัดการหน่วยความจำ (Memory Management) เป็นหน้าที่หนึ่งของระบบปฏิบัติการ ซึ่งมีหน้าที่หลักในการจัดสรรพื้นที่ในการใช้งานให้กับหลายชุดคำสั่ง (Multiprogramming) หรือข้อมูลต่างๆ โดยใช้ฮาร์ดแวร์ (Hardware provided) เข้าไปช่วยสนับสนุนวิธีการเข้าถึงตำแหน่งในหน่วยความจำทั้งในส่วนหน่วยความจำทางตรรกะ (Logical memory) และหน่วยความจำทางกายภาพ (Physical memory) ซึ่งมีหน่วยประมวลผลกลาง (CPU) เป็นตัวประมวลผล (Generated) อีกที

อัลกอริทึมในการจัดการหน่วยความจำ (Memory Management Algorithms) แบ่งออกเป็น 4 แบบ คือ

1. การจัดสรรแบบต่อเนื่อง (Contiguous allocations)
2. การแบ่งหน้า (Paging)
3. การแบ่งส่วน (Segmentation)
4. การผสมผสานระหว่างวิธีแบ่งหน้าและแบ่งส่วน (Combination of Paging and Segmentation) โดยมีกลยุทธ์ในการจัดการหน่วยความจำ (Memory Management Strategy) ให้เลือกใช้ ดังนี้

1. ความสามารถของฮาร์แวร์ (Hardware Support) เป็นพื้นฐานที่ช่วยสนับสนุนการทำงานของกระบวนการการแบ่งหน้า (Paging) และการแบ่งส่วน (Segmentation) โดยโครงสร้างตารางการจับคู่ระหว่างข้อมูลและตำแหน่งของหน่วยความจำหลัก
2. ประสิทธิภาพ (Performance) วิธีที่ใช้จัดการหน่วยความจำหลักแต่ละวิธีจะมีความซับซ้อน และใช้ระยะเวลาในการจับคู่ระหว่างหน่วยความจำทางตรรกะ (Logical memory) และหน่วยความจำทางกายภาพ (Physical memory) ดังนั้น ระบบปฏิบัติการจึงต้องเลือกใช้วิธีที่เหมาะสมและรวดเร็ว
3. การจัดการพื้นที่ว่าง (Fragmentation) ในระบบคอมพิวเตอร์แบบหลายชุดคำสั่ง (Multiprogramming) มีความสามารถในการประมวลผลสูง จะต้องมีการจัดการพื้นที่ว่างที่เหมาะสมกับขนาดของกระบวนการ (Process) เพื่อป้องกันปัญหาที่เกิดขึ้น
4. การย้ายตำแหน่ง (Relocation) การนจัดการหน่วยความจำหลักที่ดีควรสามารถย้ายกระบวนการ (Process) ในหน่วยความจำหลักได้อย่างอัตโนมัติเพื่อแก้ปัญหาการเกิดพิ้นที่ว่าง (Fragmentation) ในหน่วยความจำหลัก

5. การสลับ (Swapping) เป็นกลไกการสลับกระบวนการ (Process) เข้า/ออกจากหน่วยความจำหลัก ควรให้มีการสลับกระบวนการให้น้อยที่สุดภายในหนึ่งหน่วยเวลา

6. การใช้งานร่วมกัน (Sharing) ระบบปฏิบัติการที่ดีควรสามารถจัดสรรโปรแกรมหรือข้อมูลต่างๆ ให้สามารถทำงานร่วมกันได้ เพื่อให้กระบวนการ (Process) จำนวนมากสามารถทำงานร่วมกันได้

7. การป้องกัน (Protection) ระบบปฏิบัติการที่ดีควรมีการป้องกันคำสั่งหรือข้อมูลต่างๆ โครงสร้างข้อกำหนดเงื่อนไขในการอ่านหรือเขียนข้อมูล โดยระบบต้องมีการตรวจสอบคำสั่งหรือข้อมูลต่างๆ ขณะกำลังประมวลผล พร้อมทั้งแสดงข้อผิดพลาดได้

คำถามทบทวน

1. กลยุทธ์ในการจัดการหน่วยความจำ (Memory Strategy) มีกี่วิธีอะไรบ้าง

2. ถ้าต้องการค้นหาหมายเลขเพจใน TLB แล้วเจอ 70% ของเวลาทั้งหมดหรือถ้าเราใช้เวลา 25 Nanoseconds ในการค้นหา ใน TLB และ 120 Nanoseconds ในการเข้าถึงหน่วยความจำ (Access Memory) ดังนั้นเวลาทั้งหมดที่ใช้ไป (Mapped-Memory Access) จะเท่ากับ 140 Nanoseconds เมื่อหมายเลขเพจ (Page Numbers) อยู่ใน TLB แต่ถ้าไม่พบหมายเลขเพจ (Page Numbers) ใน TLB ซึ่งเสียเวลาไป 25 Nanoseconds และเสียเวลาครั้งแรกไปในการเข้าถึงหน่วยความจำสำหรับการค้นหาในตารางเพจ (Page Table) 110 Nanoseconds และเสียเวลาครั้งที่สองในการค้นหาหมายเลขเฟรม (Frame Numbers) อีก 110 Nanoseconds ซึ่งจะใช้เวลาทั้งหมดในการค้นหาเท่ากับ 220 Nanoseconds จงคำนวณหาประสิทธิภาพของเวลาในการเข้าถึงหน่วยความจำ (Effective Memory-Access Time)

3. จงอธิบายอัลกอริทึมในการจัดการหน่วยความจำ (Memory Management Algorithms) ต่อไปนี้

    3.1 การจัดสรรแบบต่อเนื่อง (Contiguous allocations)

    3.2 การแบ่งหน้า (Paging)

    3.3 การแบ่งส่วน (Segmentation)

    3.4 การผสมผสานระหว่างวิธีแบ่งหน้าและแบ่งส่วน (Combination of Paging and Segmentation)

4. กลุ่มบิตป้องกัน(Associating Protection Bits)แบ่งเป็นกี่ประเภทอะไรบ้าง

5. เทคนิคการสร้างตารางเพจแบ่งออกเป็นกี่รูปแบบอะไรบ้าง

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน*. ค้นเมื่อ 27 พฤษภาคม 2556, จาก: http://rirs3.royin.go.th/coinages

ไลบราลี. (2556). *วิกิพีเดีย*. ค้นเมื่อ 27 พฤษภาคม 2556, จาก: http://th.wikipedia.org/wiki

พีรพร หมุนสนิท, สุธี พงศาสุกุลชัย, อัจจิมา เลี้ยงอยู่. (2553). *ระบบปฏิบัติการ: Operating Systems*. กรุงเทพฯ : เคทีพี แอนด์ คอนซัลท์.

พีระพนธ์ โสพัศสถิตย์. (2552). *ระบบปฏิบัติการ*. Operating Systems. กรุงเทพฯ : สำนักพิมพ์จุฬาลงกรณ์มหาวิทยาลัย.

Silberschartz, Galvin, Gangne. (2011). *Operating System Concepts*. 8 th (ed), New York: McGra Hill.