เอกสารประกอบการสอน

วิชา ระบบปฏิบัติการ 2 (Operating Systems 2) รหัส 4121402

บทที่ 2 การจัดการหน่วยความจำ (Memory Management)

หลักสูตรระดับปริญญาตรี

พุทธศักราช 2551 (ปรับปรุง 2554)

โดย

จุฑาวุฒิ จันทรมาลี

สาขาวิทยาการคอมพิวเตอร์

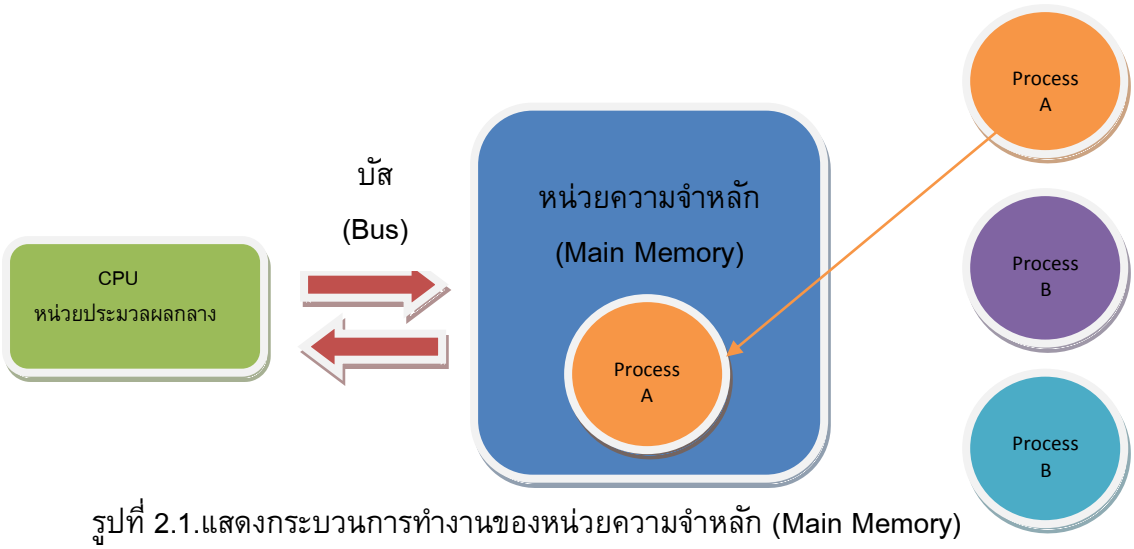
คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏสวนดุสิต

บทที่ 2
การจัดการหน่วยความจำ
(Memory Management)

การจัดการหน่วยความจำ (Memory Management)

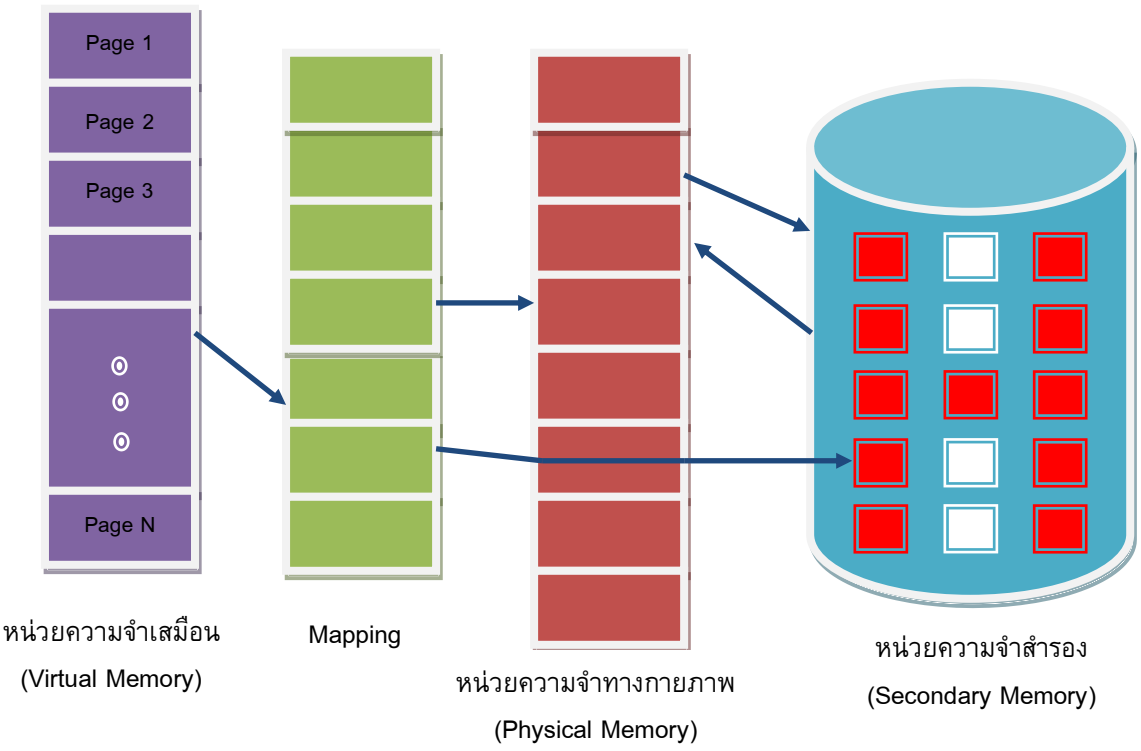
หน่วยความจำเป็นส่วนที่สำคัญที่สุดในระบบคอมพิวเตอร์ ถือเป็นศูนย์กลางให้การดำเนินการด้านต่างๆ ในระบบคอมพิวเตอร์เป็นไปอย่างราบรื่นและมีประสิทธิภาพสูงสุด โดยภายในหน่วยความจำจะมีการทำงานหลายส่วน เช่น การทำงานของโปรแกรมจำนวนมาก ซึ่งต้องมีการแบ่งพื้นที่การใช้งานและวิธีการจัดการด้านต่างๆ ซึ่งหน่วยความจำที่ใช้ในการจัดเก็บแบ่งออกเป็น 2 ส่วน ได้แก่ หน่วยความจำหลัก (Main Memory) และหน่วยความจำเสมือน (Virtual Memory) โดยแต่ละวิธีในการจัดเก็บข้อมูลทั้งสองส่วนมีข้อดีและข้อเสียต่างกันขึ้นอยู่กับซอฟต์แวร์และฮาร์ดแวร์ที่เลือกใช้ว่าสอดคล้องและสนับสนุนการทำงานและวิธีการที่จัดเก็บในหน่วยความจำที่เลือกใช้มากน้อยเพียงใด ดังรูปที่ 2.1

หน่วยความจำหลัก (Main Memory) ประกอบไปด้วยอาเรย์ขนาดใหญ่ (large array) ซึ่งภายในประกอบไปเวิร์ด (words) และไบต์ (bytes) ซึ่งแต่ละที่จะมีเลขตำแหน่ง (address) เป็นของตัวเอง นอกจากนี้หน่วยความจำหลักยังทำหน้าที่เก็บชนิดกระบวนการในการประมวลผลคำสั่ง (a typical instruction-execution cycle) เพื่อให้หน่วยประมวลผลกลาง (Central Processing Unit: CPU) นำไปใช้ในการประมวลแล้วจึงส่งผลลัพธ์ของคำสั่งนั้นๆ กลับมาจัดเก็บกลับไว้ในหน่วยความจำหลักอีกที



รูปที่ 2.1.แสดงกระบวนการทำงานของหน่วยความจำหลัก (Main Memory)

หน่วยความจำเสมือน (Virtual Memory) เป็นเทคนิคที่อนุญาตให้โปรเซส (Process) สามารถประมวลผลได้นอกหน่วยความจำหลักโดยไม่ต้องคำนึงถึงขนาดพื้นที่ใช้ในการประมวลผลว่าเพียงพอกับขนาดของโปรแกรมหรือไม่ นอกจากนี้ยังง่ายต่อการแชร์ไฟล์ (Share files) พื้นที่ว่าง (Address Space) และเพิ่มประสิทธิภาพให้โปรเซสทำงานได้เร็วขึ้น เพราะไม่ต้องคอยตรวจสอบขนาดของหน่วยความจำทางกายภาพ (Physical Memory)



รูปที่ 2.2 แสดงความสัมพันธ์ระหว่างหน่วยความเสมือน (Virtual Memory) และหน่วยความจำทางกายภาพ (Physical Memory)

การเชื่อมโยงตำแหน่ง (Address Binding)

โดยทั่วไปโปรเซสที่จะถูกนำไปประมวลผลจะขึ้นอยู่กับการจัดการหน่วยความจำที่เลือกใช้ โปรเซสอาจจะถูกเคลื่อนย้ายกลับไปกลับมาระหว่างดิสก์และหน่วยความจำก่อนที่มันจะถูกประมวลผล โดยระบบปฏิบัติการจะมีการเก็บโปรเซสที่รอประมวลผลตามลำดับคิว (input queue) ของโปรเซสที่จะนำเข้ามาประมวลผลในหน่วยความจำร่วมทั้งยังเชื่อมโยงกับค่าเริ่มต้นของตำแหน่ง ระบบคอมพิวเตอร์มักเริ่มต้นค่าตำแหน่งเชื่อมโยงที่ค่า 00000 อีกยังแบ่งค่าที่เชื่อมโยงตำแหน่ง ได้เป็น ค่าจริง (Absolute address) ของโปรเซสที่อยู่ในหน่วยความจำ และค่าตำแหน่งที่สัมพันธ์ (Relative address) หรือชุดคำสั่งต่างๆ ที่ได้รับหลังจากการคอมไพล์

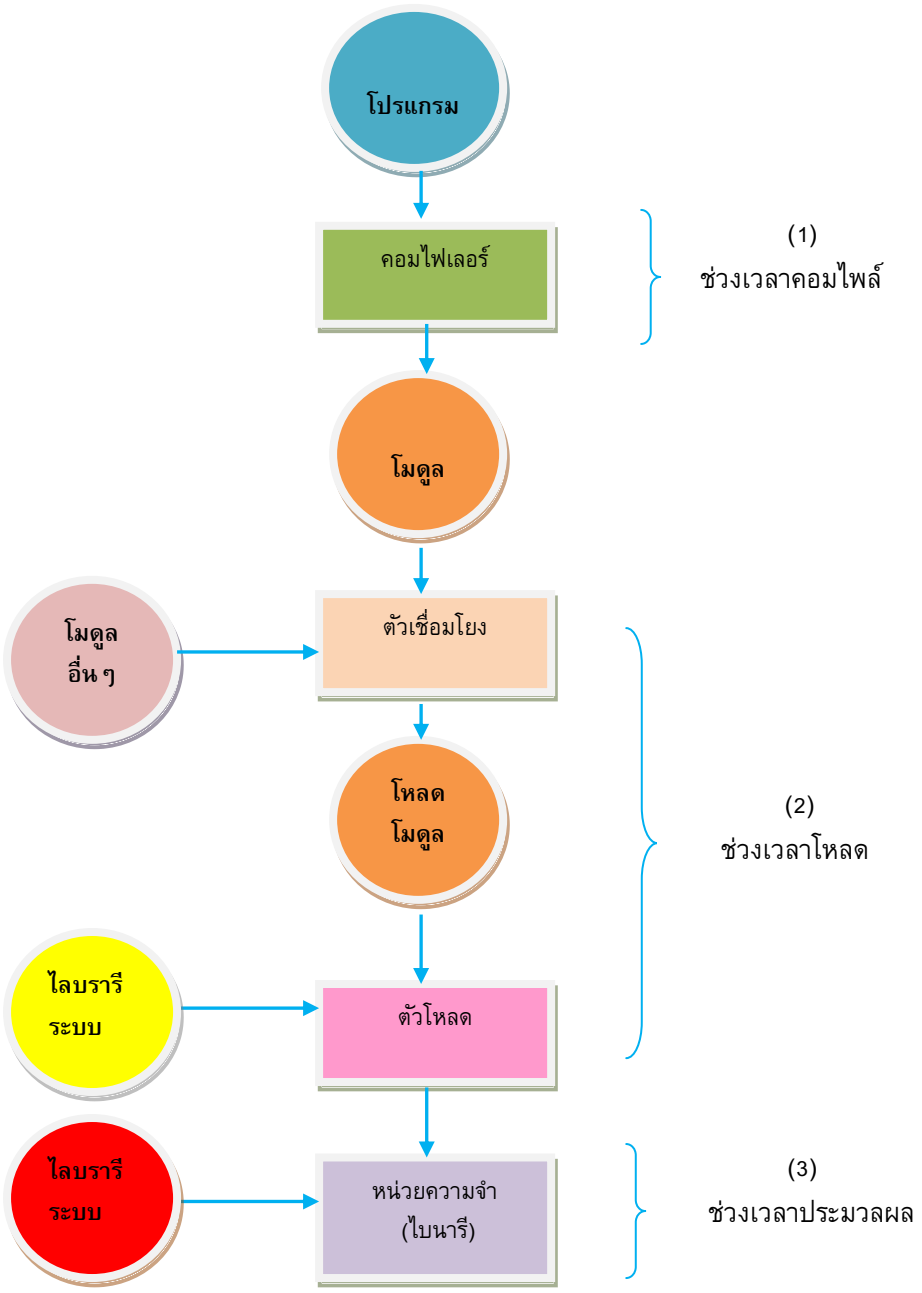
การจำแนกการเชื่อมโยงของคำสั่งและตำแหน่งของข้อมูลในหน่วยความจำสามารถแบ่งแต่ละขั้นตอนตามช่วงเวลาได้ดังนี้

1.ช่วงเวลาคอมไพล์ (Compile time) คือ ช่วงเวลาที่คำสั่งหรือข้อมูลถูกแปลคำสั่งโดยการคอมไพล์โดยตัวคอมไพเลอร์ (Compiler) โดยตัวคอมไพเลอร์จะค้นหาตำแหน่งจริง (Absolute Code) ในหน่วยความจำเมื่อพบแล้วจะสร้างโค้ดทำให้ระบบปฏิบัติการสามารถประมวลผลคำสั่งหรือข้อมูลนั้นได้ทันที แต่หากตำแหน่งจริงถูกเปลี่ยนแปลง ตัวคอมไพเลอร์ก็จะทำการรีคอมไพล์ (Recompile) โค้ดคำสั่งหรือข้อมูลนั้นใหม่ทุกครั้ง

2.ช่วงเวลาโหลด (Load time) ช่วงเวลาที่คำสั่งหรือข้อมูลถูกโหลดเข้าสู่หน่วยความจำโดยลำดับแรกตัวคอมไพเลอร์จะสร้างโค้ดที่สามารถประมวลผลได้ทันที (Relocation code)ขึ้นมาก่อน หลังจาก

คำสั่งหรือข้อมูลถูกโหลดเข้าสู่หน่วยความจำแล้วตัวคอมไพเลอร์จะทำการแปลตำแหน่งที่โหลดเข้ามาให้เป็นตำแหน่งจริง (Absolute Code) เพื่อให้ระบบปฏิบัติการสามารถประมวลผลคำสั่งหรือข้อมูลนั้นได้โดยไม่ต้องเสียเวลาในการคอมไพล์ใหม่ทุกครั้ง แต่จะเสียเวลาเฉพาะตอนโหลดคำสั่งหรือข้อมูลนั้นเข้ามาในหน่วยความจำ

3.ช่วงเวลาประมวลผล (Execution time) ช่วงเวลาที่คำสั่งหรือข้อมูลถูกประมวลผล โดยตัวคอมไพเลอร์จะทำการเชื่อมโยงตำแหน่งและแปลโค้ดคำสั่งหรือข้อมูลของตำแหน่งนั้นๆ เข้าไปเก็บไว้ในหน่วยความจำขณะที่ระบบปฏิบัติการกำลังประมวลผล (Run time) ทำให้ระบบปฏิบัติการต้องเสียเวลาในการแปลตำแหน่งคำสั่งหรือข้อมูลต่างๆ ก่อนถูกนำมาเข้าสู่กระบวนการประมวลผลทุกครั้ง



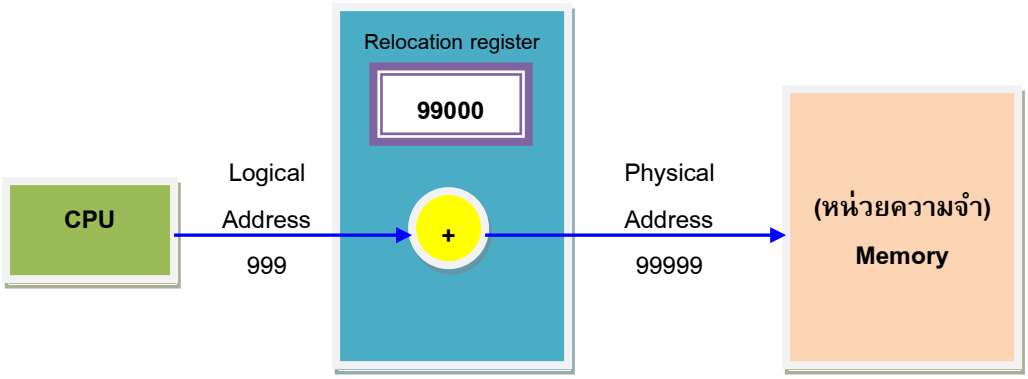
รูปที่ 2.2 แสดงช่วงเวลาเชื่อมโยงตำแหน่ง (Address Binding)

จากรูปที่ 2.2 ช่วงเวลาที่ 1 (Compile time) ใช้ในการคอมไพล์โมดูลเพื่อส่งคำสั่งหรือข้อมูลไปยังตำแหน่งจริง (Absolute Code) เพื่อส่งเข้าสู่ชวงที่ 2 (Load time) แล้วส่งไปสร้างโค้ดจึงโหลดโมดูลทั้งหมดเข้าสู่หน่วยความจำจนกระทั่งช่วงเวลาที่ 3 (Execution time) ช่วงเวลาที่คำสั่งหรือข้อมูลถูกประมวลผลและจัดเก็บลงในหน่วยความจำ

การเชื่อมโยงระหว่างพื้นที่ทางกายภาพกับพื้นที่ทางตรรกะ
(Logical- Versus Physical-Address Space)

การเชื่อมโยงพื้นที่ในหน่วยความจำจะต้องมีการอ้างอิงถึงตำแหน่งที่เกี่ยวข้องอยู่ 2 ประเภทคือ
1.ตำแหน่งพื้นที่ทางตรรกะ (Logical Address Space) หรือเรียกอีกชื่อหนึ่งว่า ตำแหน่งเสมือน (Virtual Address) ซึ่งถูกสร้างขึ้น (Generate) โดยหน่วยประมวลผลกลาง (CPU) เพื่อใช้ในการแลกเปลี่ยนข้อมูล โดยกลุ่มของตำแหน่งพื้นที่ทางตรรกะ (Logical Address Space) ทั้งหมดถูกสร้างโดยโปรแกรม
2.ตำแหน่งพื้นที่ทางกายภาพ (Physical Address Space) คือตำแหน่งที่อยู่ในหน่วยความจำหลัก (Memory Unit) และทำงานโดยตอบสนอง (Corresponding) กับตำแหน่งทางตรระกะ (Logical Address) เสมอ

โดยช่วงเวลาที่มีการประมวลผล (Run-Time) ตำแหน่งทางตรรกะ (Virtual Address) และตำแหน่งทางกายภาพ (Physical Address) จะถูกจับคู่ (Mapping) โดยอุปกรณ์ฮาร์ดแวร์ชนิดหนึ่งที่เรียกว่า หน่วยจัดการหน่วยความจำ (Memory Management Unit : MMU) ทำหน้าที่ในแปลงตำแหน่งทางตรรกะ (Logical Address) ไปเป็นตำแหน่งทางกายภาพ (Physical Address) โดยนำค่าที่ได้เก็บไว้ในรีจิสเตอร์พื้นฐาน (Base-Register) ส่วนการย้ายตำแหน่ง (Relocation Register) ก็จะนำค่าที่ได้เก็บไว้ในรีจิสเตอร์พื้นฐาน (Base-Register) มาบวกกับค่าตำแหน่งทางตรรกะ (Logical Address) เพื่อใช้ในการหาคค่าของตำแหน่งทางกายภาพ (Physical Address) แล้วส่งผลลัพธ์ที่ได้เข้าสู่หน่วยความจำ (Memory) โดยกำหนดค่าเริ่มต้นตำแหน่งทางตรรกะ (Logical Address) อยู่ในช่วงระหว่าง 0 ถึง max และกำหนดค่าเริ่มต้นตำแหน่งทางกายภาพ (Physical Address) อยู่ในช่วงระหว่าง R + 0 ถึง R + max (R คือ Relocation Register) ดังรูปที่ 2.3



รูปที่ 2.3 ตัวอย่างของการทำงานของหน่วยจัดการหน่วยความจำ (Memory Management Unit : MMU)

จากรูปที่ 2.3 แสดงให้เห็นการทำงานของหน่วยจัดการหน่วยความจำ (Memory Management Unit) เราจะเห็นว่าหน่วยประมวลผลกลาง (CPU) ไม่สามารถเข้าถึงหน่วยความจำ (Memory) ได้โดยตรง จึงเป็นหน้าที่ของอุปกรณ์ฮาร์ดแวร์ที่ชื่อหน่วยจัดการหน่วยความจำ (Memory Management Unit : MMU) โดย MMU จะทำการแปลงตำแหน่งทางตรรกะ (Logical Address) โดยนำเอาค่า 999 จับคู่หรือบวกเข้ากับค่าในส่วนการย้ายตำแหน่ง (Relocation Register) คือค่า 99000 ผลลัพธ์ที่ได้เท่ากับค่า 99999 คือค่าของตำแหน่งทางกายภาพ (Physical Address) แล้วส่งผลลัพธ์ที่ได้เข้าสู่หน่วยความจำ (Memory) จึงจะทำให้หน่วยประมวลผลกลาง (CPU) เข้าถึงหน่วยความจำ (Memory) ได้นั้นเอง

การโหลดแบบพลวัต (Dynamic Loading)

เนื่องจากพื้นที่ที่ใช้ในการประมวลผลในหน่วยความจำทางกายภาพ (Physical Memory) มีการจำกัดขนาดข้อมูล ดังนั้นโปรแกรมคำสั่งและข้อมูลทั้งหมดของกระบวนการ (Process) จะต้องมีขนาดเล็กกว่าหน่วยความจำทางกายภาพ (Physical Memory) ในขณะที่โหลดข้อมูลทั้งหมดเข้าสู่พื้นที่ว่างในหน่วยความจำในแต่ละครั้งจะต้องมีการตรวจสอบขนาดของข้อมูล เพื่อจะช่วยให้การประมวลผลโปรแกรมคำสั่ง และการโหลดข้อมูลเข้าหน่วยความจำทำได้รวดเร็วขึ้น ลักษณะการทำงานดังกล่าวนี้เรียกว่า การโหลดแบบพลวัต (Dynamic Loading) ซึ่งวิธีการทำงานจะไม่โหลดโปรแกรมย่อย (Routine) จนกว่าจะมีการเรียกใช้งาน (Call) ทำให้ไม่สิ้นเปลืองเนื้อที่ในหน่วยความจำ ทุกครั้งที่โปรแกรมหลัก (Main Program) ถูกโหลดเข้าสู่หน่วยความจำเพื่อทำการประมวลผล และมีการเรียกใช้โปรแกรมย่อย (Routine) จะมีการตรวจสอบโปรแกรมแกรมย่อยแรกก่อน (Routine First) ควบคู่กับการโหลดโปรแกรมย่อยตัวอื่นไปด้วย กรณีที่ไม่มีการโหลดโปรแกรมย่อย ตัวเชื่อมโยงการย้ายตำแหน่ง (Relocation linking Loader) จะทำการโหลดโปรแกรมย่อยไปเก็บไว้ในตารางเพื่อระบุตำแหน่ง (Address Table) ซึ่งตัวโปรแกรมจะทำการปรับเปลี่ยนตำแหน่งที่มีผลกระทบกับการเคลื่อนย้ายตำแหน่งโดยไม่ต้องอาศัยระบบปฏิบัติในการจัดการ ซึ่งตัวโปรแกรมจะเป็นตัวจัดการเหตุการณ์ดังกล่าวเอง

การใช้ไลบรารีร่วมกัน และการเชื่อมโยงแบบพลวัต (Dynamic Linking and Shared Libraries)

วิธีการเชื่อมโยงแบบพลวัต (Dynamic Linking) มีความคล้ายกันกับการโหลดแบบพลวัต (Dynamic Loading) ต่างกันตรงเวลาที่ใช้ในการเชื่อมโยงที่มากกว่าเมื่อระบบต้องการเรียกใช้งานโปรแกรมย่อย (Subroutine Libraries) โดยใช้ชุดคำสั่งขนาดเล็กที่ชื่อว่า สตั๊บ (Stub) โดยที่ชุดคำสั่งนี้เมื่อถูกประมวลผล ก็จะทำการตรวจสอบว่ามีโปรแกรมย่อยที่ต้องการใช้งานอยู่ในหน่วยความจำแล้วหรือยัง ถ้าไม่มีชุดคำสั่งสตั๊บ (Stub) นี้ก็จะทำการโหลดโปรแกรมย่อยใหม่เข้าสู่หน่วยความจำทันทีและจะแทนที่ค่าตำแน่งโปรแกรมย่อยใหม่ทับตำแหน่งโปรแกรมย่อยเดิมทันทีโดยเรียกกระบวนการทำงานนี้ว่า วิธีการเชื่อมโยงแบบพลวัต (Dynamic Linking) โดยที่ทุกๆ โปรเซสจะใช้ภาษาของไลบรารีในการประมวลมลเพียงหนึ่งครั้งในการทำสำเนารหัสไลบรารี (Library code) ซึ่งเป็นไฟล์ที่มีนามสกุลเป็น .dll โดยการทำงานในการปรับเปลี่ยนไลบรารี (Update library) จากไลบรารีรุ่นเก่า (Old Library) เป็นไลบรารีรุ่น

ใหม่ (New Library) จะมีการสร้างการเชื่อมโยงไลบรารีแบบอัตโนมัติ เพื่อให้ระบบสามารถเรียกใช้งานไฟล์ .dll นี้จากโปรแกรมย่อยได้ โดยลักษณะการใช้ไฟล์ร่วมกัน (Shared Libraries)

การแบ่งส่วน (Overlay)

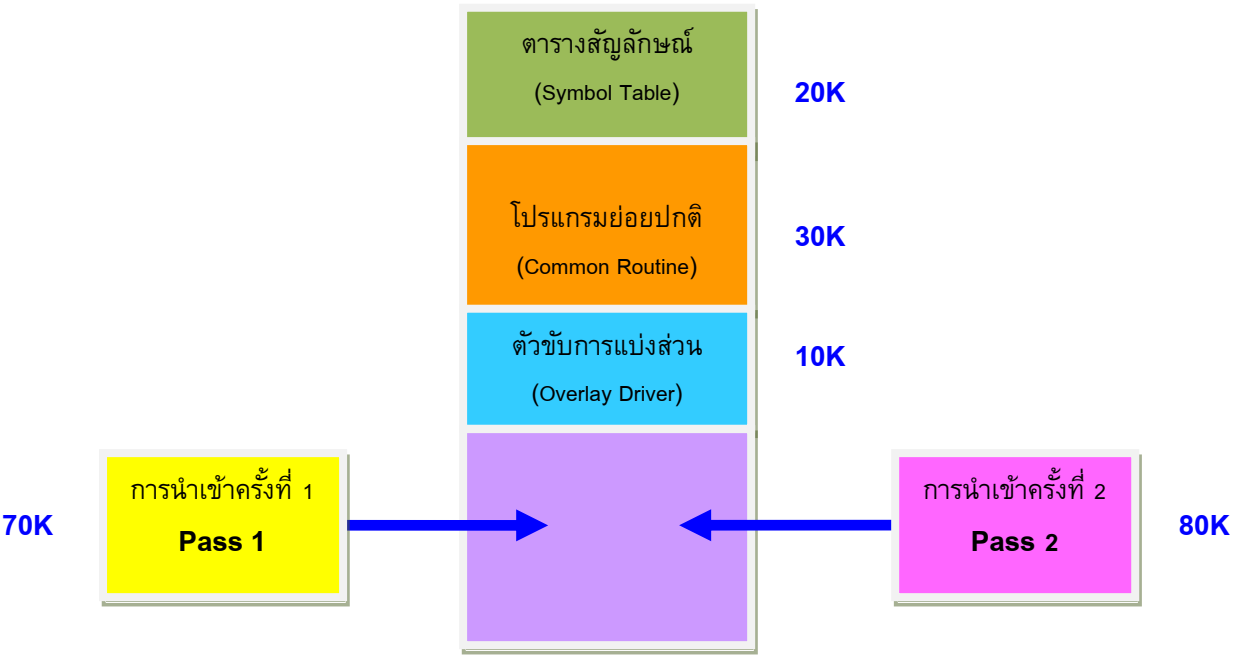
เป็นวิธีนี้ใช้ในกรณีที่โปรเซสมีขนาดใหญ่กว่าหน่วยความจำที่จัดเก็บ จำเป็นที่จะต้องจัดสรรหน่วยความจำให้เหมาะสมโดยการแบ่งส่วน (Overlay) หลักการทำงานของการแบ่งส่วนโดยที่คำสั่งและข้อมูลจะถูกเก็บอยู่ในหน่วยความจำตามขนาดของหน่วยความจำที่มี ซึ่งจะต้องอาศัยการทำงานของตัวขับในการแบ่งส่วน (Overlay Driver) เพื่อโหลดคำสั่งที่ต้องการใช้งานหน่วยความจำเข้ามาใช้งานจนเสร็จก่อนแล้วจึงสลับให้คำสั่งอื่นที่ต้องการใช้งานหน่วยความจำเข้ามาทำใช้งานหน่วยความจำต่อไป

ตัวอย่าง พิจารณาจากตัวเอสเซมเบอร์ในการจัดการนำเข้าการทำงานของ 2 โปรเซส โดยที่โปรเซส 1 ต้องการที่จะใช้ตารางสัญลักษณ์ (Symbol Table) ขณะที่โปรเซส 2 ต้องการที่จะสร้างคำสั่งภาษาเครื่อง (Machine-language code) เราอาจจะต้องทำการแบ่งพาร์ทิชั่นให้ตัวเอสเซมเบอร์ในการจัดการการนำเข้าทำงานของคำสั่งในโปรเซส 1 และโปรเซส 1 และการใช้งานตารางสัญลักษณ์ (Symbol Table) และโปรแกรมย่อยปกติ (Common Routine) ในหน่วยความจำร่วมกันทั้งสองโปรเซส โดยกำหนดขนาดการใช้งานของแต่ละคำสั่งได้ดังนี้

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

จะเห็นว่าการโหลดทุกคำสั่งหนึ่งครั้งจะต้องใช้หน่วยความจำถึง 200KB ซึ่งเรามีขนาดของหน่วยความจำเพียง 150KB ทำให้ไม่สามารถประมวลผลโปรเซสดังกล่าวได้ แต่จะสังเกตพบว่าการนำเข้าครั้งที่ 1 และการนำเข้าครั้งที่ 2 ไม่ต้องการใช้งานหน่วยความจำในเวลาเดียวกัน

วิธีการแก้ปัญหา เราจะแบ่งออกเป็นสองส่วน (Two Overlays) คือ Overlay A ทำงานกับตารางสัญลักษณ์ (Symbol Table) กับตารางสัญลักษณ์ (Symbol Table) โปรแกรมย่อยปกติ (Common Routine) และการนำเข้าครั้งที่ 1 (Pass 1) ส่วน Overlay B ก็ทำงาน (Symbol Table) โปรแกรมย่อยปกติ (Common Routine) และการนำเข้าครั้งที่ 2 (Pass 2) ระบบจะทำการเพิ่มตัวขับในการแบ่งส่วน (Overlay Driver) ขนาด 10 KB เพื่อโหลดคำสั่งใน Overlay A ที่ต้องการใช้งานหน่วยความจำ 120KB จนทำงานเสร็จ แล้วจึงกระโดดกลับไปที่ตัวขับในการแบ่งส่วนอีกครั้งเพื่อโหลดคำสั่งใน Overlay B ที่ต้องการใช้งานหน่วยความจำ 130KB โดยเข้าไปแทนที่ (Overwriting) Overlay A แสดงได้ดังรูปที่ 2.4 ซึ่งทำให้ตัวเอสเซมเบอร์สามารถที่จะประมวลบนพื้นที่หน่วยความจำขนาด 150KB ได้



รูปที่ 2.4 การแบ่งส่วนหน่วยความจำสำหรับการนำเข้าของโปรเซสครั้งที่ 1 - 2 ของตัวเอสเซมเบอร์ (Overlays for two-pass assembler)

กลยุทย์ในการจัดการหน่วยความจำ (Memory Strategy)

การจัดการหน่วยความจำถือว่าเป็นหน้าที่หนึ่งของระบบปฏิบัติการ เพื่อจัดสรรพื้นที่หน่วยความจำให้กับโปรแกรมหรือข้อมูลต่างๆ ได้อย่างถูกต้องเหมาะสม แบ่งออกเป็น 3 วิธี ดังนี้

1. **กลยุทธ์การโหลด (Fetch Strategy)** เป็นกลยุทธ์ที่ใช้ในการโหลดคำสั่งหรือข้อมูลต่างๆ จากหน่วยเก็บข้อมูลสำรองเข้าสู่หน่วยความจำหลัก แบ่งออกเป็น 2 วิธี
   1.1 Demand Fetch Strategy คือ วิธีการโหลดเฉพาะคำสั่งหรือข้อมูลต่างๆ ที่ต้องการใช้งานเข้าสู่หน่วยความจำหลัก
   1.2 Anticipate Fetch Strategy คือ วิธีการคาดเดาว่าคำสั่งหรือข้อมูลใดจะถูกโหลดเข้าสู่หน่วยความจำหลักก่อนที่จะถูกใช้งานจริง
2. **กลยุทธ์การวาง (Placement Strategy)** เป็นกลยุทธ์ที่ใช้ในการจัดว่างคำสั่งหรือข้อมูลใหม่เข้าสูหน่วยความจำหลักบนพื้นที่ว่างในหน่วยความจำที่เรียกว่า “โฮล (Hole)” เพื่อให้กับโปรแกรมหรือข้อมูลต่างๆ ไว้ให้เหมาะสมกับขนาดคำสั่งหรือข้อมูลนั้น
3. **กลยุทธ์การแทนที่ (Replacement Strategy)** โดยระบบปฏิบัติการจะเป็นส่วนที่ช่วยในการตัดสินใจว่าจะเลือกวิธีใดในการแทนที่คำสั่งข้อมูลในหน่วยความจำหลัก (Main Memory) ให้เหมาะสมที่สุดมี 5 วิธีดังนี้

2.1 วิธีสุ่ม (Random) คือ การสุ่มหาพื้นที่ว่างในหน่วยความจำ โดยคำสั่งหรือข้อมูลมีโอกาสถูกเลือกมาใช้งานเท่ากัน

2.2 วิธีมาก่อนได้ใช้ก่อน (First-in, First-out : FIFO) คือ คำสั่งหรือข้อมูลใดถูกโหลดเข้าไปใช้งานหน่วยความจำหลักก่อนจะถูกเลือกออกไปก่อน

2.3 NFU (Not Frequency Use) คือ การเลือกคำสั่งหรือข้อมูลที่ถูกใช้งานน้อยที่สุดออกไปก่อน เนื่องจากคำสั่งหรือข้อมูลกลุ่มนี้ถูกใช้งานน้อยโอกาสจะถูกโหลดเข้ามาใช้งานอีกก็น้อยตามไปด้วย ดังนั้นจึงควรนำออกจากหน่วยความจำหลัก

2.4 LRU (Lead Recently Use) คือ วิธีที่เก็บเวลาการใช้งานหน่วยความจำครั้งล่าสุด โดยคำสั่งหรือข้อมูลที่ไม่ได้ถูกนำมาใช้งานนานที่สุดจะถูกเลือกออกไปก่อน

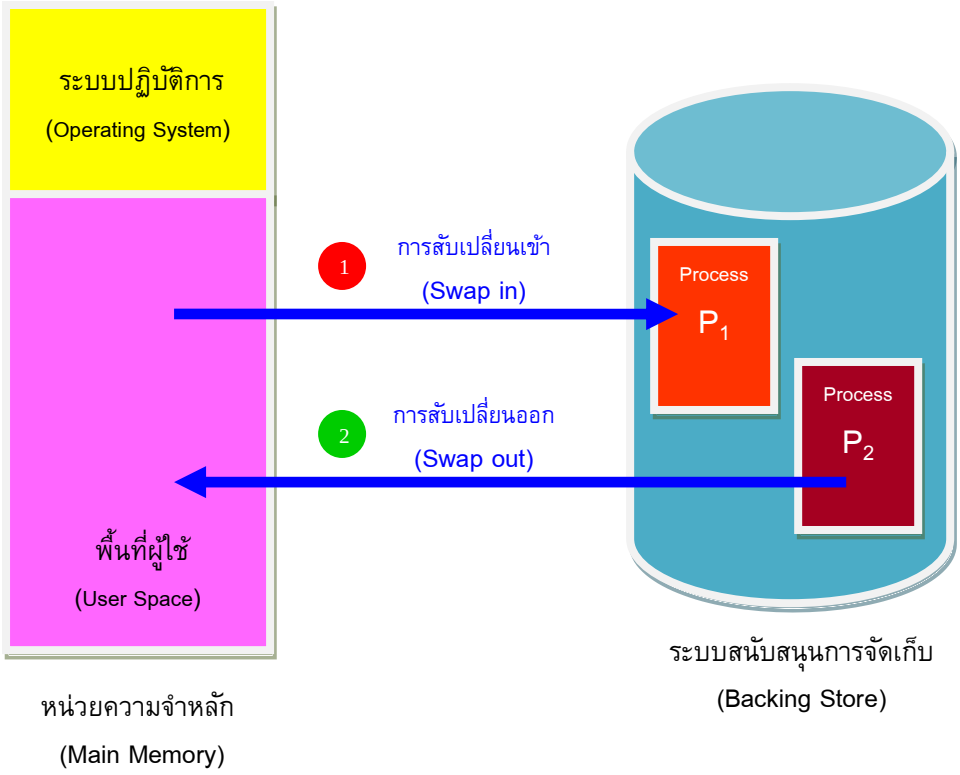
2.5 NRU (Not Recently Use) คือ วิธีนี้แต่ละคำสั่งหรือข้อมูลจะเพิ่มบิตข้อมูลที่เกี่ยวข้อง 2 บิต ด้วยกันคือ บิตอ้างอิง (Reference Bit) และบิตแก้ไข (Modify Bit) ซึ่งจะมีค่าเป็น 0 เมื่อมีการเข้าถึงคำสั่งหรือข้อมูลและจะมีค่าเป็น 1 เมื่อมีการแก้ไข

การสับเปลี่ยน (Swapping)

การสับเปลี่ยน (Swapping) เป็นวิธีที่ในการสลับโปรเซสที่ต้องการจะเข้าหรือออกเพื่อประมวลผลในหน่วยความจำ โดยหลักการจะต้องสลับโปรเซสเก่าออกมาก่อนแล้วจึงนำเข้าโปรเซสใหม่เข้าไปใช้งานหน่วยความจำโดยนำเข้าไปเก็บไว้ในระบบสนับสนุนการจัดเก็บ (Backing Store) แสดงได้ดังรูปที่ 2.5 โดยใช้อัลกอริทึมการกำหนดตารางลำดับความสำคัญ (Priority-base Scheduling Algorithms) ถ้าโปรเซสใดมีลำดับความสำคัญสูงกว่า (Higher-priority process) เข้ามาถึงและต้องการจะใช้บริการหน่วยความจำ ตัวจัดการหน่วยความจำสามารถที่จะทำการสลับเอาโปรเซสที่มีลำดับความสำคัญต่ำกว่า (Lower-priority process) ออกไปก่อนเพื่อให้โปรเซสใดมีลำดับความสำคัญสูงกว่า (Higher-priority process) ทำงานให้เสร็จก่อน โปรเซสที่มีลำดับความสำคัญต่ำกว่า (Lower-priority process) จึงจะสลับกลับมา (Swap back) เพื่อทำงานในหน่วยความจำต่อไปได้บางครั้งเราอาจจะเรียกวิธีการสลับแบบนี้ว่าการหมุนเข้า (Roll in) หรือหมุนออก (Roll out) โดยระบบปฏิบัติการจะเป็นส่วนจัดการเหตุการณ์ในการสลับโปรเซสเข้า-ออก จากหน่วยความจำดังนี้

1. กระบวนการ (Process) ดำเนินการเสร็จสิ้น
2. กระบวนการ (Process) ร้องขอการใช้งานอุปกรณ์ I/O
3. กระบวนการ (Process) หมดเวลาการทำงาน (Quantum Time)

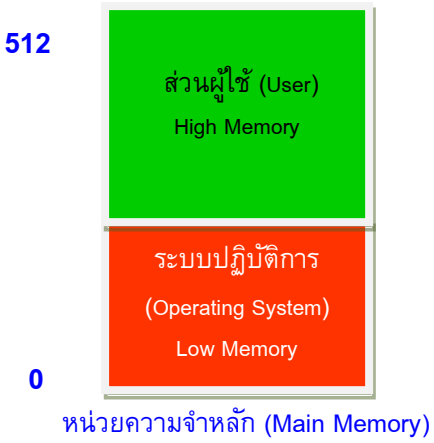
ดังนั้นหลักการสำคัญในการสับเปลี่ยน (Swapping) จะต้องคำนึงถึงระยะเวลาในการเคลื่อนย้าย (Transfer) กระบวนการซึ่งเป็นสัดส่วนโดยตรงกับจำนวนขนาดหรือพื้นที่ในหน่วยความจำที่มีอยู่ด้วย



รูปที่ 2.5 การสับเปลี่ยนระหว่าง 2 โปรเซส ในการใช้งานพื้นที่ดิสก์ในการจัดเก็บ
(Swapping of two processes using a disk as a backing store)

การจัดสรรหน่วยความจำที่ต่อเนื่องกัน (Continues Memory Allocation)

หน่วยความจำหลักต้องจำเป็นจะต้องอำนวยความสะดวกให้กับระบบปฏิบัติการและความหลากของผู้ใช้งานโปรเซส ดังนั้นในแต่ละโปรเซส (Process) จึงมีความต้องการใช้พื้นที่ในหน่วยความจำอย่างต่อเนื่อง จึงเป็นหน้าที่ของระบบปฏิบัติการในการจัดสรรพื้นที่หน่วยความจำในหน่วยความจำให้มีประสิทธิภาพสูงสุด ซึ่งโดยทั่วไปแล้วหน่วยความจำหลัก (Main Memory) จะถูกแบ่งออกเป็น 2 ส่วน คือ หน่วยความจำระดับบน (High Memory) ซึ่งเป็นส่วนที่ใช้ติดต่อกับส่วนของผู้ใช้ (User) และหน่วยความจำระดับล่าง (Low Memory) ซึ่งเป็นส่วนของระบบปฏิบัติการ (Operating System) ซึ่งแสดงได้ดังรูปที่ 2.6

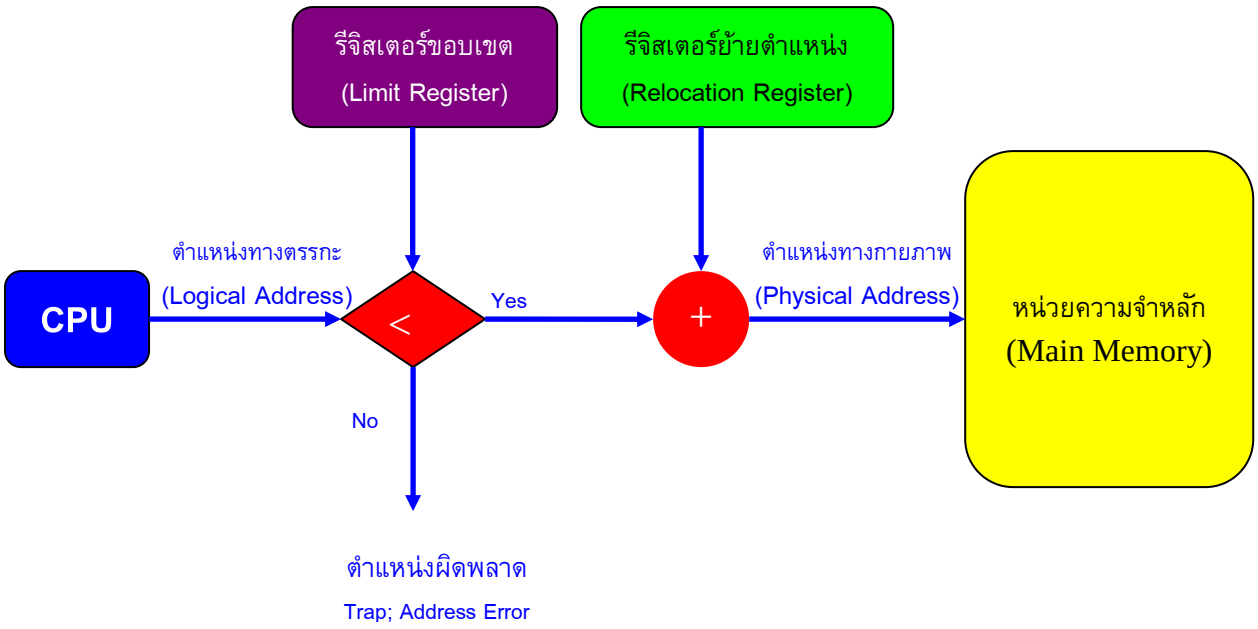


รูปที่ 2.6 การแบ่งระดับของหน่วยความจำหลัก (The Level of Main Memory)

นอกจากนี้การจัดสรรพื้นที่ในหน่วยความจำหลัก (Main Memory) เกี่ยวข้องกับการป้องกัน ซึ่งจะต้องพิจารณาจากผู้ใช้งานโปรเซส (User Processes) และผู้ที่เกี่ยวข้องกับการใช้งานโปรเซสอื่นด้วย 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 แสดงได้ดังรูปที่ 2.7



รูปที่ 2.7 แสดงภาพฮาร์ดแวร์ที่สนับสนุนการทำงานของรีจิสเตอร์ย้ายตำแหน่งรีจิสเตอร์ขอบเขต (Hardware Support for relocation and limit registers)

การจัดสรรพื้นที่ในหน่วยความจำแบ่งออกเป็น 2 รูปแบบ ดังนี้

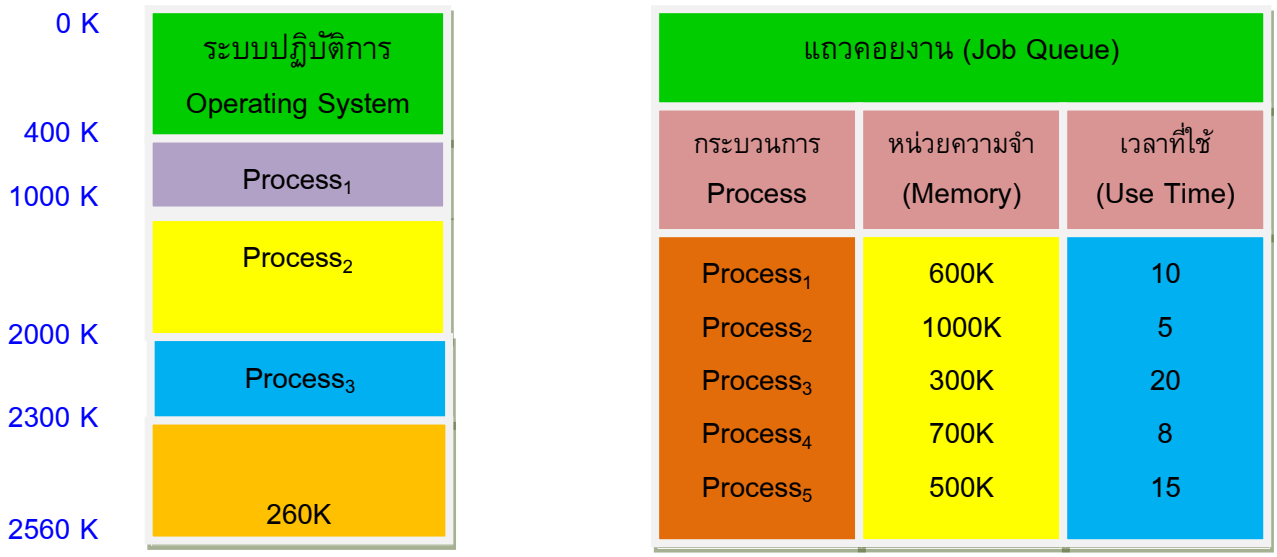
1.การจัดสรรหน่วยความจำแบบพื้นที่เดียว (Single-partition Allocation) วิธีนี้หน่วยความจำจะไม่ถูกแบ่งพื้นที่ โดยโปรเซสในส่วนของระบบปฏิบัติการจะอยู่ในส่วนหน่วยความจำระดับล่าง (Low Memory) และส่วนของโปรเซสของผู้ใช้อยู่ในส่วนหน่วยความจำระดับบน (High Memory) โดยเกี่ยวข้องกับรีจิสเตอร์ย้ายตำแหน่ง (Relocation Register) ในการแยกโปรเซส (Process) ในส่วนของผู้ใช้ออกจากส่วนของระบบปฏิบัติการและรีจิสเตอร์ขอบเขต (Limit Register) เป็นรีจิสเตอร์ที่ใช้ระบุขนาดของค่า

ตำแหน่งทางตรรกะ (Logical Address) โดยค่าตำแหน่งทางตรรกะ (Logical Address) ต้องน้อยกว่ารีจิสเตอร์ขอบเขต (Limit Register) เสมอ

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

2.1 การแบ่งแบบคงที่ (Fixed Partition) โดยการแบ่งพื้นที่หน่วยความจำออกเป็นหลาย ๆ พาร์ทิชั่น (Partition) แต่ละพาร์ทิชั่นมีขนาดเท่ากันและบรรจุโปรเซสอยู่ภายในเพียงหนึ่งโปรเซสเท่านั้น โดยที่จำนวนพาร์ทิชั่นถูกกำหนดโดยจำนวนโปรเซสที่มีอยู่ในหน่วยความจำ ข้อเสียของวิธีคือ หากโปรเซสที่บรรจุอยู่ในหน่วยความจำมีขนาดเล็กกว่าพาร์ทิชั่นที่กำหนดจะทำให้เหลือพื้นที่ในหน่วยความจำ (พาร์ทิชั่นมีพื้นที่ว่างเหลือ) เรียกพื้นที่ว่างที่เหลือนี้ว่า “Internal Fragmentation” กรณีที่ขนาดของโปรเซสมีขนาดใหญ่กว่าพาร์ทิชั่นที่กำหนดก็จะไม่สามารถนำโปรเซสนั้นเข้าไปใช้งานพื้นที่ในหน่วยความจำหลักได้

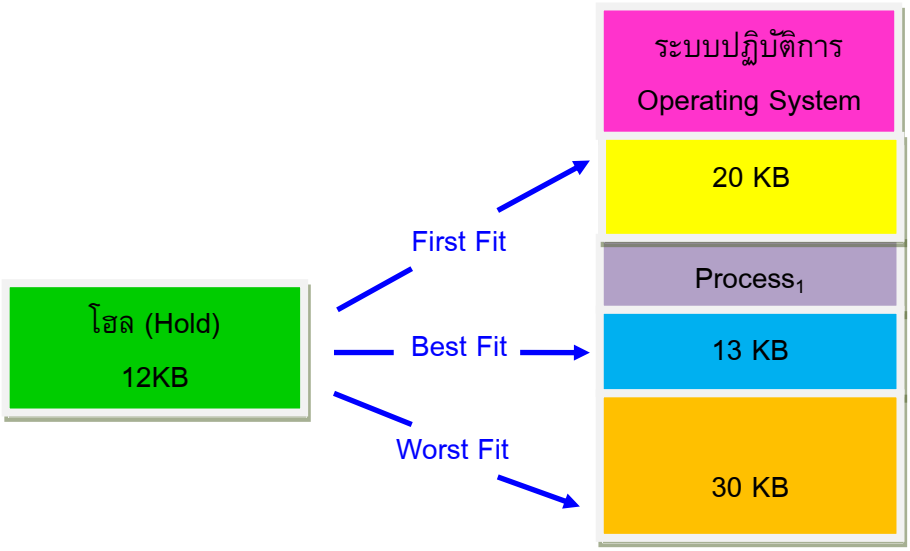
2.2 การแบ่งแบบพลวัต (Dynamic Partition) เป็นการแบ่งพื้นที่ตามขนาดของโปรเซส โดยการใช้พื้นที่ว่างทางกายภาพที่เรียกว่า โฮล (Hold) เมื่อมีโปรเซสต้องการใช้งานพื้นที่ในหน่วยความจำ ระบบปฏิบัติการจำมีหน้าที่ในค้นหาพื้นที่โฮล (Hold) ที่มีขนาดใหญ่เพียงพอกับโปรเซส แล้วจึงนำโปรเซสนั้นเข้าไปใช้งานหน่วยความจำและจะทำการเก็บข้อมูลของโปรเซสอื่นที่ยังไม่ได้นำเข้าไปใช้งานหน่วยความจำและพื้นที่ว่าง (Free Partition Hold) ที่ยังไม่ได้ถูกจัดสรรให้กับโปรเซสใดๆ ด้วยแถวคอยการทำงาน (Job Queue) โดยสัมพันธ์กับเวลาที่ใช้งานจริง (Use time) ของแต่โปรเซสด้วย แสดงดังรูปที่ 2.8



รูปที่ 2.8 แสดงการแบ่งหน่วยความจำแบบพลวัต (Dynamic Partition)

ปัญหาการจัดสรรพื้นที่หน่วยความจำแบบพลวัต (Dynamic Storage-allocation Problem) ที่พบบ่อยคือการคืนพื้นที่หน่วยความจำให้กับระบบเมื่อโปรเซสได้ทำงานเสร็จแล้วทำให้เกิดพื้นที่ว่างในหน่วยความจำ “โฮล (Hold)” ดังนั้นเราจึงใช้วิธีการวาง (Placement) เพื่อใช้แก้ปัญหาดังกล่าวซึ่งจะทำให้การจัดสรรพื้นที่ว่างในหน่วยความจำให้เกิดประโยชน์สูงสุด ดังนี้

- First fit เป็นการเลือกโฮลตัวแรก (First Hold) ที่พบก่อนและมีขนาดใหญ่เพียงพอกับโปรเซสที่จะนำเข้าไปวางในหน่วยความจำ ซึ่งเป็นวิธีที่ใช้เวลาน้อยที่สุด
- Best fit เป็นการเลือกโฮลที่มีขนากเล็กที่สุด (smallest Hold) เพื่อให้เหมาะสมที่สุดกับโปรเซสที่จะนำเข้าไปวางในหน่วยความจำ ซึ่งต้องทำการค้นหาทั้งรายการที่เก็บขนาดของโฮล (Hold) ที่ว่าง ซึ่งเป็นวิธีจัดสรรพื้นที่ว่างในหน่วยความจำให้เกิดประโยชน์สูงสุดเพราะเหลือพื้นที่ว่างโฮลน้อยที่สุด (smallest leftover hold) แต่ก็เสียเวลาการค้นหา
- Worst fit เป็นการเลือกโฮลขนาดที่ใหญ่ที่สุด (Largest Hold) เพื่อให้เหมาะสมที่สุดกับโปรเซสที่จะนำเข้าไปวางในหน่วยความจำ ซึ่งต้องทำการค้นหาทั้งรายการที่เก็บขนาดของโฮล (Hold) ที่ว่าง วิธีนี้อาจจะดีกว่าวิธี Best fit เพราะเหลือพื้นที่ว่างโฮลน้อยกว่า



รูปที่ 2.9 แสดงการวิธีการวาง (Placement) แบบต่างๆ

วิธีการจัดสรรพื้นที่ในหน่วยความจำให้กับแต่ละโปรเซส อาจเกิดการสูญเปล่าของเนื้อที่ว่างของโฮล (Hold) หรือพื้นที่ว่างที่ไม่สามารถนำไปใช้งานได้ทั้งภายใน (Internal Fragmentation) และภายนอก (External Fragmentation) ที่มีอยู่กระจัดกระจายอยู่เต็มไปหมดในหน่วยความจำหลัก (Main Memory) และมีขนาดใหญ่หรือเล็กเกินไปทำให้ไม่เหมาะสมกับขนาดของโปรเซสที่จะนำไปใช้งานได้ ซึ่งมีวิธีแก้ปัญหาการจัดสรรพื้นที่สูญเปล่าดังกล่าวอยู่ 2 วิธีดังนี้

1. การบีบอัด (Compression) หรือการจัดระเบียบพื้นที่ (Defragmentation) เป็นการจัดพื้นที่ว่างระหว่างโปรเซสใหม่ โดยการสับเปลี่ยนตำแหน่งของโปรเซสต่างๆ ในหน่วยความจำให้

เรียงต่อกัน ทำให้มีพื้นที่เพิ่มขึ้นหรือมีขนาดใหญ่ขึ้นซึ่งเหมาะกับการจัดสรรพื้นที่แบบพลวัต (Dynamic Allocation) ระบบปฏิบัติการจะเป็นตัวจัดการ โดยที่โปรเซสต่างๆ จะถูกย้ายไปยังตำแหน่งต่างๆ โดยอัตโนมัติ

2. อนุญาตให้ตำแหน่งพื้นที่ทางตรรกะ (Logical Address) ของโปรเซสที่ไม่ได้อยู่ติดกัน (Noncontiguous) โดยที่โปรเซสจะถูกจัดสรรไว้บนพื้นที่หน่วยความจำทางกายภาพ (Allocated Physical Memory) ที่ใดก็ได้ที่ทุกตำแหน่งสามารถใช้งานได้

การแบ่งหน้า (Paging)

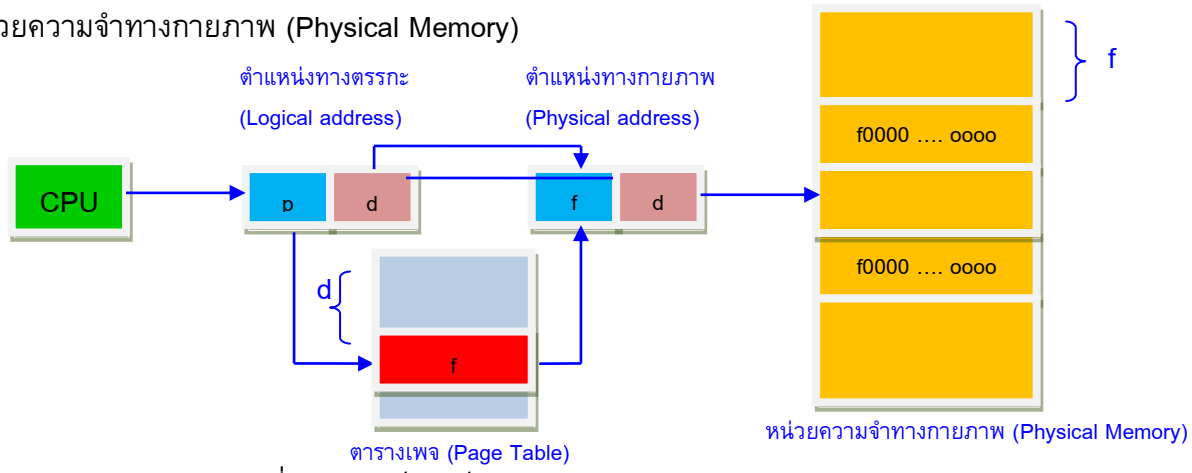
การแบ่งหน้า (Paging) เป็นการจัดสรรพื้นที่ว่างบนหน่วยความจำ โดยทำให้โปรเซสที่อยู่บนหน่วยความจำได้โดยไม่ต้องเรียงต่อเนื่องกันทั้งโปรเซสเป็นการใช้พื้นที่ว่างอยู่กระจัดกระจายโดยไม่สูญเปล่า และไม่จำเป็นจะต้องบีบอัดพื้นที่ว่างในหน่วยความจำก่อน แบ่งออกเป็น 2 ประเภท ดังนี้

1. การจัดสรรหน่วยความจำทางกายภาพ (Paging Model of Physical Memory) เป็นวิธีการแบ่งพื้นที่ให้มีขนาดคง (Fixed-Size Block) เรียกพื้นที่ส่วนนี้ว่า เฟรม (Frames) โดยที่ขนาดของเพจถูกกำหนดโดยฮาร์ดแวร์ ขนาดของเพจเป็นขนาดยกกำลัง 2 มีขนาดอยู่ระหว่าง 512 ไบต์ถึง 16 MB ต่อเพจ ขึ้นอยู่กับสถาปัตยกรรมของคอมพิวเตอร์

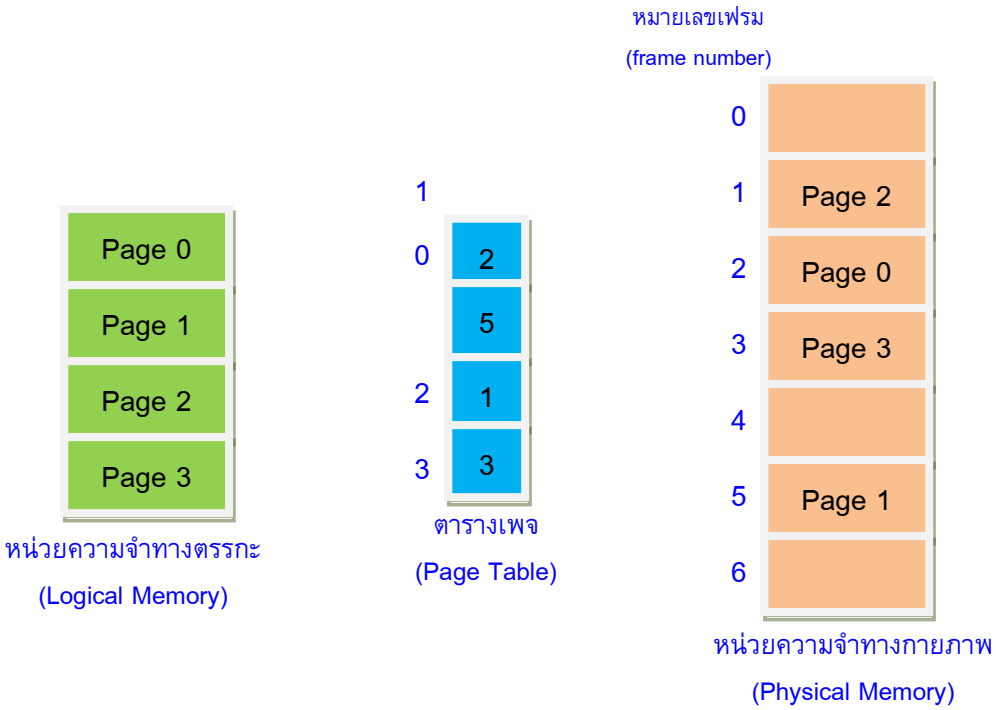
2. การจัดสรรหน่วยความจำทางตรรกะ (Paging Model of Logical Memory) เป็นวิธีการแบ่งพื้นที่แบบบล็อค (Block) เรียกพื้นที่ส่วนนี้ว่า เพจ (Pages) โดยกำหนดขนาดเพจเท่ากับขนาดเฟรม รูปที่ 2.10 แสดงภาพของฮาร์ดแวร์ซึ่งจะเป็นตัวจัดการแบ่งหน้า ทุกๆ ตำแหน่งจะถูกจัดสรรโดยหน่วยประมวลผลกลาง (CPU) โดยแบ่งออกเป็นสองส่วน 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)



รูปที่ 2.10 ฮาร์ดแวร์การแบ่งตาราง (Hardware Paging)



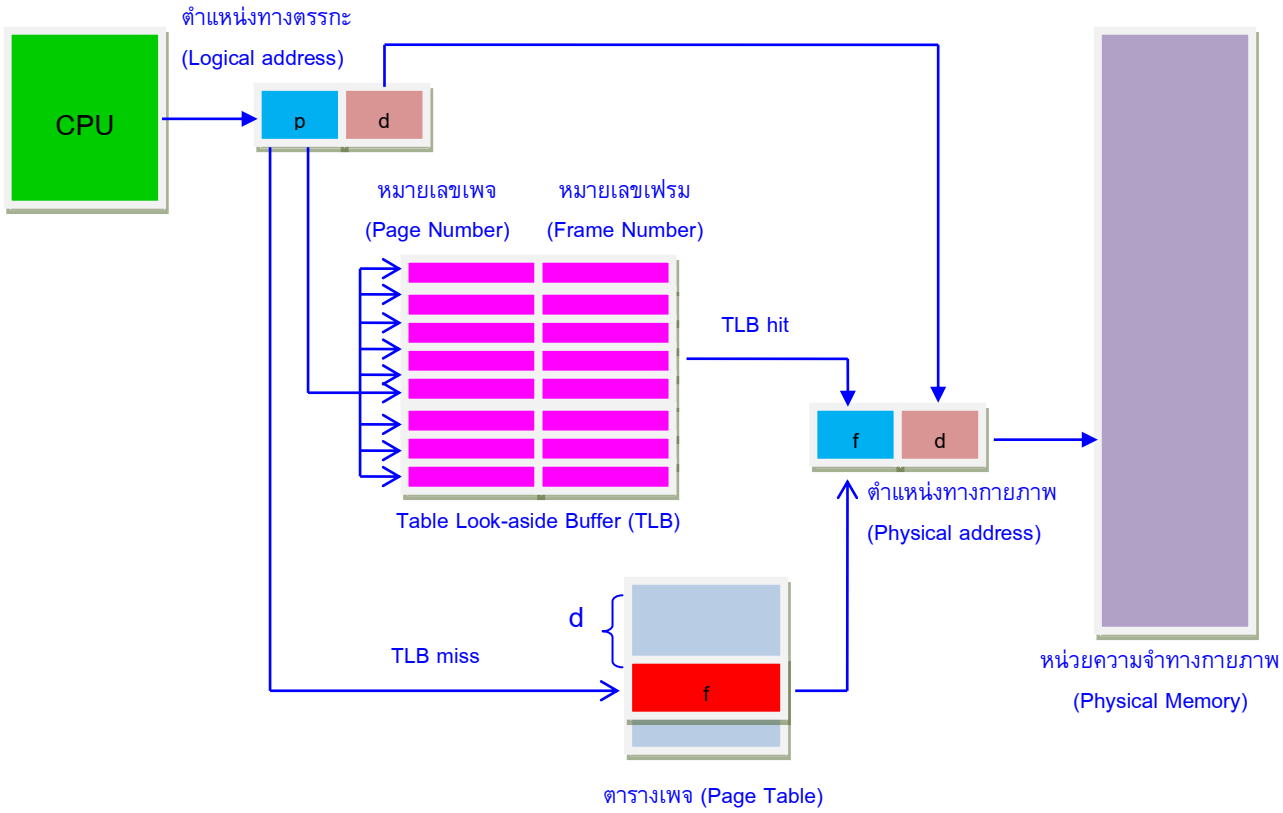
รูปที่ 2.11 แสดงรูปแบบการแบ่งเพจในหน่วยความจำแบบตรรกะและหน่วยความจำแบบกายภาพ (Paging model of logical and physical memory)

ฮาร์ดแวร์กับการสนับสนุนการแบ่งหน้า (Hardware Support)

การนำฮาร์ดแวร์กับการสนับสนุนการจัดการตารางเพจ (Page Table) ทำได้หลายทาง ซึ่งการเข่ถึงตำแหน่งในหน่วยความจำแต่ละครั้ง จะต้องมีอ่านข้อมูลจากหน่วยความจำทางกายภาพ (Physical Memory) ถึงสองครั้ง ครั้งที่หนึ่งจากตารางเพจ (Page Table) และครั้งที่สองจากตำแหน่งข้อมูลทางกายภาพ (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) หรือเท็กซ์ (Tag) ส่วนที่สองเก็บค่า (Value) หากต้องการอ่านข้อมูลจากหน่วยความจำก็จำทำการ เปรียบเทียบกุญแจว่าตรงกันหรือไม่ ถ้าพบรายการที่ค้นหาที่สอดคล้องกับค่าของฟิลด์ (Value Field) ก็จะส่งคืนค่ากับไปให้ ซึ่งประสิทธิภาพของการค้นหาจะรวดเร็ว แต่ฮาร์ดแวร์ชนิดนี้จะมีราคาแพง และค่าของจำนวนข้อมูลที่เข้ามาอยู่ใน TLB จะมีขนาดไม่ใหญ่มากนักมักมีค่าอยู่ระหว่าง 64 ถึง 1024 ตัว

จากรูปที่ 2.12 เป็นการอ้างถึงตำแหน่งทางตรรกะ (Logical Address) ซึ่งในการค้นหาจะใช้การส่งหมายเลขเพจ (Page Number) ที่ต้องการค้นหาเข้าสู่ TLB และเปรียบเทียบกับหมายเลขเฟรม (Frame

Number) ที่สัมพันธ์กับหมายเลขเพจ (Page Numbers) ทุกตัวพร้อมๆ กัน ซึ่งการค้นหาตำแหน่งใน TLB เรียกวิธีการนี้ว่า “ค้นหาแบบคู่ขนาน (Parallel Search)” กรณีที่ไม่พบหมายเลขเพจที่ต้องการค้นหาใน TLB ก็จะทำการค้นหาจากตารางเพจ (Page Table) ว่าอยู่เฟรมใดในหน่วยความจำหลักอีกที



รูปที่ 2.12 ฮาร์ดแวร์กับการสนับสนุนการแบ่งหน้าด้วย 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 เราต้องการการค้นหาหมายเลขเพจใน TLB แล้วเจอ 80% ของเวลาทั้งหมดหรือถ้าเราใช้เวลา 20 Nanoseconds ในการค้นหาใน 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 และเสี่ยเวลาครั้งที่สองในการค้นหาหมายเลขเฟรม (Frame Numbers) อีก 100 Nanoseconds ซึ่งจะใช้เวลาทั้งหมดในการค้นหาเท่ากับ 200 Nanoseconds ดังนั้นวิธีการคำนวณหา

ประสิทธิภาพของเวลาในการเข้าถึงหน่วยความจำ (Effective Memory-Access Time) เราจึงถั่วเฉลี่ยไปตามเหตุการณ์ต่างที่เกิดขึ้น จึงสรุปได้ว่า

Effective Memory-Access Time = 0.80 X 120 X 0.20 X 220

= 140 Nanoseconds

สรุปได้ว่า เวลาจะช้าลงไปถึง 40% ในการเข้าถึงหน่วยความจำ

(คือจาก 100 เป็น 140 นาโนวินาที)

<u>อีกหนึ่งตัวอย่าง</u> ถ้า 98% ของอัตราส่วนที่พบหมายความว่าอย่างไร อธิบายได้ดังนี้

Effective Memory-Access Time = 0.98 X 120 X 0.20 X 220

= 122 Nanoseconds

สรุปได้ว่า เวลาจะช้าลงไปถึง 22% ในการเข้าถึงหน่วยความจำ

(คือจาก 100 เป็น 122 นาโนวินาที)

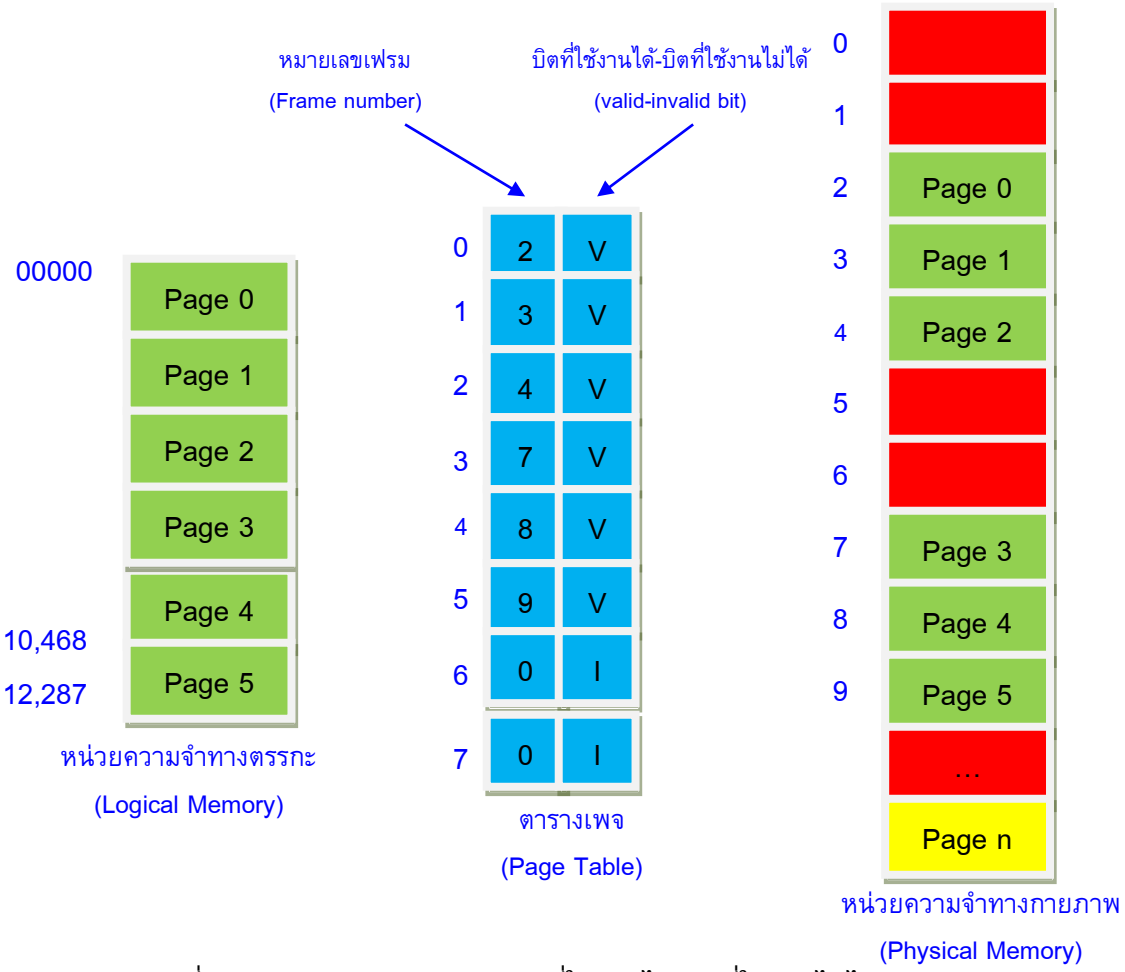
การป้องกันหน่วยความจำ (Memory Protection)

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)” จึงจำเป็นต้องโหลดเพจข้อมูลเข้าสู่หน่วยความจำก่อนแล้วจึงเปลี่ยนค่าของบิตใช้งานไม่ได้ (Invalid Bit) ให้เป็นบิตใช้งานได้ (Valid Bit) แล้วจึงทำการประมวลผลข้อมูลนั้นใหม่อีกครั้งหนึ่ง

รูปที่ 2.13 แสดงตารางเพจของบิตที่ใช้งานได้-บิตที่ใช้งานไม่ได้ (V-I Bit) ของระบบปฏิบัติการ 14 บิต (14-bit) ซึ่งมีพื้นที่ใช้งานได้ตั้งแต่ตำแหน่งที่ 0 ถึง 16383 และคำสั่งเพื่อใช้งานตำแหน่งตั้งแต่ 0 ถึง 10468 มีขนาดเพจ (Page Size) เท่ากับ 2 KB ตำแหน่งในเพจที่ 0, 1, 2, 3, 4, 5 จะมีการแม็บ (Map) ไปยังตารางเพจ (Page Table) เข้ากับหมายเลขตารางเฟรม (Frame Number) 2, 3, 4, 7, 8, 9 เพื่อบอกสถานการณ์ใช้งานบนหน่วยความจำทางกายภาพ (Physical Memory) ส่วนตารางเพจที่ 6 และ 7 บอกสถานะบิตเป็นบิตใช้งานไม่ได้ (Invalid Bit) ระบบปฏิบัติการ (Operating System) จะเป็นต้องโหลดเพจข้อมูลเข้าสู่หน่วยความจำก่อนแล้วจึงเปลี่ยนค่าของบิตใช้งานไม่ได้ (Invalid Bit) ให้เป็นบิตใช้งานได้ (Valid Bit) แล้วจึงทำการประมวลผลข้อมูลนั้นใหม่อีกครั้งหนึ่ง



รูปที่ 2.13 แสดงตารางเพจของบิตที่ใช้งานได้-บิตที่ใช้งานไม่ได้

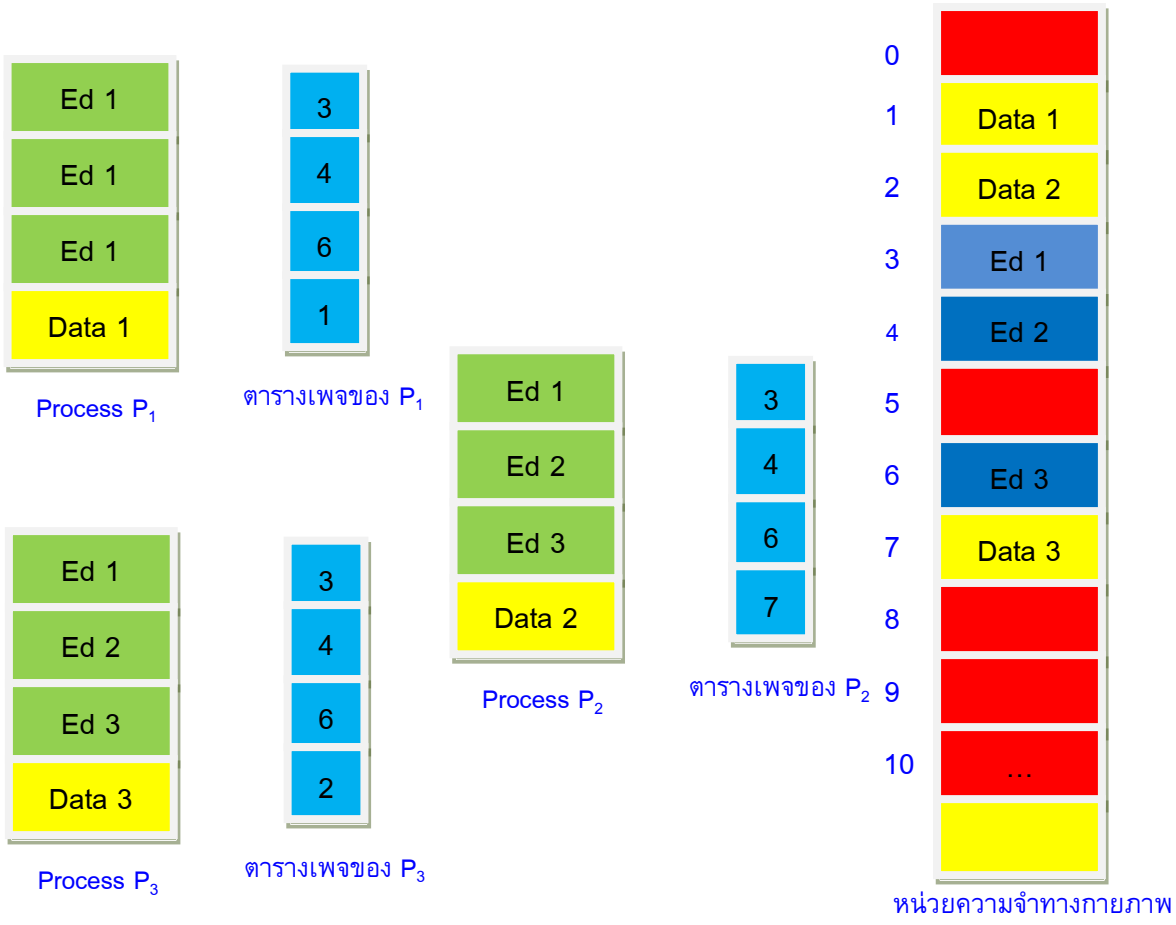
(Valid (v) or Invalid (i) Bit in a Page Table)

การใช้เพจร่วมกัน (Shared Pages)

เป็นวิธีการแบ่งปันการใช้เพจร่วมกันในรูปแบบการแบ่งเวลา (Time-Sharing) โดยข้อมูลนั้นจะเป็นข้อมูลที่ใช้อยู่ในลักษณะมีสามารถเปลี่ยนแปลงได้ (Non-Self-modifying Code) ในขณะที่มีการประมวลผล แต่ใช้งานร่วมกันได้ เรียกข้อมูลชนิดนี้ว่า “Reentrant Code” หรือ “Pure Code” ซึ่งจะเก็บไว้ในตำแหน่งทางตรรกะ (Logical Address) เดียวกัน แต่ละโปรเซสมีข้อมูลเพจ (Own Data Page) เป็นของตัวเอง

รูปที่ 2.14 แสดงการใช้เพจร่วมกันระหว่างโปรเซส ซึ่งประกอบไปด้วย 3 โปรเซส คือ P_1, P_2, และ P_3 ต้องการใช้งานโปรแกรม Text Editor ร่วมกันใน Page 0, Page1 และ Page3 ที่ถูกจัดเก็บไว้ในหมายเลขเฟรมที่ 3, 4 และ 6 ดังนั้นถ้าระบบปฏิบัติการมีผู้ต้องการใช้งาน 40 คน กำลังใช้งานโปรแกรม Text Editor ซึ่งใช้พื้นที่ 150 KB และใช้เพจสำหรับเก็บข้อมูล 50 KB ดังนั้นหากโปรเซสต่างใช้งานพื้นที่ในหน่วยความจำไม่ร่วมกันจะต้องใช้พื้นที่ในหน่วยความจำเท่ากับ 40 x (150 + 50) = 8000

KB แต่หากโปรเซสทั้งหมดใช้งานเพจร่วมกันจะใช้พื้นที่ในหน่วยความจำเท่ากับ 150 + (40 x 50) = 1250 KB ซึ่งจะช่วยลดการใช้พื้นที่ในหน่วยความจำลงได้ แต่ก็มีข้อเสียคือถ้าโปรเซสใดโปรเซสหนึ่งเสร็จจะไม่สามารถลบข้อมูลในเพจที่ใช้งานร่วมกัน (Shared Pages) ได้ ต้องรอจนกว่าโปรเซสอื่นจะใช้งานข้อมูลในเพจที่ใช้งานร่วมกันเสร็จก่อนจึงจะลบข้อมูลในเพจได้ การใช้งานของโปรแกรมร่วมกันแบบอื่น เช่น คอมไพเลอร์ (Compilers) การประมวลผลในส่วนไลบรารี (Run-Time Libraries) ระบบฐานข้อมูล (Database System) เป็นต้น



รูปที่ 2.14 แสดงภาพแวดล้อมของการใช้งานเพจร่วมกัน
(Sharing of Code In a Paging Environment)

โครงสร้างของตารางเพจ (Memory Protection)

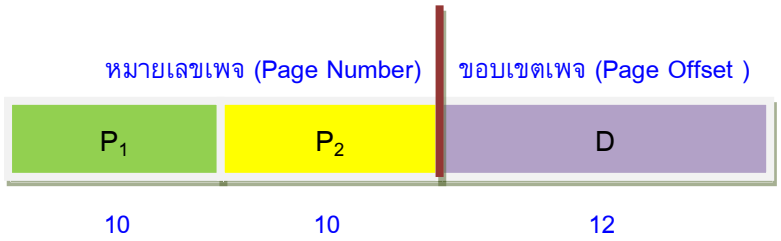
เทคนิคการสร้างตารางเพจแบ่งออกเป็น 3 รูปแบบ ได้แก่

1.โครงสร้างแบบลำดับชั้น (Hierarchical Paging) ระบบคอมพิวเตอร์สมัยใหม่จะสบับสนุนตำแหน่งพื้นที่ทางตรรกะ (Logical-Address Space) ขนาดใหญ่ (2^{32} ถึง 2^{64}) ซึ่งโครงสร้างแบบลำดับชั้นเป็นการแบ่งพื้นที่ทางตรรกะออกเป็นตารางเพจ (Page Table) หลายตาราง โดยใช้อักอริทึมในการแบ่งเพจแบบ 2 ระดับ (Two-Level Page Table) ได้แก่

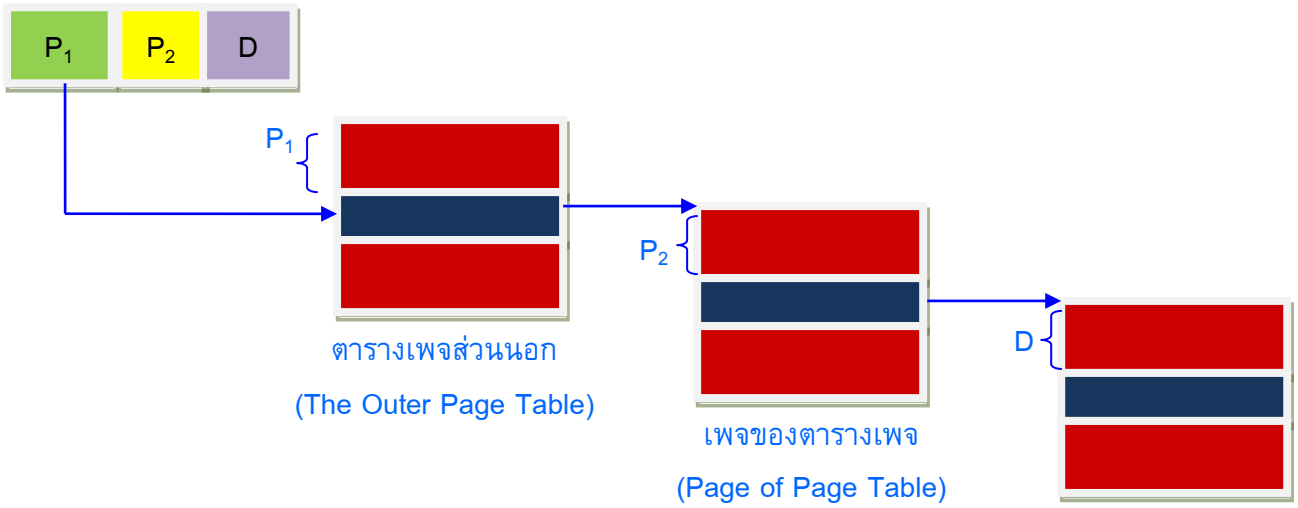
1.1 หมายเลขเพจ (Page Number)

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



รูปที่ 2.15 แสดงโครงสร้างตำแหน่งทางตรรกะแบบ 2 ระดับขนาด 32 บิต ซึ่งประกอบไปด้วย โปรเซส P_1 เป็นดัชนีที่อ้างอิงไปยังตารางเพจส่วนแรกที่เรียกว่า “ตารางเพจส่วนนอก (The Outer Page Table)” หลังจากนั้นก็จะอ้างอิงไปยังส่วนที่สองที่เรียกว่า “เพจของตารางเพจ (Page of Page Table)” โดยมีโปรเซส P_2 อยู่ในเพจของตารางเพจ



รูปที่ 2.15 แสดงโครงสร้างตำแหน่งทางตรรกะแบบ 2 ระดับขนาด 32 บิต
(Address Translation for a Two-Level 32-Bit Paging Architecture)

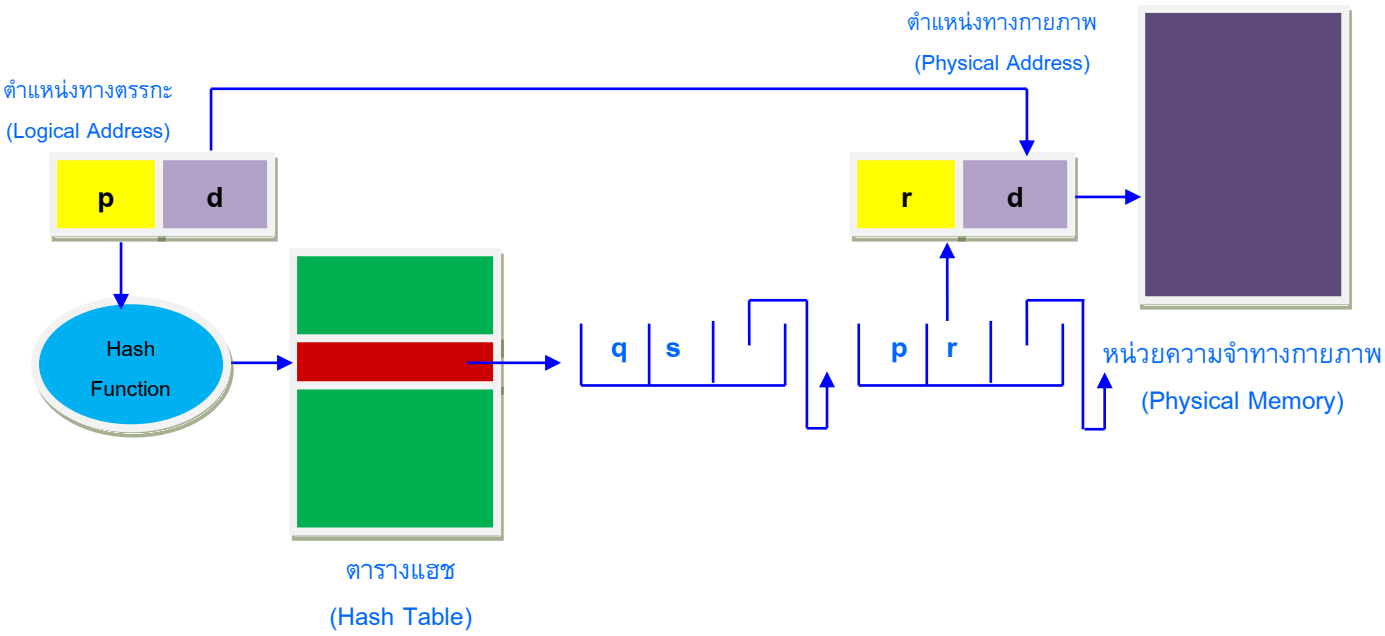
2.โครงสร้างแบบตารางเฮช (Hash Page Table) ระบบคอมพิวเตอร์ที่จะใช้โครงสร้างแบบนี้ ต้องมีตำแหน่งขนาดพื้นที่ (Address Space) มากกว่า 32 บิต โดยค่าของเฮชจะอยู่ภายในหมายเลขเพจเสมือน (Virtual-Page Number) ซึ่งแต่ละหน่วย (Elements) ในตารางเฮชภายในจะมีลิงก์ลิสต์ (Link List) เชื่อมโยงไปยังหน่วยที่อยู่ในพื้นที่เดียวกัน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

2.1 ค่าหมายเลขเพจเสมือน (Virtual-Page Number)

2.2 ค่าดรรชนีที่ชี้ไปยังเฟรมเพจ (Page Frame)

2.3 ค่าของพอยเตอร์ (Pointer) ที่ชี้ไปยังหน่วยที่เชื่อมโยงในลิงก์ลิสต์ (Link List)

จากรูปที่ 2.16 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

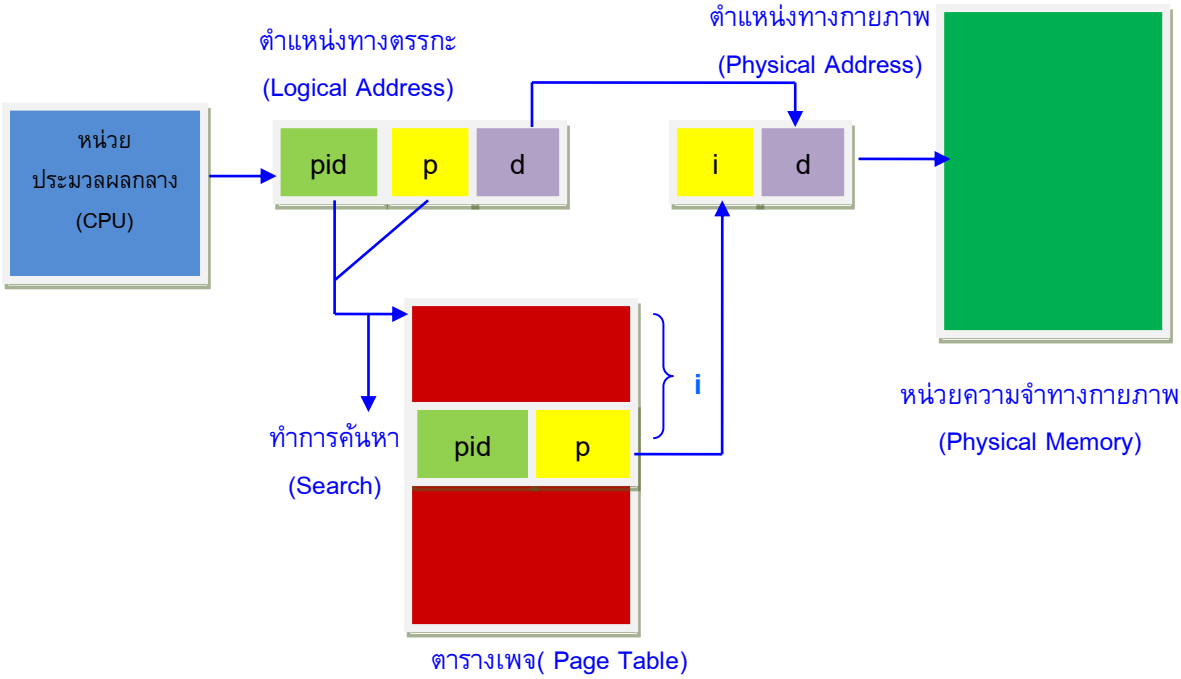


รูปที่ 2.16 แสดงโครงสร้างแบบตารางแฮช (Hashed Page Table)

3.โครงสร้างเพจแบบผกผัน (Inverted Page Table) ระบบคอมพิวเตอร์ที่จะใช้โครงสร้างแบบนี้ ต้องมีคุณสมบัติกำหนดโดยตารางเพจ (Page Table) จะมีเพียงหนึ่งโปรเซสใช้งานพื้นที่หน่วยความจำ โดยโปรแกรมหรือข้อมูลจะประกอบไปด้วยตำแหน่งของเพจเสมือน (Virtual-Address of Page) ที่เก็บอยู่ในหน่วยความจำจริง (Real Memory) ดังรูปที่ 2.17 จะมีเพียงหนึ่งตารางเพจ (One Page Table) เท่านั้นที่อยู่ในหน่วยความจำ โดยแต่ละตำแหน่งของเพจเสมือน (Virtual-Address of Page) ประกอบไปด้วย

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

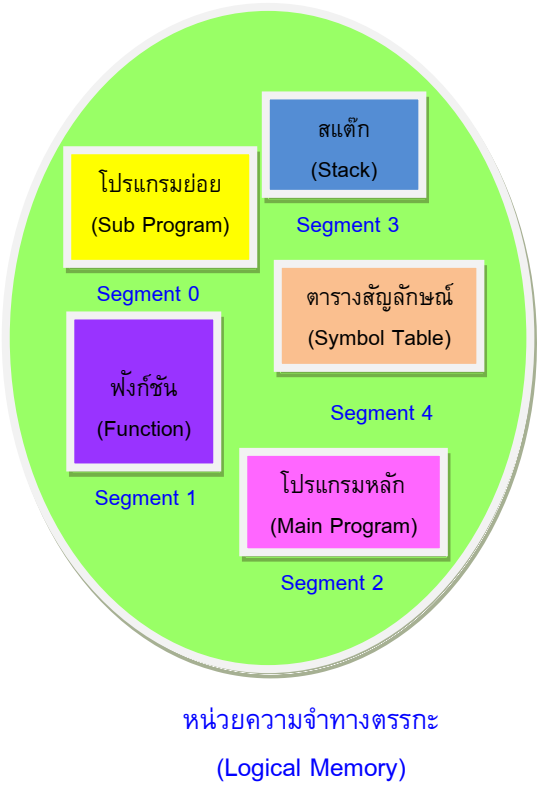
และแต่ละตารางเพจแบบผกผัน (Inverted Page Table) ประกอบด้วย <process-id, page-number> โดยจะทำการค้นหาโดยการจับคู่ตำแหน่งของเพจที่เหมือนกัน (Match) ในตารางเพจ (Page Table) ถ้าพบก็จะทำการ Map ค่าของ entry i กับตำแหน่งทางกายภาพ (Physical Address) เพื่อส่งเข้าไปประมวลผลในหน่วยความจำหลัก แต่ถ้าไม่พบก็จะทำการค้นต่อไปเรื่อยๆ ถึงแม้ว่าวิธีจะลดความต้องการใช้งานหน่วยความจำอย่างมีประสิทธิภาพ แต่ก็จะเสียเวลาในการค้นหาตำแหน่งในตารางในกรณีที่มีการอ้างถึงเหตุการณ์ที่เกิดขึ้น



รูปที่ 2.17 แสดงโครงสร้างแบบผกผัน (Inverted Page Table)

การแบ่งส่วน (Segmentation)

วิธีการนี้เป็นการจัดสรรพื้นที่ในหน่วยความจำหลักออกเป็นส่วนๆ ตามขนาดของโปรแกรมหรือโมดูลย่อย เรียกพื้นที่นี้ว่า Segment โดยแต่ละโมดูลจะถูกแบ่งออกเป็นส่วนๆ ที่มีขนาดไม่เท่ากัน เช่น โปรแกรมหลัก ไรซีเยอร์ ฟังก์ชัน ตัวแปร บล็อก สแต็ก ตารางสัญลักษณ์ และอาร์เรย์ ซึ่งแต่ละโมดูลจะมีการทำงานสัมพันธ์กัน แสดงดังรูปที่ 2.18



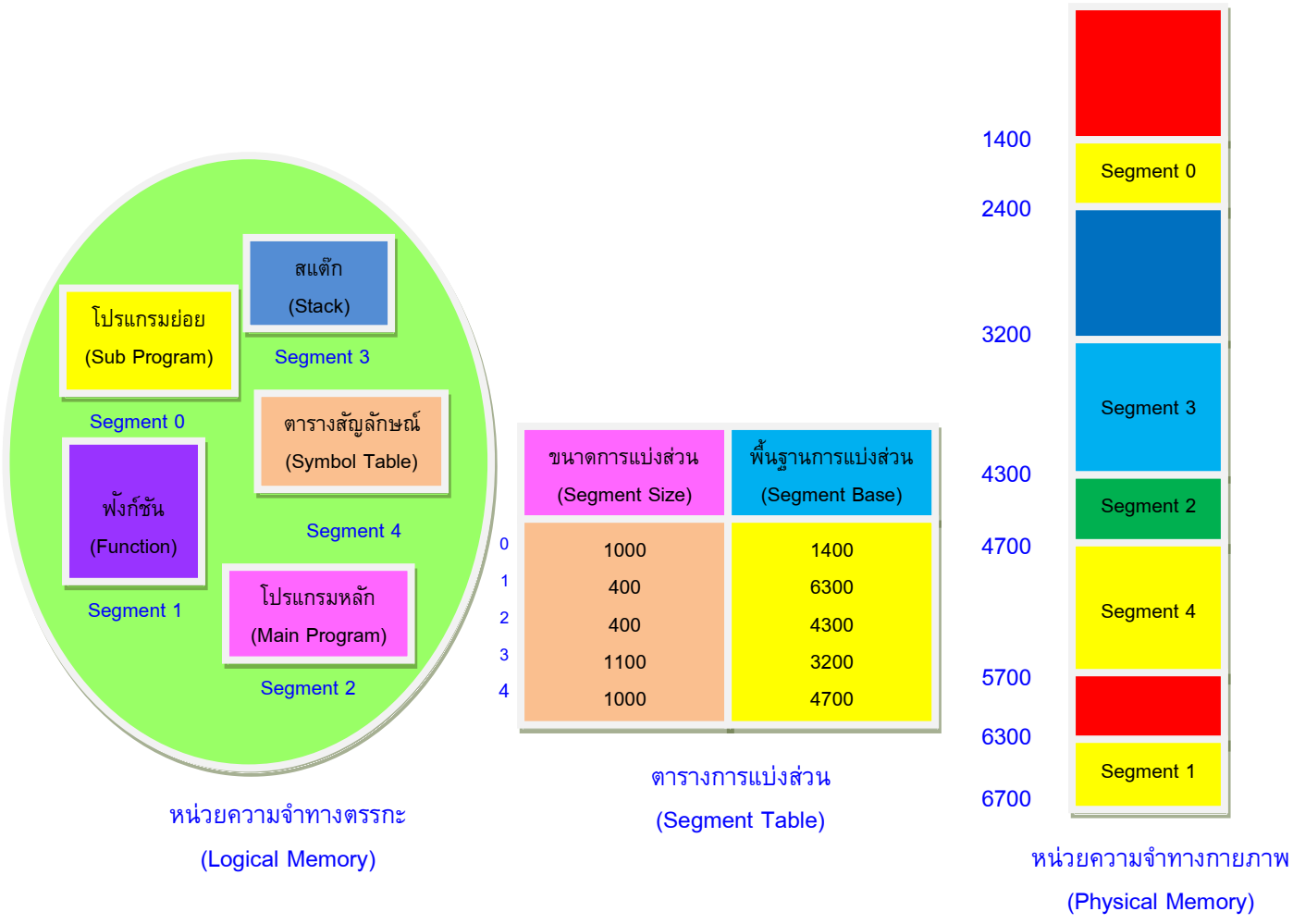
รูปที่ 2.18 แสดงโครงสร้างแบบแบ่งส่วน (Segmentation)

ตัวอย่าง จากรูปที่ 2.19 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 ซึ่ง Segment 0 มีความยาวเพียง 1000 ไบต์ เป็นหน้าที่ของระบบปฏิบัติการในการเลื่อน (Tab) ไปยัง Segment อื่นๆ ที่มีขนาดความยาวไบต์ (Bytes Long) เพียงพอแทน



รูปที่ 2.19 แสดงโครงสร้างตารางการแบ่งส่วน (Segmentation Table)

สรุป

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

อัลกอริทึมในการจัดการหน่วยความจำ (Memory Management Algorithms) มี 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) เป็นพื้นฐานที่ชาวยสนับสนุนการทำงานของกระบวนการการแบ่งหน้า (Paging) และการแบ่งส่วน (Segmentation) โดยโครงสร้างตารางการจับคู่ระหว่างข้อมูลและตำแหน่งของหน่วยความจำหลัก
2. ประสิทธิภาพ (Performance) วิธีที่ใช้จัดการหน่วยความจำหลักแต่ละวิธีจะมีความซับซ้อน และใช้ระยะเวลาในการจับคู่ระหว่างหน่วยความจำทางตรรกะ (Logical memory) และหน่วยความจำทางกายภาพ (Physical memory) ดังนั้น ระบบปฏิบัติการจึงต้องเลือกใช้วิธีที่เหมาะสมและรวดเร็ว
3. การจัดการพื้นที่ว่าง (Fragmentation) ในระบบคอมพิวเตอร์แบบหลายโปรแกรม (Multiprogramming) มีความสามารถในการประมวลผลสูง จะต้องมีการจัดการพื้นที่ว่างที่เหมาะสมกับขนาดของโปรเซส (Process) เพื่อป้องกันปัญหาที่เกิดขึ้น
4. การย้ายตำแหน่ง (Relocation) การนจัดการหน่วยความจำหลักที่ดีควรสามารถย้ายโปรเซส (Process) ในหน่วยความจำหลักได้อย่างอัตโนมัติเพื่อแก้ปัญหาการเกิดพื้นที่ว่าง (Fragmentation) ในหน่วยความจำหลัก
5. การสลับ (Swapping) กลไกการสลับโปรเซส (Process) เข้า/ออก จากหน่วยความจำหลัก ควรให้มีการสลับโปรเซสให้น้อยที่สุดภายในหนึ่งหน่วยเวลา
6. การใช้งานร่วมกัน (Sharing) ระบบปฏิบัติการที่ดีควรสามารถจัดสรรโปแกรมหรือข้อมูลต่างๆ ให้สามารถทำงานร่วมกันได้ เพื่อให้โปรเซส (Process) จำนวนมากสามารถทำงานร่วมกันได้
7. การป้องกัน (Protection) การป้องกันคำสั่งหรือข้อมูลต่างๆ โครงสร้างข้อกำหนดเงื่อนไขในการอ่านหรือเขียนข้อมูล โดยระบบต้องมีการตรวจสอบคำสั่งหรือข้อมูลต่างๆ ขณะกำลังประมวลผล พร้อมถึงแสดงข้อผิดพลาดได้