6.5 Virtual Memory

หน่วยความจำเสมือน

Virtual Memory

เป็นวิธีการหนึ่งที่สามารถทำให้โปรเซสเซอร์ทำงานได้ ถึงแม้ว่าโปรเซสนั้นจะไม่ได้อยู่ในหน่วยความจำหลักทั้งหมดก็ตาม

ระบบปฏิบัติการจะทำหน้าที่เก็บบางส่วนของโปรแกรมที่กำลังทำงานไว้ในหน่วยความจำหลัก และเก็บส่วนที่เหลือไว้ในฮาร์ดดิสก์

ข้อดีคือโปรแกรมของผู้ใช้สามารถมีขนาดใหญ่กว่าหน่วยความจำจริงได้ ทำให้ผู้เขียนโปรแกรมทำการเขียนโปรแกรมได้อย่างอิสระมากขึ้น ไม่ต้องกังวลถึงขนาดของหน่วยความจำอีกต่อไป

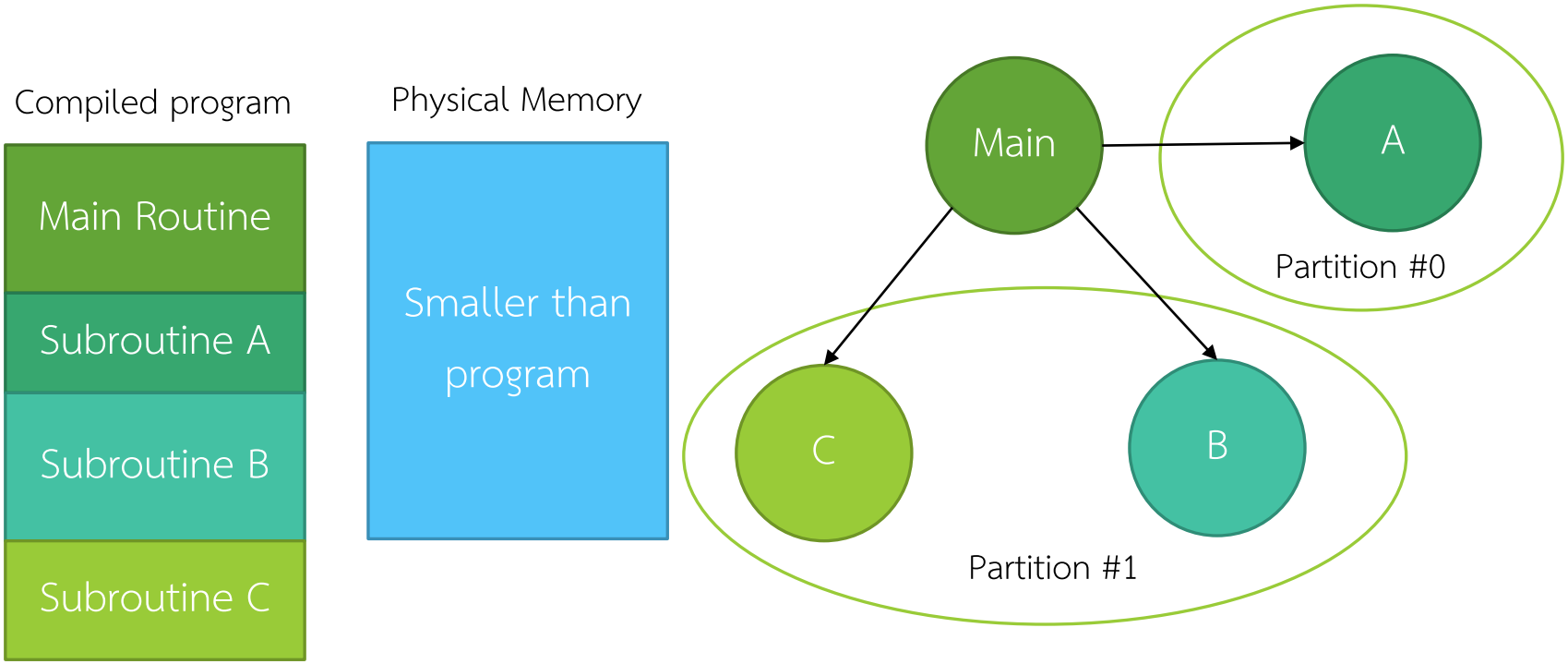
ข้อเสียคืออาจทำให้ประสิทธิภาพของระบบทำงานได้ช้าลงอย่างมาก ถ้ามีการออกแบบไม่รอบคอบ

การวางโปรแกรมทับซ้อนกัน (Overlays)

เป็นกลยุทธของการใช้ดิสก์เพื่อขยายพื้นที่สำหรับเก็บข้อมูลในหน่วยความจำในสมัยแรกๆ โดยใช้วิธีวางโปรแกรมซ้อนทับกัน (Overlays)

จะอนุญาตให้โปรแกรมทำการเขียนโค้ดของตัวเองทับโค้ดอื่นๆ ตามที่ต้องการได้ โดยโปรแกรมเมอร์จะเป็นผู้ที่ทำหน้าที่จัดการกับหน่วยความจำให้ทำงานดังกล่าว

การจัดการโปรแกรมที่มี 1 โปรแกรมหลัก และ 3 โปรแกรมย่อย



การวางโปรแกรมทับซ้อนกัน (Overlays)

วิธีการนี้จะทำงานได้ดีในสถานการณ์ที่หลายหลาย แต่วิธีการที่ดีคือการให้ระบบปฏิบัติการจัดการเกี่ยวกับการวางโปรแกรมซ้อนทับกัน

เมื่อมีการโหลดโปรแกรมเข้าสู่หน่วยความจำหลักมากกว่า 1 โปรแกรม การจัดวางโปรแกรมทับซ้อนกันจะไม่สามารถทำได้ถ้าไม่มีการติดต่อกับระบบปฏิบัติการเพื่อหาว่าส่วนใดในหน่วยความจำที่ว่างและสามารถนำมาใช้งานได้

ในสถานการณ์นี้จะทำให้การจัดการดังกล่าวซับซ้อนมากยิ่งขึ้น เพราะว่าจะต้องมีการทำงานร่วมกันระหว่างระบบปฏิบัติการและโปรแกรมอื่นๆ

6.6 การแบ่งออกเป็นหน้า (Paging)

การแบ่งออกเป็นหน้า (Paging)

เป็นวิธีการส่งโปรแกรมซ้อนทับกัน (Overlay) อีกรูปแบบหนึ่ง ซึ่งเป็นวิธีการจัดการแบบอัตโนมัติโดยระบบปฏิบัติการ

วิธีการทำงานเริ่มต้นโดยมีการแบ่งพื้นที่และแอดเดรสของหน่วยความจำเสมือนออกเป็นบล็อกๆ ที่เท่ากัน ที่เรียกว่า “หน้า (Page)”

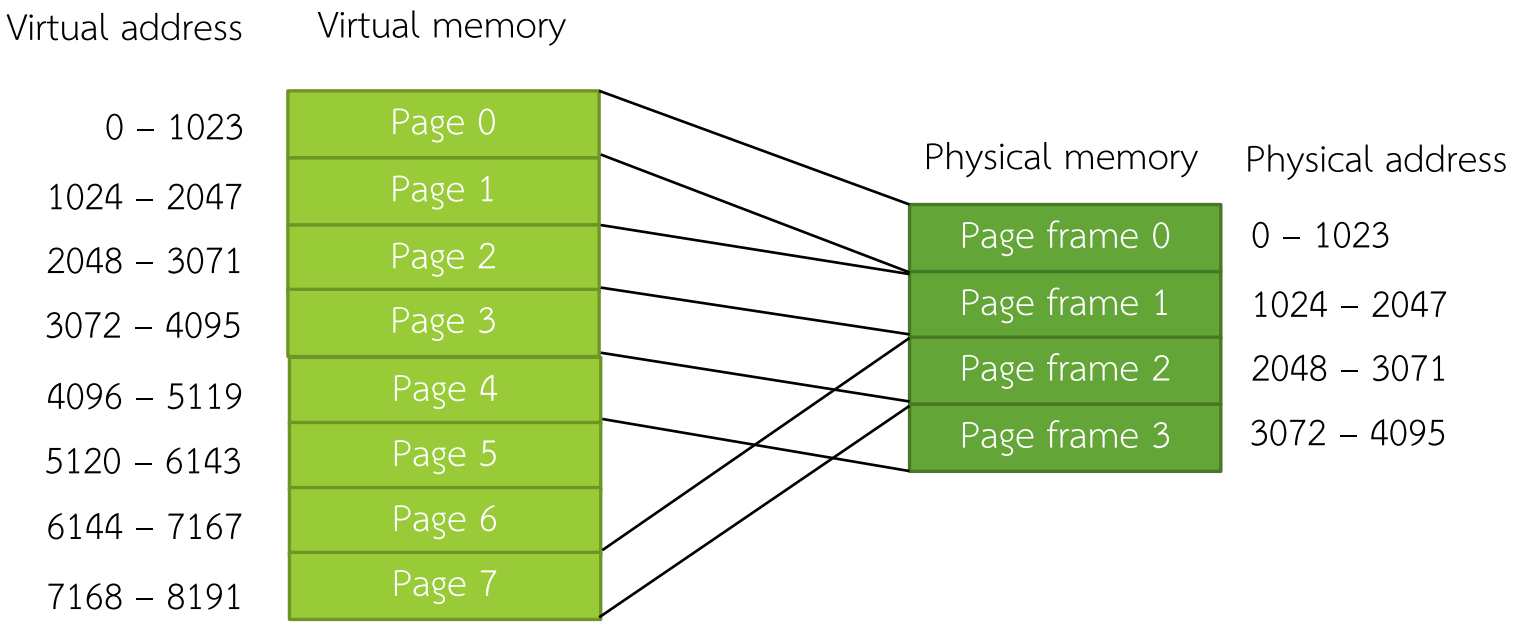
ขนาดของหน้าโดยปกติจะมีขนาดเป็นรูปแบบยกกำลังของ 2 เช่น 2^{10} = 1024 B

การแบ่งออกเป็นหน้านี้ทำให้หน่วยความจำหลักดูเหมือนมีขนาดที่ใหญ่ขึ้น เพราะมีการเชื่อมโยงหน่วยความจำหลักเข้ากับบางส่วนของหน่วยความจำเสมือน

การใช้งานหน่วยความจำเสมือนจะต้องมีการจัดการกับการอ้างอิงหน่วยความจำเสมือนในส่วนอื่นๆ ที่ไม่ได้ใช้งานหรือไม่ได้ถูกแมปเข้ากับหน่วยความจำหลัก

เมื่อมีหน่วยความจำเสมือนที่ถูกอ้างอิงนั้น ไม่ได้อยู่ในหน่วยความจำหลัก เรียกว่า “การผิดหน้า (Page fault)”

การเชื่อมโยงระหว่างหน่วยความจำเสมือนกับหน่วยความจำหลัก



การใช้งานหน่วยความจำเสมือน

1. ถ้าเฟรมของหน้าใดหน้าหนึ่งถูกกำหนดว่าจะมีการเขียนโปรแกรมหรือข้อมูลทับ เนื้อหาในเฟรมหน้านั้นก็จะถูกเขียนลงในหน่วยความจำสำรอง (ถ้ามีการเปลี่ยนแปลงข้อมูล) เพื่อให้มีการบันทึกค่าดังกล่าวก่อนที่เฟรมของหน้านั้นจะถูกเขียนข้อมูลทับ
2. หน้าของข้อมูลที่เราต้องการเข้าถึงและเก็บไว้ในหน่วยความจำสำรองจะถูกเขียนลงในหน่วยความจำหลัก
3. ตารางหน้า จะถูกอัปเดทข้อมูลการเชื่อมโยงระหว่างหน่วยความจำเสมือนเข้ากับหน่วยความจำหลักใหม่
4. ทำงานอื่นๆ ต่อไป

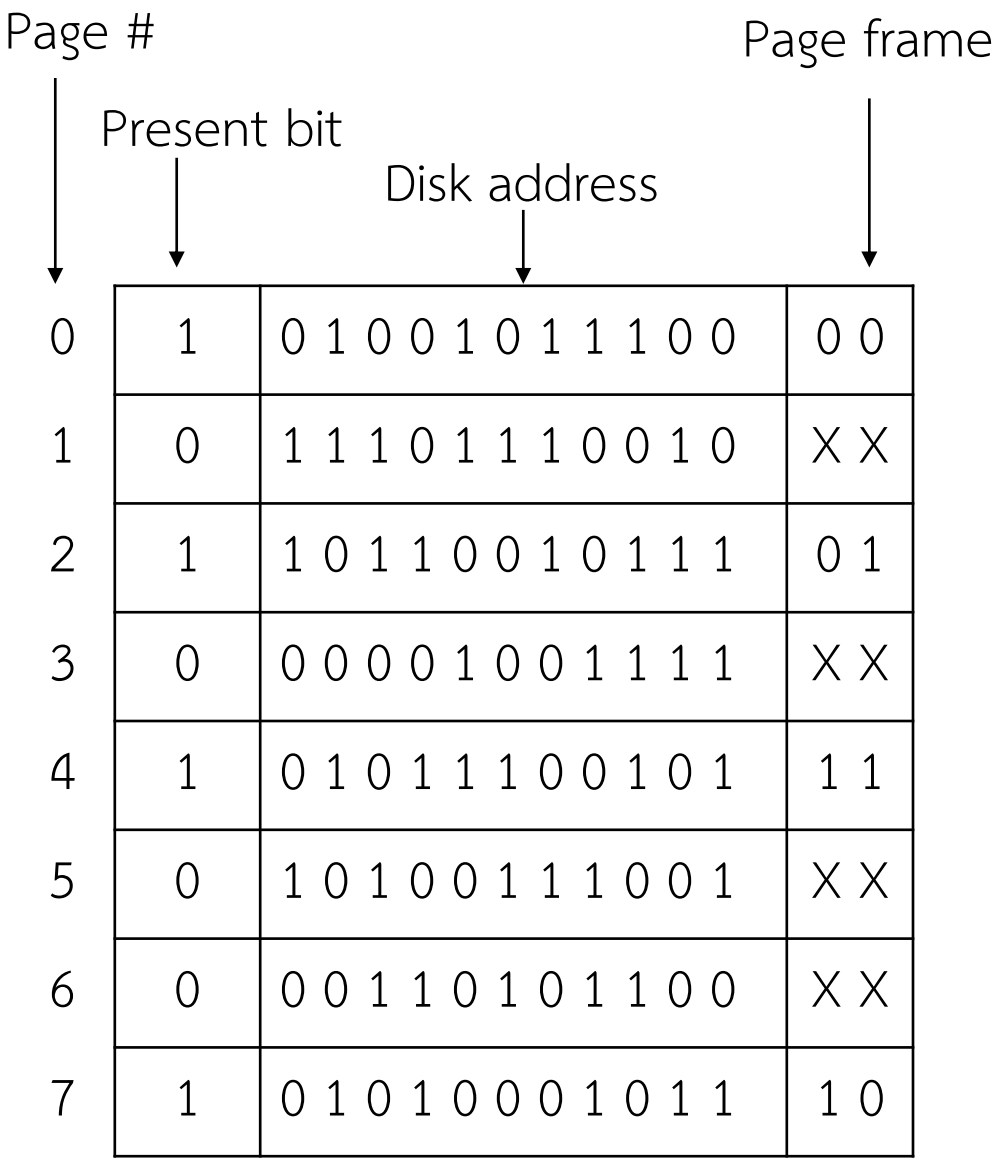
ตารางหน้า (Page table)

- **Present bit** มีขนาด 1 บิต ทำหน้าที่แสดงว่าหน้าดังกล่าวอยู่ในหน่วยความจำหลักหรือไม่ ถ้ามีการพยายามเข้าถึงข้อมูลของหน้าที่บิตนี้กำหนดเป็น 0 แล้วจะทำให้เกิดปัญหาการผิดหน้า (Page fault) ได้
- **Disk address** จะเป็นตัวชี้ (Pointer) ไปยังตำแหน่งที่หน้าดังกล่าวถูกเก็บไว้ในดิสก์
- **Page frame** มีความสำคัญมากที่สุด เพราะจะกำหนดว่าหน้าเฟรมนี้จะไปอยู่กับเฟรมเลขที่เท่าใดในหน่วยความจำหลัก และสำหรับหน้าที่ไม่ได้อยู่ในหน่วยความจำหลัก ฟิลด์นี้จะเก็บค่าที่ไม่สื่อความหมาย ที่เราจะเขียนค่าเป็น XX

Present bit

0: Page is not in physical memory

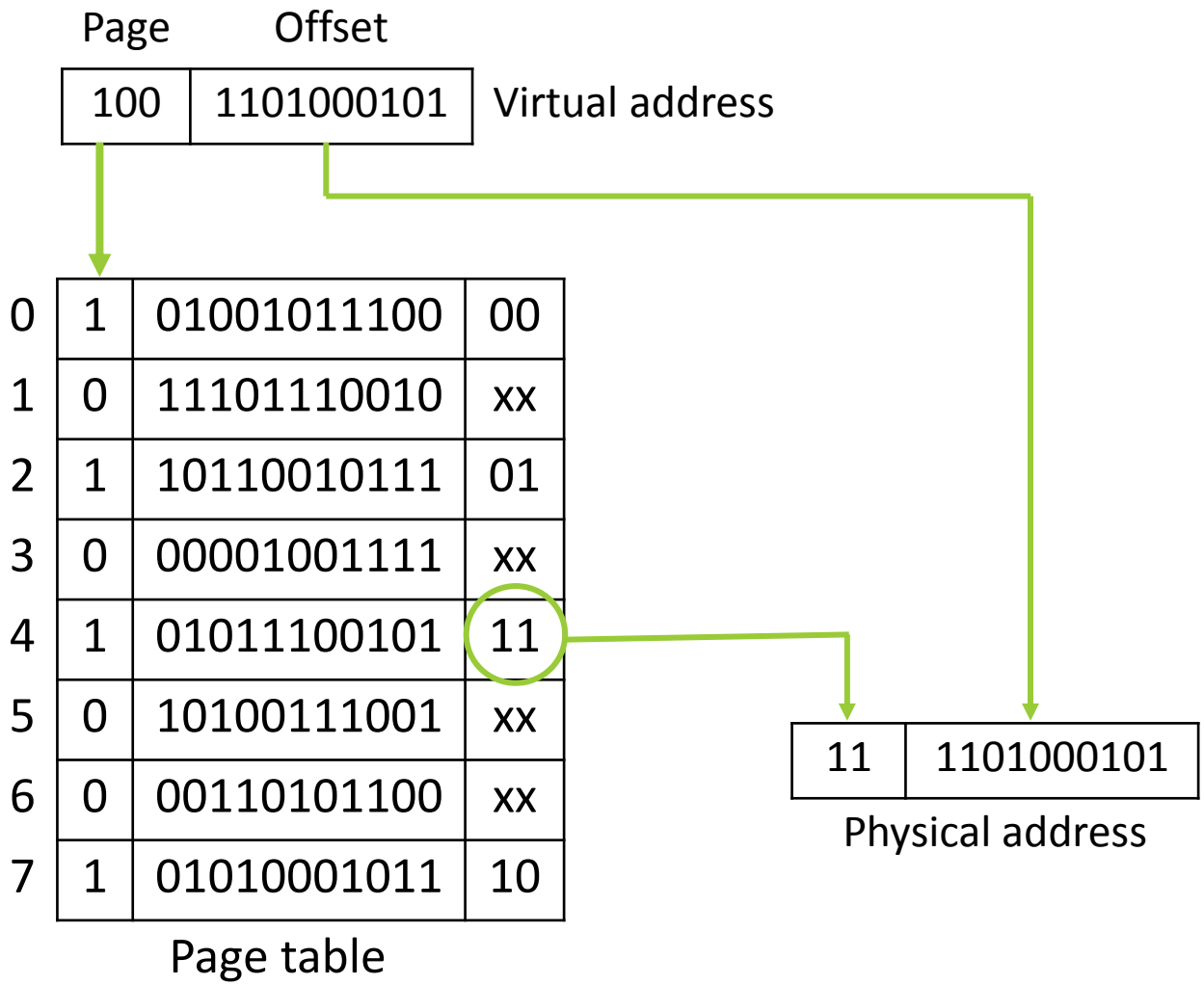
1: Page is in physical memory



การแปลงค่าแอดเดรส

การแปลงค่าแอดเดรสทางตรรกะไปเป็นแอดเดรสจริงทางกายภาพ จะใช้บิตทั้งสองของ Page frame จากตารางหน้า

โดยวางบิตดังกล่าวติดไปทางซ้ายของ 10 บิต ที่เหลือ ซึ่งจะรวมกันเป็นแอดเดรสทางกายภาพสำหรับใช้อ้างถึง


Page
Offset
100
1101000101
Virtual address
0
1
01001011100
00
1
0
11101110010
xx
2
1
10110010111
01
3
0
00001001111
xx
4
1
01011100101
11
5
0
10100111001
xx
6
0
00110101100
xx
7
1
01010001011
10
Page table
11
1101000101
Physical address

หน้าที่ต้องการ (Demand page)

การโหลดโปรแกรมเข้าไปใช้งานในหน่วยความจำนั้นบางทีก็ใช้เวลานาน และบางครั้งโปรแกรมอาจจะยังไม่ถูกเรียกใช้งานทั้งหมดก็ได้

ทำให้เวลาที่ต้องการเพื่อโหลดโปรแกรมจากดิสก์เข้าสู่หน่วยความจำสามารถลดลงได้โดยการโหลดข้อมูลเฉพาะส่วนของโปรแกรมที่ต้องการสำหรับช่วงระยะเวลาหนึ่งเท่านั้น

วิธีการโหลด “หน้าที่ต้องการ” นั้นจะไม่ทำการโหลดเข้าสู่หน่วยความจำหลัก จนกระทั่งมีการ “ผิดหน้า” เกิดขึ้น

หลังจากที่รันโปรแกรมไปได้สักระยะหนึ่ง จะมีเพียงหน้าที่กำลังถูกใช้งานเท่านั้นที่จะอยู่ในหน่วยความจำหลัก (Working set)

บัฟเฟอร์ค้นหาที่อยู่

Translation Look aside Buffer

โดยทั่วไปแล้วเมื่อมีการอ้างถึงหน่วยความจำเสมือนทุกๆ ครั้ง จะมีการเรียกหาหน่วยความจำหลักถึง 2 ครั้ง คือมีการดึงแถวของตารางหน้า และการดึงข้อมูลที่ต้องการออกมาจากหน่วยความจำหลัก

ดังนั้นการทำงานของหน่วยความจำเสมือนจะใช้เวลาในการเข้าถึงหน่วยความจำเป็น 2 เท่า

วิธีการป้องกันปัญหาดังกล่าวทำได้โดยใช้บัฟเฟอร์หรือหน่วยความจำพิเศษในการเก็บแถว

การทำงานของ TLB

เมื่อมีการให้ค่าแอดเดรสเสมือนกับระบบ ซีพียูจะทำการสืบค้นในตาราง TLB ก่อน

- ถ้าพบแถวที่ต้องการในตารางหน้า (TLB hit) เราก็สามารถเก็บเลขที่เฟรมและแปลงเป็นแอดเดรสจริงได้เลย
- ถ้าไม่พบแถวที่ต้องการ (TLB miss) ซีพียูก็จะใช้เลขที่หน้าไปเปรียบเทียบกับอินเด็กซ์ของตารางหน้าเพื่อหาแถวที่ต้องการ และซีพียูก็จะเพิ่ม TLB ให้มีแถวของตารางหน้าใหม่นี้ด้วย

เนื่องจาก TLB จะเก็บข้อมูลของบางแถวและจากตารางหน้าเท่านั้น ทำให้เราไม่สามารถเรียงลำดับแถวของ TLB ด้วยเลขที่หน้าได้ ดังนั้นค่าของฟิลด์ในแถวของ TLB จะต้องมีเลขที่หน้ากำกับด้วย

การค้นหาเลขที่หน้าใน TLB ของซีพียูนั้นจะเป็นการเชื่อมโยงแบบสัมพันธ์ (Associative mapping) ซึ่งจะแตกต่างจากการเชื่อมโยงโดยตรงอย่างที่ทำกันในตารางหน้า

6.7 การสับเปลี่ยนหน้า

Page Replacement Algorithm

แนวความคิด

- มีจำนวนเฟรมเท่าไหร่ที่สามารถให้กับโปรเซสเซอร์แต่ละตัวได้
- กลุ่มของหน้าที่ถูกพิจารณาให้สับเปลี่ยนหน้านั้นคงรจะถูกจำกัดหรือไม่ เพื่อป้องกันไม่ให้เกิดการผิดหน้า (Page fault) ทำงาน
- ในกลุ่มของหน้าที่เราพิจารณา เราจะเลือกหน้าใดที่จะนำไปสับเปลี่ยน

ข้อควรพิจารณา

- ถ้าเราให้หน่วยความจำแก่โปรเซสเซอร์ใดๆ น้อยเท่าไหร่ ก็จะทำให้มีจำนวนโปรเซสเข้ามาใช้งานหน่วยความจำมากขึ้นเท่านั้น ซึ่งจะทำให้ความน่าจะเป็นของระบบที่จะสามารถค้นพบโปรเซสอย่างน้อยหนึ่งโปรเซสที่พร้อมที่จะทำงานมากขึ้น และจะทำให้เราเสียเวลาในการสลับหน้าน้อยลง
- ถ้ามีจำนวนหน้าของโปรเซสหนึ่งๆ ที่สามารถอยู่ในหน่วยความจำหลักน้อย (จำนวนเฟรมน้อย) ก็จะทำให้อัตราการเกิดการผิดหน้าสูงมากขึ้น
- นอกเหนือจากการกำหนดขนาดของหน้าคงที่แล้ว การให้หน่วยความจำเพิ่มกับโปรเซสใดโปรเซสหนึ่งจะไม่มีผลกระทบต่ออัตราการผิดหน้ามาก

วิธีสับเปลี่ยนแบบมาก่อน-ออกก่อน

First-in First-out Algorithm

เป็นวิธีการสับเปลี่ยนหน้าที่ง่ายที่สุด

วิธีการนี้จะใช้เวลาที่หน้านั้นๆ ถูกนำเข้ามาในหน่วยความจำหลักเป็นเกณฑ์ในการตัดสินใจ

เมื่อต้องการเลือกหน้าบางหน้าออก ก็ให้เลือกจากหน้าที่เข้ามานานที่สุดนั่นเอง

ในทางปฏิบัติเราอาจจะไม่ต้องจดเวลาจริงๆ ที่หน้านั้นเข้ามาใช้งานก็ได้ เพียงแต่สร้างแถวรอคอยแบบมาก่อน-ออกก่อน (FIFO Queue) สำหรับเก็บหมายเลขหน้าที่อยู่ในหน่วยความจำ

เมื่อมีการนำหน้าใหม่เข้ามาให้เอาหมายเลขมาไว้ที่ปลายแถว แล้วเลือกหน้าออกจากหัวแถว

วิธีสับเปลี่ยนแบบให้โอกาสครั้งที่สอง

Second Chance Page Replacement Algorithm

เราสามารถปรับปรุงวิธีแบบ FIFO ได้โดยการป้องกันการเปลี่ยนหน้าที่ถูกเรียกใช้งานบ่อยออกไป

สามารถทำได้โดยการตรวจสอบที่บิต Referenced (R) ของหน้าที่เข้ามานานที่สุด

- ถ้าบิต R มีค่าเป็น 0 ก็แสดงว่าหน้านั้นเก่าและไม่ได้ถูกเรียกใช้งานเลย ระบบก็สามารถทำการสับเปลี่ยนได้ทันที
- ถ้าบิต R มีค่าเท่ากับ 1 ก็กำหนดให้บิต R นั้นเป็น 0 และนำหน้านั้นกลับไปเข้าแถวใหม่อีกครั้ง พร้อมกับทำการเปลี่ยนแปลงเวลาของหน้านั้นใหม่เหมือนดังหน้านั้นเพิ่งเข้ามาในหน่วยความจำ จากนั้นก็ทำการค้นหาหน้าที่จะถูกสับเปลี่ยนต่อไป

วิธีทำงานของวิธีการนี้คือการค้นหาหน้าที่เก่าที่สุด และไม่ได้ถูกอ้างอิงหรือเรียกใช้งาน แต่ถ้าทุกหน้าถูกเรียกใช้งานหมด วิธีการแบบนี้ก็จะกลายเป็นวิธีการแบบ FIFO นั่นเอง

วิธีสับเปลี่ยนแบบวงรอบนาฬิกา

Clock Page Replacement Algorithm)

เป็นการเรียงเฟรมทุกเฟรมเป็นรูปวงกลมให้เหมือนนาฬิกา และมีเข็มนาฬิกาชี้ไปหน้าไปที่เก่าที่สุด

เมื่อมีการผิดหน้าเกิดขึ้น หน้าที่มีเข็มนาฬิกาจะถูกตรวจสอบ

- ถ้าบิต R มีค่าเป็น 0 หน้านั้นก็จะถูกสับเปลี่ยนออกไป และหน้าใหม่ก็จะถูกใส่เข้ามาในตำแหน่งเดิม พร้อมกันนั้นเข็มนาฬิกาก็จะทำการเลื่อนไปด้านหน้า 1 ตำแหน่ง
- ถ้าบิต R ถูกกำหนดให้เป็น 1 ก็ให้ลบค่าของบิตนั้นเป็น 0 และเลื่อนเข็มไปหน้าถัดไป วิธีการดังกล่าวจะถูกทำซ้ำจนกว่าเราจะได้หน้าที่มีบิต R เป็น 0

วิธีการนี้จะเหมือนวิธีการแบบให้โอกาสครั้งที่สอง เพียงแต่แตกต่างกันไปในวิธีการใช้งานเท่านั้น

วิธีสับเปลี่ยนแบบที่ดีที่สุด

Optimal Page Replacement Algorithm

วิธีการสับเปลี่ยนหน้าที่ดีที่สุดเรียกว่า OPT หรือ MIN ซึ่งมีวิธีการคือ “ให้เลือกสับเปลี่ยนหน้าที่จะไม่ถูกเรียกใช้งาน และมีระยะเวลารอการเรียกใช้ที่นานที่สุด”

การใช้วิธีนี้จะทำให้อัตราการผิดหน้าต่ำที่สุดสำหรับพื้นที่ที่มีจำนวนเฟรมหนึ่ง ๆ ซึ่งดีกว่าวิธีแบบ FIFO มาก

แต่วิธีแบบที่ดีที่สุดนี้ยากที่จะสร้างได้จริงๆ เพราะเราต้องรู้ในอนาคตว่าจะมีการเรียกหน้าใดเมื่อไหร่ ดังนั้นวิธีแบบที่ดีที่สุดนี้จึงมีไว้เพื่อใช้ในการเปรียบเทียบเท่านั้น

การสับเปลี่ยนแบบที่ไม่ได้ใช้งานออกก่อน

Not Recently Used (NRU)

เราสามารถนำวิธีการสับเปลี่ยนแบบวงรอบนาฬิกามาเพิ่มประสิทธิภาพในการทำงานโดยการพิจารณาบิตที่สำคัญในตารางหน้าเพิ่มขึ้นมาอีกหนึ่งบิต

ระบบสามารถทำการเก็บข้อมูลสถิติว่าหน้าใดที่กำลังถูกใช้หรือไม่ได้ใช้งานได้จากบิตในแถวของตารางหน้าที่แสดงสถานะการทำงานของแต่ละบิต

บิต R จะถูกกำหนดเป็น 1 เมื่อหน้านั้นถูกเรียกใช้งาน และบิต Modified (M) จะถูกกำหนดเป็น 1 เมื่อหน้านั้นมีการเปลี่ยนแปลง

บิตทั้งสองเป็นฟิลด์ที่อยู่ในแถวของตารางหน้าและจะถูกเปลี่ยนแปลงค่าทุกครั้งเมื่อมีการเรียกใช้งานหรือถูกอ้างอิงถึง

การกำหนดค่าต่าง ๆ ควรเป็นหน้าที่ฮาร์ดแวร์ เมื่อบิตถูกกำหนดเป็น 1 เมื่อใดบิตนั้นก็จะมีค่าเท่ากับ 1 จนกว่าระบบปฏิบัติการจะกำหนดกลับเป็น 0 โดยใช้ซอฟต์แวร์

การสับเปลี่ยนแบบที่ไม่ได้ใช้งานออกก่อน

Not Recently Used (NRU)

บิต R และ M สามารถนำมาใช้ในการสร้างวิธีการสับเปลี่ยนหน้าแบบใหม่ โดยเมื่อโปรแกรมหนึ่ง ๆ เริ่มทำงาน บิตทั้งสองในทุกๆ หน้าจะถูกกำหนดให้เป็น 0 โดยระบบปฏิบัติการ

เมื่อครบวงรอบของระยะเวลาหนึ่ง ๆ เช่น จากการแทรกของอินเทอร์รัพต์ บิต R ก็จะถูกทำความสะอาด หรือถูกกำหนดค่าเป็น 0 เพื่อเป็นตัวแยกหน้าที่ไม่ได้ถูกเรียกใช้งานออกจากหน้าอื่น ๆ

เมื่อมีการผิดหน้าเกิดขึ้น ระบบปฏิบัติการจะทำการตรวจสอบทุกหน้าและแบ่งหน้าเหล่านั้นออกเป็น 4 กลุ่ม

กลุ่มที่ 0: ไม่ถูกเรียกใช้งาน และไม่มีการเปลี่ยนแปลงค่า

กลุ่มที่ 1: ไม่ถูกเรียกใช้งาน แต่มีการเปลี่ยนแปลงค่า

กลุ่มที่ 2: ถูกเรียกใช้งาน แต่ไม่มีการเปลี่ยนแปลงค่า

กลุ่มที่ 3: ถูกเรียกใช้งาน และมีการเปลี่ยนแปลงค่า

การสับเปลี่ยนแบบที่ไม่ได้ใช้งานออกก่อน

Not Recently Used (NRU)

วิธีการนี้จะสุ่มเอาหน้าออกจากกลุ่มลำดับต่ำที่สุด ที่มีหน้าของโปรเซสอยู่

นอกจากนั้นวิธีการนี้ยังดูเหมือนว่าจะพยายามที่จะนำเอาหน้าที่ถูกเปลี่ยนแปลงแต่ไม่ได้ถูกเรียกใช้งานหรืออ้างอิงถึง ออกทุกครั้งที่มีการครบรอบวงนาฬิกา มากกว่าหน้าที่ไม่มีการเปลี่ยนแปลงแต่ถูกเรียกใช้งานบ่อย

ข้อเด่นของวิธี NRU คือง่ายที่จะทำความเข้าใจ ไม่ยากเกินไปที่จะนำมาใช้งานจริง และมีประสิทธิภาพค่อนข้างดี

เป็นวิธีการที่ใช้งานในระบบปฏิบัติการของเครื่อง Macintosh

การสับเปลี่ยนแบบใช้งานน้อยที่สุดออกก่อน

Least Recently Used

ใช้วิธีบันทึกเวลาที่แต่ละหน้าถูกอ้างอิงครั้งล่าสุดไว้ เมื่อต้องการเลือกหน้าเพื่อทำการสับเปลี่ยนออก ก็จะเลือกหน้าที่ไม่ได้ใช้มาเป็นเวลานานที่สุด (หน้าที่มีตัวเลขเวลาน้อยที่สุด)

แม้ว่า LRU น่าจะใช้งานได้ดีในทางทฤษฎี แต่ก็ไม่ง่ายในทางปฏิบัติ เนื่องจากการใช้ LRU นั้นระบบจะต้องสร้างลิงค์ลิสต์ของทุก ๆ หน้าในหน่วยความจำที่มีหน้าที่ใช้งานน้อยอยู่ที่หัวแถว และหน้าที่การใช้งานบ่อยอยู่ท้ายสุด

ความยากของวิธีนี้อยู่ที่ว่าลิสต์จะต้องทำการปรับเปลี่ยนทุกครั้งที่มีการอ้างอิงถึง หรือมีการเรียกใช้งานในหน่วยความจำ ซึ่งการค้นหาหน้าในลิสต์ การทำการลบ หรือการย้ายหน้าไปมานั้น ล้วนแต่เป็นการทำงานที่ต้องใช้เวลาของซีพียูทั้งสิ้น

6.8 การแบ่งเป็นเซ็กเมนต์

Segmentation

การแบ่งเป็นเซ็กเมนต์

Segmentation

จะแบ่งโปรเซสหรือโปรแกรมออกเป็นส่วนๆ โดยแต่ละส่วนไม่จำเป็นต้องมีความยาวเท่ากัน (แต่ก็อาจจะมีการกำหนดให้อยู่ภายใต้ขนาดที่ใหญ่ที่สุดที่ระบบกำหนด)

เช่นเดียวกับการแบ่งเป็นหน้า การแปลงแอดเดรสทางตรรกะเป็นแอดเดรสจริงในหน่วยความจำจะต้องใช้ข้อมูล 2 ส่วน คือเลขที่ของเซ็กเมนค์และระยะออปเซ็ทจากเซ็กเมนต์นั้น

การแบ่งเป็นเซ็กเมนต์ทำให้เกิดชิ้นส่วนที่ไม่เท่ากัน ซึ่งจะเหมือนการแบ่งหน่วยความจำแบบเปลี่ยนแปลงขนาดได้ (Dynamic partitioning) แต่ข้อแตกต่างคือโปรแกรมหนึ่ง ๆ อาจจะมีหลายเซ็กเมนต์ที่สามารถใช้งานในหลายๆ พาร์ติชั่นได้และไม่จำเป็นต้องอยู่ติดกันด้วย

การแบ่งเป็นเซ็กเมนต์
Segmentation

การแบ่งเป็นเซ็กเมนต์นี้จะช่วยลดปัญหาการสูญเสียพื้นที่ภายใน (Internal fragmentation) ได้ด้วย

การแบ่งเป็นหน้าโปรแกรมเมอร์จะไม่สามารถมองเห็นและมีส่วนร่วมได้ แต่การแบ่งเป็นเซ็กเมนต์โปรแกรมเมอร์สามารถมองเห็นและมีส่วนร่วมในการจัดการกับโปรแกรมและข้อมูลของตนได้

โดยปกติโปรแกรมเมอร์หรือคอมไพเลอร์จะทำการแบ่งโปรแกรมและข้อมูลออกเป็นส่วน ๆ เอง ผลของการแบ่งหน่วยความจำออกเป็นส่วนๆ ที่ไม่เท่ากัน จึงทำให้ความสัมพันธ์ระหว่างแอดเดรสทางตรรกะกับแอดเดรสจริงเป็นความสัมพันธ์ที่ค่อนข้างจะซับซ้อน

ตารางของเซ็กเมนต์
Segmentation table

ใช้เก็บรายละเอียดของแต่ละโปรเซสและแอดเดรสเริ่มต้นของที่ว่างในหน่วยความจำ

นอกจากจะเก็บจุดเริ่มต้นของเซ็กเมนต์ภายในแถวของตารางแล้ว เราจะต้องเก็บค่าความยาวของเซ็กเมนต์นั้น ๆ ด้วย เพื่อเป็นการยืนยันว่าระบบจะไม่ใช้แอดเดรสที่ไม่ถูกต้อง

ข้อดีการนำวิธีการแบ่งเป็นเซ็กเมนต์มาใช้ในหน่วยความจำเสมือน

การจัดการกับการเพิ่มโครงสร้างของข้อมูลในโปรแกรมสามารถทำได้ง่าย ถ้าโปรแกรมเมอร์ไม่สามารถทราบในอนาคตได้ว่าโครงสร้างของข้อมูลจะมีขนาดเป็นเท่าไหร่ และถ้าเราไม่สามารถทำการแบ่งพื้นที่แบบที่เปลี่ยนแปลงขนาดได้ เราก็จะต้องทำการประมาณขนาดของข้อมูลก่อน แต่การใช้วิธีการแบ่งเป็นเซ็กเมนต์กับหน่วยความจำเสมือนนั้น โครงสร้างข้อมูลจะถูกกำหนดให้อยู่ในส่วนของตน และระบบปฏิบัติการจะทำการขยายหรือย่อขนาดของส่วนนั้นได้เมื่อต้องการ ถ้าส่วนหรือเซ็กเมนต์นั้นต้องการขยายในหน่วยความจำหลัก แต่ไม่มีพื้นที่เพียงพอ ระบบปฏิบัติการอาจจะทำการย้ายส่วนนั้นไปยังพื้นที่ในหน่วยความจำที่มีขนาดใหญ่ขึ้น

ข้อดีการนำวิธีการแบ่งเป็นเซ็กเมนต์มาใช้ในหน่วยความจำเสมือน

วิธีการนี้สามารถทำให้เราเปลี่ยนแปลง หรือคอมไพล์โปรแกรมแยกออกเป็นส่วน ๆ ได้ โดยไม่จำเป็นที่จะต้องให้โปรแกรมย่อย ๆ เหล่านั้นถูกโหลดขึ้นมาใหม่หรือเชื่อมโยงกัน

การแบ่งเป็นเซ็กเมนต์นี้จะอนุญาตให้มีการแบ่งปันข้อมูลระหว่างโปรเซสหลาย ๆ โปรเซส เช่น โปรแกรมเมอร์สามารถนำโปรแกรมยูทิลิตี้ หรือข้อมูลที่เป็นประโยชน์ใส่ลงในส่วนของเซ็กเมนต์หนึ่ง ๆ ที่สามารถอ้างอิงโดยโปรเซสอื่น ๆ ได้

การแบ่งแบบนี้จะอนุญาตให้มีการป้องกันข้อมูลด้วยตัวเองได้ เพราะในส่วนของเซ็กเมนต์หนึ่ง ๆ นั้นจะถูกสร้างขึ้นมาเพื่อใส่โปรแกรม หรือข้อมูลที่ได้รับการนิยามมาอย่างดีแล้ว ซึ่งโปรแกรมเมอร์ที่เป็นผู้ดูแลระบบสามารถกำหนดสิทธิในการเข้าถึงข้อมูลในส่วนนั้นได้

เปรียบเทียบวิธีการแบ่งเป็นหน้ากับการแบ่งเป็นเซ็กเมนต์

ข้อดีของการแบ่งเป็นหน้า

- การปกปิดไม่ให้โปรแกรมเมอร์มองเห็นการทำงานของมัน การป้องกันการสูญเสียพื้นที่ภายนอก และจะทำให้ระบบใช้หน่วยความจำหลักอย่างมีประสิทธิภาพ
- เนื่องจากหน้าของโปรเซสที่ถูกสลับเข้าและออกจากหน่วยความจำนั้นมีขนาดคงที่ และเท่ากัน จึงมีความเป็นไปได้ที่เราจะพัฒนาอัลกอริทึ่มที่ซับซ้อนเพื่อนำไปใช้ในการจัดการกับหน่วยความจำ

ข้อดีของการแบ่งเป็นเซ็กเมนต์

- โปรแกรมเมอร์สามารถมองเห็นการทำงานของเซ็กเมนต์ได้ มีความสามารถในการจัดการกับโครงสร้างข้อมูลที่มีขนาดไม่คงที่ การทำโปรแกรมเป็นโมดูล และการสนับสนุนการแบ่งปันและการป้องกันเซ็กเมนต์

การรวมข้อดีของทั้งสองวิธีนั้น ในบางระบบจะใช้ฮาร์ดแวร์เป็นตัวช่วย และซอฟต์แวร์ของระบบปฏิบัติการจะต้องทำได้ทั้งสองวิธี

การรวมวิธีการแบ่งเป็นหน้ากับการแบ่งเป็นเซ็กเมนต์เข้าด้วยกัน

การรวมข้อดีของทั้งสองวิธีนั้น ในบางระบบจะใช้ฮาร์ดแวร์เป็นตัวช่วย และซอฟต์แวร์ของระบบปฏิบัติการจะต้องทำได้ทั้งสองวิธี

ในระบบที่มีการทำงานร่วมกัน พื้นที่ใช้งานของผู้ใช้จะถูกแบ่งออกเป็นเซ็กเมนต์จำนวนหนึ่งตามที่โปรแกรมเมอร์กำหนด และในแต่ละเซ็กเมนต์ก็จะถูกแบ่งออกเป็นหน้า ๆ ที่มีขนาดคงที่ และมีความยาวเท่ากับเฟรมในหน่วยความจำ ถ้าเซ็กเมนต์ใดมีความยาวน้อยกว่าขนาดของหน้า เซ็กเมนต์นั้นก็มีแค่ 1 หน้า

จากมุมมองของโปรแกรมเมอร์นั้น แอดเดรสทางตรรกะจะประกอบไปด้วยเลขที่ของเซ็กเมนต์ และระยะห่างจากขอบของเซ็กเมนต์

มุมมองของระบบ ระยะห่างจากขอบของเซ็กเมนต์ก็จะประกอบไปด้วยเลขที่หน้า และระยะห่างจากขอบหน้าภายในเซ็กเมนต์นั้นนั่นเอง