แผนการบริหารการสอนประจำบทที่ 8

เนื้อหาประจำบท

บทที่ 8 หน่วยความจำเสมือน

1. การจัดการและจัดสรรหน่วยความจำเสมือน
2. การสลับหน้าตามคำขอ (demand paging) ของหน่วยความจำเสมือน
3. การเลือกอัลกอริทึมและประสิทธิภาพของการสับเปลี่ยนหน้า

จุดประสงค์เชิงพฤติกรรม

เมื่อศึกษาบทที่ 8 แล้วนักศึกษาสามารถ

1. สามารถอธิบายถึงประโยชน์ของระบบหน่วยความจำเสมือน
2. เพื่อเข้าใจการสลับหน้าตามคำขอของหน่วยความจำเสมือน
3. เพื่อเข้าใจถึงความซับซ้อนและค่าใช้จ่ายในการใช้งานหน่วยความจำเสมือน

กิจกรรมการเรียนการสอนประจำบท

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

สื่อการเรียนการสอน

1. สื่ออิเล็กทรอนิกส์ประกอบการสอนวิชาระบบปฏิบัติการ
2. เอกสารประกอบการสอนวิชาระบบปฏิบัติการ
3. หนังสืออ่านประกอบค้นคว้าเพิ่มเติม

การวัดผลและประเมินผล

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

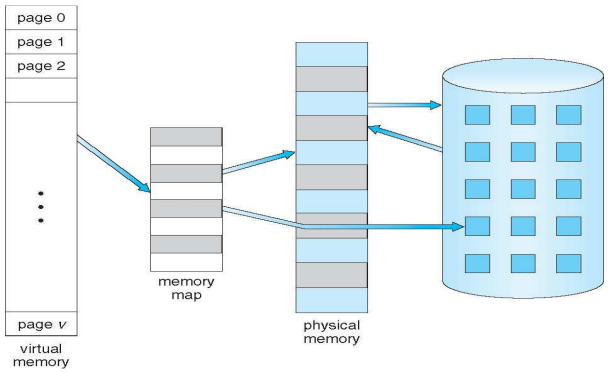
บทที่ 8
หน่วยความจำเสมือน

8.1 แนวคิดหน่วยความจำเสมือน

หน่วยความจำเสมือนเป็นวิธีหนึ่งที่สามารถทำให้โพรเซสทำงานได้ ถึงแม้ว่าโพรเซสนั้นจะไม่ได้อยู่ในหน่วยความจำหลักทั้งหมด โดยระบบปฏิบัติการจะทำหน้าที่เก็บบางส่วนของโปรแกรมที่กำลังทำงานไว้ในหน่วยความจำหลัก และเก็บส่วนที่เหลือไว้ในฮาร์ดดิสก์ ตัวอย่างเช่น โปรแกรมที่ต้องการหน่วยความจำ 100 MB สามารถทำงานบนเครื่องที่มีหน่วยความจำ 64 MB ได้โดยการเลือกเอาบางส่วนของโปรแกรมขนาด 64 MB เข้ามาทำงานในหน่วยความจำหลัก และบางส่วนเก็บไว้ในฮาร์ดดิสก์และทำการสลับเปลี่ยนไปมาเมื่อมีบางส่วนของโปรแกรมต้องการทำงาน

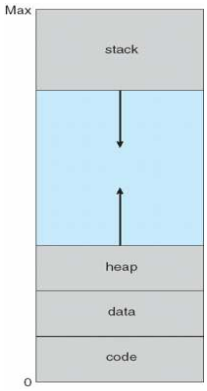
ข้อดีของวิธีนี้ คือ โปรแกรมของผู้ใช้สามารถมีขนาดใหญ่กว่าหน่วยความจำจริงได้ เพราะวิธีนี้จะทำการแยกส่วนของหน่วยความจำเชิงตรรกะ (Logical memory) ของผู้ใช้ออกจากหน่วยความจำเชิงกายภาพ (Physical memory) มีเพียงส่วนของโปรแกรมที่ต้องการอยู่ในหน่วยความจำเพื่อดำเนินการเท่านั้น พื้นที่ตำแหน่งเชิงตรรกะ (Logical address space) จึงสามารถมีขนาดใหญ่กว่าขนาดของพื้นที่หน่วยความจำเชิงกายภาพ (Physical address space) และยินยอมให้มีการใช้พื้นที่หน่วยความจำร่วมกันได้จากหลาย ๆ โพรเซส ทำให้มีการสร้างโพรเซสขึ้นมาได้โดยสะดวก

ในระบบคอมพิวเตอร์แบบ 32 บิต และ 64 บิต ตำแหน่งที่อ้างอิงได้ที่เป็นตำแหน่งของ Virtual address หรือประมาณ 4 กิกะไบต์ (4 พันล้านตำแหน่ง) ส่วนขอบเขตของ Physical address คือ จำนวนหน่วยความจำที่มีในเครื่อง สำหรับเครื่องทั่วไปจะมีหน่วยความจำสูงสุดได้ 2 กิกะไบต์ (2 พันล้านตำแหน่ง) หรือต่ำกว่านั้น เช่น 256 เมกะไบต์ (2.5 ร้อยล้านตำแหน่ง)



ภาพที่ 8.1 ไดอะแกรมแสดงหน่วยความจำเสมือนที่มีขนาดใหญ่กว่าหน่วยความจำหลักทางกายภาพ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.399)

พื้นที่หน่วยความจำเสมือน (Virtual address space) ของโพรเซสที่จะกล่าวถึงหน่วยความจำเสมือน ในมุมมองของโพรเซสเป็นที่เก็บในหน่วยความจำได้อย่างไร ในมุมมองนี้โพรเซสจะเริ่มต้นที่ตำแหน่งที่อยู่ที่แน่นอน คือ ตำแหน่งที่ 0 และอยู่ในหน่วยความจำที่ติดกัน ดังแสดงในภาพที่ 8.2



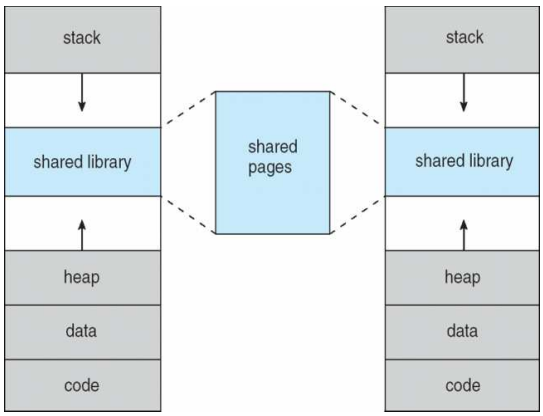
ภาพที่ 8.2 พื้นที่ว่างในหน่วยความจำเสมือน
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.399)

ในการแยกหน่วยความจำเชิงตรรกะออกจากหน่วยความจำเชิงกายภาพ หน่วยความจำเสมือนยอมให้แฟ้มข้อมูลและหน่วยความจำถูกแบ่งได้โดยสองโพรเซสหรือมากกว่านั้น และนำไปสู่การได้รับประโยชน์ที่จะตามมาดังนี้

1) ระบบ Libraries สามารถใช้พื้นที่ร่วมกัน โดยรวมหลาย ๆ โพรเซสเข้าด้วยกันโดยการแชร์วัตถุไปยังพื้นที่หน่วยความจำเสมือน ถึงแม้ว่าจะพิจารณาแต่ละโพรเซสให้แชร์ Libraries ในส่วนของพื้นที่หน่วยความจำเสมือน สิ่งที่ต้องการคือ Libraries ที่อยู่ในหน่วยความจำเชิงกายภาพ ถูกแชร์ให้กับโพรเซสทั้งหมด ดังแสดงในภาพที่ 8.3

2) หน่วยวามจำเสมือนอย่างง่าย ต้องทำให้โพรเซสสามารถแชร์หน่วยวามจำเสมือน โดยในหนึ่งโพรเซสสามารถสร้างพื้นที่ของหน่วยความจำ มันสามารถแชร์กับโพรเซสอื่น ๆ ได้ การแชร์โพรเซสจำเป็นจะต้องพิจารณาพื้นที่ในบางส่วนของพื้นที่หน่วยความจำเสมือน

3) หน่วยความจำเสมือนสามารถยอมให้พื้นที่ที่ต้องการแชร์ ระหว่างที่โพรเซสถูกสร้างจากฟังก์ชั่น the fork() และเรียกผ่าน System call ดังนั้นโพรเซสจะสามารถถูกสร้างได้เร็ว



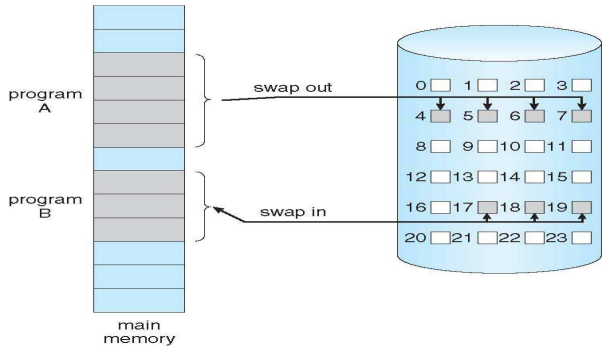
ภาพที่ 8.3 แชร์ Library ในหน่วยความจำเสมือน
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.400)

8.2 การจัดสรรหน้าตามคำร้องขอ

เพจจะถูกนำมาไว้ในหน่วยความจำก็ต่อเมื่อต้องการทำให้เพจสามารถทำงานได้ แม้ว่าจะเหลือพื้นที่หน่วยความจำไม่มาก โดยใช้แนวคิดที่ว่า โปรแกรมหรืองานจะมีการทำงานตามลำดับ เพราะฉะนั้นจึงไม่จำเป็นต้องนำทุก ๆ เพจของงานมาไว้ในหน่วยความจำที่เดียว

เมื่อต้องการใช้โพรเซสระบบจะย้ายเฉพาะหน้าที่ ๆ ต้องการเข้ามาในหน่วยความจำหลัก โดยใช้ตัวสับเปลี่ยน (Pager) แต่แทนที่จะย้ายเข้ามาทั้งตัว เราจะใช้ Lazy Swapper ซึ่งจะย้ายเฉพาะเมื่อมีการร้องขอและย้ายเข้ามาเฉพาะหน้าที่ร้องขอเท่านั้น ดังแสดงในภาพที่ 8.4 เพจจะไม่ถูกนำเข้ามาในหน่วยความจำถ้าไม่มีความจำเป็นที่จะใช้เพจนั้น Swapper มักใช้กรณีจัดการสลับโพรเซส (เข้าออกหน่วยความจำ) แต่ในการนำเพจเข้าหน่วยความจำ (Swapped in) และนำออกจากหน่วยความจำ (Swapped out) เราจะใช้ ตัวสับเปลี่ยน ในการนำเฉพาะเพจที่ต้องการใช้เข้ามาไว้ในหน่วยความจำ มีข้อดีคือ

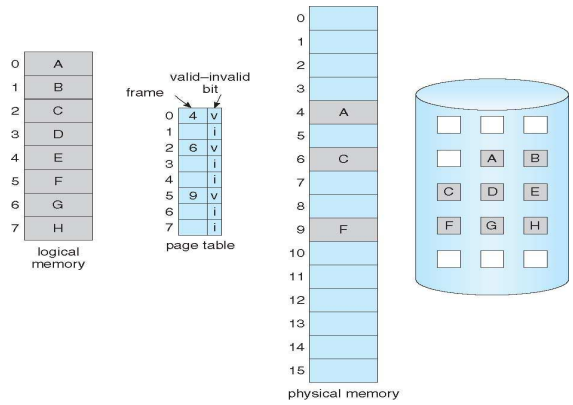
1) ลดการใช้ (Less I/O needed)
2) ลดการใช้หน่วยความจำ (Less memory needed)
3) โต้ตอบได้รวดเร็วกว่า (Faster response)
4) รองรับผู้ใช้ได้มากกว่า (More users)



ภาพที่ 8.4 แสดงการย้ายเพจในหน่วยความจำไปที่พื้นที่เก็บข้อมูลต่อเนื่อง
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.401)

8.2.1 หลักการขั้นพื้นฐาน (Basic Concepts)

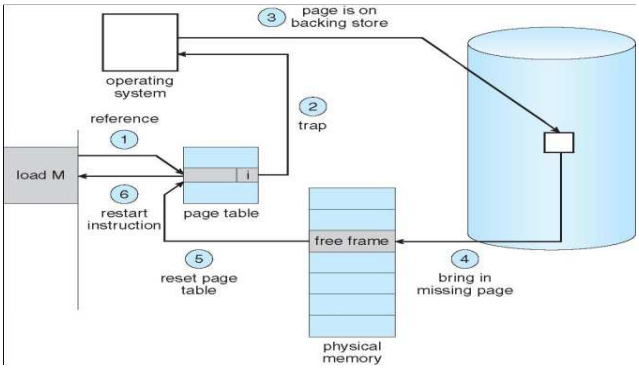
วิธีการนี้ต้องมีอุปกรณ์ทางฮาร์ดแวร์ คือ เพิ่มบิตสถานะ (Valid-invalid bit) อีกหนึ่งบิตต่อช่องในตารางเลขหน้า ถ้าสถานะมีค่าเป็นใช้ได้ (Valid) แสดงว่าหน้านั้นอยู่ในหน่วยความจำหลัก แต่ถ้าบิตสถานะมีค่าเป็นใช้ไม่ได้ (Invalid) แสดงว่าหน้านั้นอยู่ในจานบันทึก การใช้ตารางเลขหน้าจะทำงานเหมือนในระบบแบ่งหน้า คือ ตารางเลขหน้าจะต้องถูกนำลงในหน่วยความจำหลักพร้อมกับตัวโพรเซส เพียงแต่ไม่ต้องนำหน้าจริงเข้ามาด้วยเท่านั้น ดังแสดงในภาพที่ 8.5



ภาพที่ 8.5 แสดง Page table เมื่อมีบางเพจไม่อยู่ในหน่วยความจำ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.402)

การที่ระบบเรียกใช้เพจที่ใช้ไม่ได้เราเรียกว่า การผิดหน้า (Page fault) เมื่อเกิดการผิดหน้าขึ้น โพรเซสที่ถูกเรียกเพจหน้านั้นจะถูกยกเลิกไป ระบบก็จะนำข้อมูลเพจนั้นกลับเข้าสู่หน่วยความจำ จากนั้นก็ทำการปรับปรุงตารางจาก Invalid เป็น Valid สุดท้ายส่งสัญญาณไปให้ระบบ เพื่อเริ่มทำโพรเซสนั้นใหม่อีกครั้ง

เมื่อเกิดการผิดหน้า โพรเซสต้องการใช้ส่วนของการทำงานที่ยังไม่ได้ถูกย้ายเข้ามาในหน่วยความจำหลัก หน้าที่ของโพรเซสต้องการมีบิตสถานะเป็นใช้ไม่ได้ และจะต้องรายงานข้อผิดพลาดไปยังระบบปฏิบัติการ ดังแสดงในภาพที่ 8.6



ภาพที่ 8.6 แสดงขั้นตอนในการควบคุม Page fault
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.403)

สามารถอธิบายขั้นตอนในการจัดการการผิดหน้า (Page fault) ได้ดังต่อไปนี้

1) ตรวจดูตารางข้อมูลจำเพาะของโพรเซส (PCB) ว่าหน้าที่ต้องการอ้างอิงถูกต้องหรือไม่

2) ถ้าเป็นการอ้างอิงผิดพลาดก็ให้ยกเลิกโพรเซส ถ้าถูกต้องแต่หน้าที่ต้องการนี้ไม่ได้อยู่ในหน่วยความจำหลักให้ดำเนินการขั้นต่อไป

3) หาเนื้อที่ว่างในหน่วยความจำจริง (Frame) 1 หน้า เช่น เลือกจากรายชื่อหน้าว่าง

4) จัดคำร้องขอไปยังจานบันทึก เพื่อให้อ่านหน้าที่ต้องการมาไว้ในเนื้อที่ว่างที่เลือกไว้

5) เมื่อจานบันทึกนำหน้าเข้ามาเสร็จแล้ว ระบบก็จะแก้ไขบิตสถานะในตารางเลขหน้าเป็นใช้ได้

6) จัดให้โพรเซสทำงานต่อจากจุดที่เกิด “การผิดหน้า” โพรเซสจึงสามารถทำงานต่อไปได้เสมือนว่าหน้าที่ต้องการอยู่ในหน่วยความจำหลักตลอดเวลา

การที่มีหน่วยความจำเสมือนทำให้ระบบสามารถรันโปรแกรมต่าง ๆ ได้มากขึ้น ระบบมีพื้นที่หน่วยความจำรวมเพิ่มขึ้น ทำงานได้หลากหลายพร้อม ๆ กันได้มากขึ้น แต่การทำเช่นนี้มีข้อเสีย คือ เมื่อระบบสามารถรองรับเพจได้มากขึ้น ทำให้จำนวนข้อมูลในตารางเพจจะมีมากขึ้นตาม ถ้ามีข้อมูล 1 เพจต้องใช้ข้อมูล 1 แถวในตารางเพจเพื่อมาใช้อ้างอิง เพราะฉะนั้นในตารางจึงมีความเป็นไปได้ที่จะมีจำนวนแถวสูงสุดถึง 1 ล้านแถวในตาราง ซึ่งถ้าเป็นเช่นนั้นการทำ Linear search เพื่อหาข้อมูลจะเป็นเรื่องที่เป็นไปได้ยาก

8.2.2 ประสิทธิภาพของระบบจัดสรรหน้าตามคำร้องขอ (Performance of Demand Paging)

ในการจัดสรรหน้าตามคำร้องขอ (Demand paging) มีผลกระทบต่อประสิทธิภาพของระบบเป็นอย่างมาก ในระบบคอมพิวเตอร์ส่วนใหญ่เวลาที่ใช้เวลาเฉลี่ยในการอ้างอิงหน่วยความจำ (Memory access time) มีค่าระหว่าง 10 ถึง 200 นาโนวินาที ถ้าไม่มีการผิดหน้า เวลาเฉลี่ยในการอ้างอิงหน่วยความจำจะอยู่ในช่วงเวลาดังกล่าว แต่ถ้าเกิดการผิดหน้า เราต้องอ่านหน้าที่ต้องการจากจานบันทึกแล้วจึงอ้างอิงตำแหน่งที่ต้องการซ้ำใหม่อีกครั้ง

ให้ p เป็นโอกาสที่จะเกิดการผิดหน้า ($0 \le p \le 1$) ต้องการให้ค่า p เข้าใกล้ 0 มากที่สุด นั่นหมายถึง ทำให้เกิดการผิดหน้า (Page fault) น้อยครั้งที่สุด ดังนั้นสามารถหาเวลาเฉลี่ยในการอ้างอิงหน่วยความจำ (Effective access time) ดังนี้

$$\text{Effective Access Time (EAT)} = (1 - p) \times ma + p \times \text{page fault time.}$$

เมื่อเกิดการผิดหน้ามีขั้นตอนในการดำเนินการดังต่อไปนี้

1) รายงานข้อผิดพลาดไปยังระบบปฏิบัติการ

2) ใช้เก็บค่าต่าง ๆ ของรีจีสเตอร์ และสถานะของโพรเซส

3) ตรวจดูว่าสัญญาณขัดจังหวะนั้นเป็นการผิดหน้า

4) ตรวจดูว่าหน้าที่ต้องการถูกต้อง อยู่ในขอบเขตที่มีสิทธิใช้ และหาค่าตำแหน่งของหน้านี้จากการบันทึก

5) ทำการร้องขอไปยังจานบันทึกให้อ่านหน้านั้นเข้ามายังพื้นที่ว่าง (Free frame)

6) เข้าไปรอในแถวอุปกรณ์จนกระทั่งถึงคราวของเขา

7) อุปกรณ์และจานบันทึกจะหมุนและกวาดไปยังตำแหน่งที่ต้องการ

8) มีการโอนถ่ายข้อมูลจากอุปกรณ์ไปยังพื้นที่ว่าง

9) ขณะที่รอคอย ตัวจัดตารางการทำงานอาจจัดให้โพรเซสอื่นทำงานไปก่อน

10) สัญญาณขัดจังหวะอินเตอร์รัพจากจานบันทึกว่าอ่านเสร็จแล้ว

11) จัดเก็บค่าของรีจีสเตอร์และสถานะของโพรเซสอื่นที่กำลังทำงานอยู่

12) ตรวจดูสัญญาณจากจานบันทึก

13) แก้ไขค่าในตารางเลขหน้าและตารางอื่น ๆ ที่เกี่ยวข้อง เพื่อแสดงว่าหน้าที่ที่ต้องการอยู่ในหน่วยความจำหลักแล้ว

14) เข้าไปรอคอยอยู่ในแถวพร้อม เพื่อรอเข้าใช้หน่วยประมวลผล

15) เมื่อถึงคราวบรรจุค่าต่าง ๆ ที่เก็บไว้ลงในรีจิสเตอร์และตารางเลขหน้าของโพรเซสแล้ว เริ่มทำคำสั่งเดิมที่หยุดไว้

ขั้นตอนในการดำเนินการอาจรวมขั้นตอนต่าง ๆ เป็น 3 ขั้นตอนใหญ่ ๆ ในการจัดการ การผิดหน้าได้แก่ 1) จัดการสัญญาณขัดจังหวะ (Interrupt) จากการผิดหน้า 2) อ่านหน้าที่ต้องการเข้ามา 3) ให้โพรเซสทำงานต่อ

<u>ตัวอย่าง</u> Demand Paging

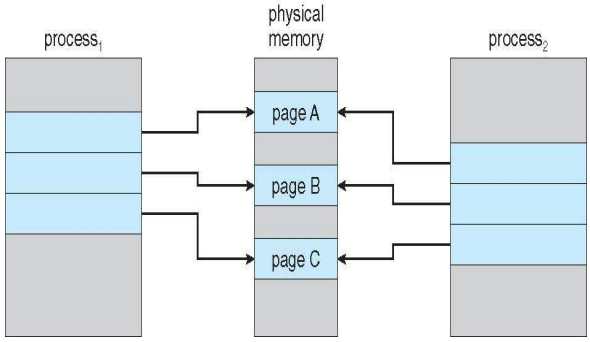
เวลาเฉลี่ยในการเข้าถึงหน่วยความจำ (Memory access time) เท่ากับ 200 นาโนวินาที และเวลาเฉลี่ยของการจัดการการผิดหน้า (Page fault time) เป็น 8 มิลลิวินาที จงหาเวลาเฉลี่ยในการอ้างอิงหน่วยความจำ (Effective access time) ให้มีหน่วยเป็น นาโนวินาที

$$
\begin{aligned}
\text{Effective access time} &= (1 - p) \times (200) + p \text{ (8 มิลลิวินาที)} \\
&= (1 - p) \times 200 + p \times 8{,}000{,}000 \\
&= 200 + 7{,}999{,}800 \times p. \text{ นาโนวินาที}
\end{aligned}
$$

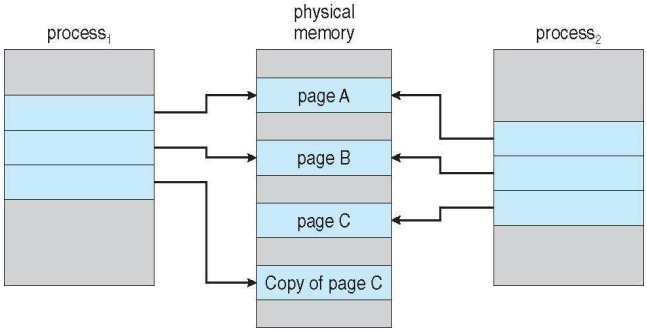
8.3 เทคนิคการบันทึกข้อมูลด้วยการคัดลอกข้อมูล

ในหัวข้อนี้ เราจะอธิบายโพรเซสที่เริ่มต้นทำงานอย่างรวดเร็วได้อย่างไร โดยในเพจประกอบด้วยคำสั่งแรก อย่างไรก็ตามโพรเซสใช้ฟังก์ชั่น fork() (เรียกผ่าน System call) ในการเริ่มต้นแบบทางอ้อม ระบบต้องการวิธีการ Demand Paging โดยใช้วิธีที่เหมือนกับ Page Sharing เทคนิคนี้จัดการสำหรับการสร้างโพรเซสแบบรวดเร็วและเล็ก จำนวนเพจใหม่ซึ่งจำเป็นต้องจัดสรรให้กับโพรเซสที่เพิ่งถูกสร้างขึ้นใหม่ การเรียกฟังก์ชั่น fork() กลับมาสร้างโพรเซสลูก (A child process) เหมือนกับของโพรเซสแม่ วิธี fork() ทำงานโดยคัดลอกตำแหน่งพื้นที่ว่างของโพรเซสแม่ให้กับโพรเซสลูก เหมือนกับเพจเป็นส่วนหนึ่งของโพรเซสแม่

อย่างไรก็ตามเมื่อพิจารณาโพรเซสลูกที่เกิดจากฟังก์ชั่น exec() (เรียกผ่าน System call) ทันทีหลังการสร้าง การคัดลอกตำแหน่งพื้นที่ว่างของโพรเซสแม่ อาจจะไม่จำเป็นหรืออาจใช้วิธีการอื่นที่รู้จักในเทคนิค Copy-on-Write ซึ่งทำงานโดยจัดสรรโพรเซสแม่และโพรเซสลูกในขั้นต้นเพจที่เหมือนกันให้ใช้ร่วมกัน (Share page) การ Share page เป็นการทำ Copy-on-Write หมายความว่า ถ้าโพรเซสหนึ่งเขียน Share Page การคัดลอก Share page เป็นการสร้าง Copy-on-Write แสดงตัวอย่างในภาพที่ 8.7 และ ภาพที่ 8.8 ซึ่งแสดงเพจของหน่วยความจำทางกายภาพก่อนและหลัง Process1 จะเปลี่ยนแปลง Page C



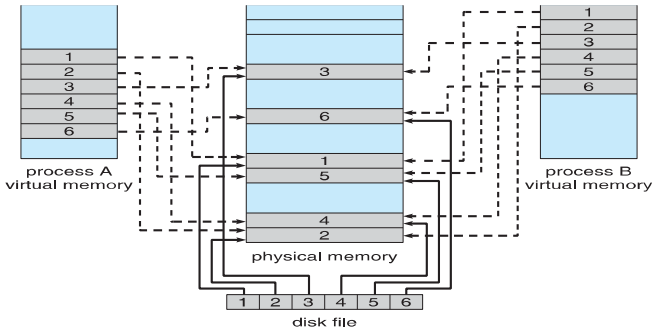
ภาพที่ 8.7 แสดงก่อน โพรเซส 1 จะเปลี่ยนแปลง Page C
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.408)



ภาพที่ 8.8 แสดงหลัง โพรเซส 1 จะเปลี่ยนแปลง Page C
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.409)

8.4 การเชื่อมโยงแฟ้มข้อมูลกับหน่วยความจำ

การเชื่อมโยงแฟ้มข้อมูลกับหน่วยความจำ ยินยอมให้แฟ้มข้อมูลทั้งอินพุตและเอาท์พุตถูกใช้งานเหมือนรูทีนการเข้าถึงหน่วยความจำได้โดยการเชื่อมโยงดิกส์บล็อก (Mapping a disk block) ไปเป็นเพจในหน่วยความจำ แฟ้มข้อมูลจะถูกกำหนดค่าเริ่มต้นของการอ่านโดยใช้ Demand paging สัดส่วนของ Page sized ของไฟล์ที่อ่านจากระบบไฟล์ไปยัง Physical page ลำดับย่อยของการอ่านเขียนแฟ้มข้อมูลจะดำเนินการเช่นเดียวกับการเข้าถึงหน่วยความจำ การเข้าถึงแฟ้มข้อมูลอย่างง่ายโดยการดูแลแฟ้มจากอินพุตเอาท์พุต (Treating file I/O) ผ่านหน่วยความจำมากกว่าการใช้ฟังก์ชั่น read() write() (เรียกผ่าน System call) และอนุญาตให้หลายโพรเซสทำการ Map แฟ้มข้อมูลเดียวกันให้ใช้เพจร่วมกันในหน่วยความจำได้อีกด้วย



ภาพที่ 8.9 การเตรียมข้อมูลของหน่วยความจำ mapping กับไฟล์ใน disk
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.432)

8.5 อัลกอริทึ่มการสับเปลี่ยนหน้า

นโยบายการสับเปลี่ยนหน้าเป็นการเลือกหน้าเก่าที่อยู่ในหน่วยความจำออกเพื่อจะแทนที่ด้วยหน้าใหม่ที่จะนำเข้ามา ซึ่งจะมีสิ่งที่ต้องพิจารณาหลายกรณีด้วยกัน เช่น

1) มีจำนวนเฟรมเท่าไหร่ที่สามารถให้กับโพรเซสแต่ละตัวได้

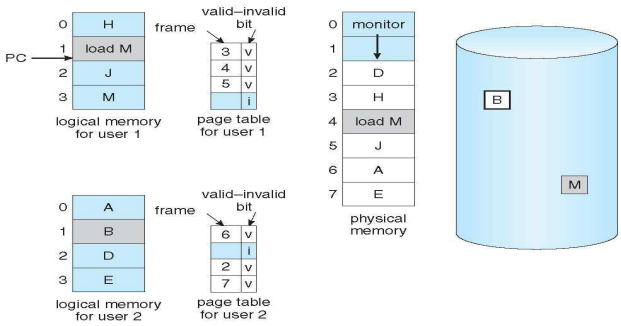
2) กลุ่มของหน้าที่ถูกพิจารณาให้สับเปลี่ยนหน้านั้นควรจะถูกจำกัดหรือไม่ เพื่อป้องกันไม่ให้โพรเซสที่ทำให้เกิดการผิดหน้าทำงาน

3) ในกลุ่มของหน้าที่เราพิจารณา เราจะเลือกหน้าใดที่จะนำไปสับเปลี่ยน

4) ถ้าเราให้หน่วยความจำแก่โพรเซสใด ๆ น้อยเท่าไหร่ ก็จะทำให้มีจำนวนโพรเซสเข้ามาใช้งานหน่วยความจำมากขึ้นเท่านั้น ซึ่งจะทำให้ความน่าจะเป็นในการที่จะค้นพบโพรเซสอย่างน้อยหนึ่งโพรเซสที่พร้อมจะทำงานมากขึ้น ทำให้เราเสียเวลาในการสลับหน้าน้อยลง

5) ถ้ามีจำนวนหน้าของโพรเซสหนึ่ง ๆ ที่สามารถอยู่ในหน่วยความจำหลักน้อย (จำนวนเฟรมน้อย) ก็จะทำให้เกิดอัตราการเกิดการผิดหน้าสูงมากขึ้น

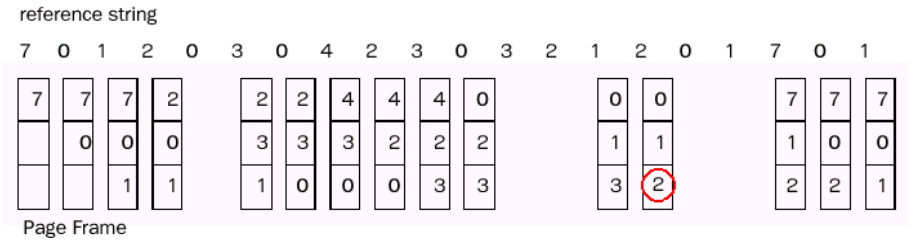
6) นอกเหนือจากการกำหนดขนาดของหน้าคงที่แล้ว การให้หน่วยความจำเพิ่มกับโพรเซสใดโพรเซสหนึ่งจะไม่มีผลกระทบต่ออัตราการผิดหน้ามากนัก



ภาพที่ 8.10 แสดงความจำเป็นที่จะต้องมีการสับเปลี่ยนหน้า
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.410)

8.5.1 วิธีสับเปลี่ยนแบบมาก่อน-ออกก่อน (FIFO: First-In-First-Out Algorithms)

เป็นวิธีที่ง่ายที่สุด โดยจะใช้เวลาที่หน้านั้น ๆ ถูกนำเข้ามาในหน่วยความจำหลักเป็นเกณฑ์ในการตัดสินใจ เมื่อต้องการเลือกหน้าบางหน้าออก ก็ให้เลือกหน้าที่เข้ามานานที่สุด ในทางปฏิบัติเราอาจจะไม่ต้องจดเวลาจริง ๆ ที่หน้านั้นเข้ามาใช้งานก็ได้ เพียงแต่สร้างคิวแบบมาก่อน-ออกก่อน (FIFO queue) สำหรับเก็บหมายเลขหน้าที่อยู่ในหน่วยความจำ เมื่อมีการนำหน้าใหม่เข้ามาให้เอาหมายเลขมาไว้ที่ปลายแถว แล้วเลือกหน้าออกจากหัวแถว

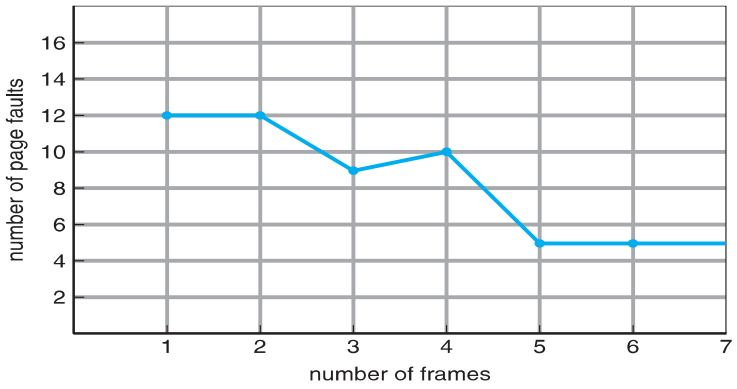


ภาพที่ 8.11 วิธีสับเปลี่ยนแบบมาก่อน-ออกก่อน (First-In-First-Out Algorithms : FIFO)

จากตัวอย่างในภาพที่ 8.11 นั้นสมมุติว่าเริ่มต้นในระบบมีแค่ 3 เฟรมและว่างอยู่ จากแถวของหน้าที่เข้ามาในเฟรมคือ (7,0,1) ซึ่งจะเกิดการผิดหน้าเนื่องจากหน้าทั้งสามยังไม่ได้อยู่ในเฟรม จึงมีการนำหน้าที่ต้องการเข้ามาในหน่วยความจำ การเรียกหน้าต่อไปคือหน้า 2 จะถูกสับเปลี่ยนเข้ามาแทน หน้า 7 เพราะหน้า 7 เข้ามาก่อนเป็นหน้าแรก ครั้งต่อไปเป็นหน้า 0 แต่หน้า 0 อยู่ในหน่วยความจำอยู่แล้ว จึงไม่เกิดการผิดหน้าและก็ไม่เกิดการสับเปลี่ยนหน้าด้วย ต่อไปหน้า 3 จะถูกเปลี่ยนเข้ามาแทนหน้า 0 และต่อไปหน้า 0 จะถูกสับเปลี่ยนเข้ามาแทนหน้า 1 ต่อไปเรื่อย ๆ

จากรูปนั้นจะแสดงให้เห็นว่าทุก ๆ ครั้งที่เกิดการผิดหน้า หน้าอะไรที่จะถูกสับเปลี่ยนเข้ามาในเฟรม ซึ่งการผิดหน้าในตัวอย่างนี้มีทั้งหมด 15 ครั้ง วิธีคิดแบบมาก่อน-ออกก่อนนี้ เป็นวิธีที่เข้าใจและสามารถเขียนเป็นโปรแกรมได้ง่าย แต่อาจจะไม่มีประสิทธิภาพมากนัก เพราะหน้าที่ถูกสับเปลี่ยนไป อาจเป็นส่วนของโปรแกรมเริ่มต้นซึ่งถูกใช้ในตอนต้น ๆ และไม่มีความต้องการอีกต่อไป หรือในทางตรงกันข้ามหน้าที่ถูกสับเปลี่ยนออกอาจเก็บค่าตัวแปรหลักของโปรแกรมซึ่งใช้บ่อยมาก โดยถูกกำหนดเป็นค่าเริ่มต้นในตอนแรก ๆ ของโปรแกรม

พึงสังเกตว่า ถึงเราจะสับเปลี่ยนหน้าที่กำลังใช้งานออกไป แต่การทำงานของโพรเซสก็ยังคงถูกต้องเหมือนเดิม เพราะหลังจากที่เรานำหน้าที่กำลังใช้งานนี้ออกไปเพื่อใส่หน้าใหม่ การผิดหน้าก็เกิดขึ้นเกือบจะทันทีเมื่อทำหน้าที่กำลังใช้งานนั้นกลับมา และหน้าบางหน้าในหน่วยความจำก็จะถูกสับเปลี่ยนออกไปแทน ดังนั้นการสับเปลี่ยนที่ไม่ดีจะเพิ่มอัตราการผิดหน้าได้ และทำให้การทำงานของโพรเซสช้าลง แต่ไม่ทำให้ระบบทำงานผิดพลาด ตัวอย่างต่อไปนี้จะแสดงปัญหาที่สามารถเกิดขึ้นกับวิธีคิดแบบมาก่อนออกก่อน ถ้าเรามีหน้าของโพรเซสที่จะเข้ามาใช้งานดังนี้



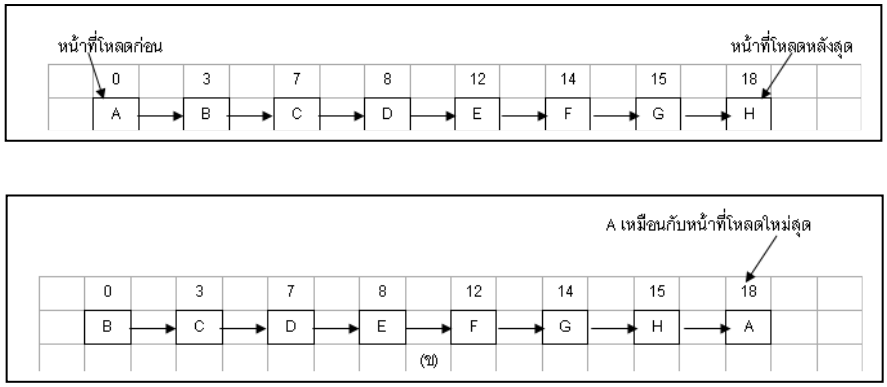
ภาพที่ 8.12 กราฟแสดงปรากฏการณ์เบลาดี้ ในวิธีสับเปลี่ยนแบบมาก่อน-ออกก่อน

เมื่อทดลองหาจำนวนครั้งของการผิดหน้าเทียบกับจำนวนเฟรมที่มี จะได้กราฟดังในภาพที่ 8.12 จะสังเกตได้ว่า เมื่อหน่วยความจำมีเฟรมทั้งหมด 4 เฟรม จะเกิดการผิดหน้าทั้งหมด 10 ครั้ง ซึ่งมากกว่า เมื่อมีทั้งหมด 3 เฟรม ผลลัพธ์ที่ไม่ปกตินี้ เราเรียกว่า ปรากฏการณ์เบลาดี้ (Belady's anomaly) ซึ่งแสดงให้เห็นว่า วิธีการสับเปลี่ยนหน้าบางแบบก็อาจจะทำให้อัตราการผิดหน้าเพิ่มขึ้นได้ ถึงเราจะเพิ่มจำนวนเฟรมขึ้นก็ตาม แต่เดิมเราคิดว่าการเพิ่มหน่วยความจำย่อมจะทำให้โพรเซสเพิ่มประสิทธิภาพและทำงานได้ดีขึ้น แต่ในการค้นคว้าล่าสุดนั้นพบว่าสมมุติฐานดังกล่าวนั้นไม่เป็นความจริงเสมอไป

8.5.2 วิธีสับเปลี่ยนแบบให้โอกาสครั้งที่สอง (Second Chance Page Replacement Algorithms)

วิธีนี้คิดขึ้นมาเพื่อแก้ปัญหาที่เกิดจากแบบมาก่อน-ออกก่อน โดยการป้องกันการเปลี่ยนหน้าที่ถูกเรียกใช้งานบ่อยออกไป ซึ่งสามารถทำได้โดยการเช็คที่บิต Referenced (R) ของหน้าที่เข้ามานานที่สุด ถ้าบิต R มีค่าเป็น 0 ก็แสดงว่าหน้านั้นเก่าและไม่ได้ถูกเรียกใช้งานเลย ระบบก็สามารถทำการสับเปลี่ยนได้ทันที แต่ถ้าบิต R มีค่าเท่ากับ 1 ก็ให้กำหนดให้บิต R นั้นมีค่าเป็น 0 และนำหน้านั้นกลับไปเข้าแถวใหม่อีกครั้ง พร้อมกับทำการเปลี่ยนแปลงเวลาของหน้านั้นใหม่เหมือนกับว่าหน้านั้นเพิ่งเข้ามาในหน่วยความจำ จากนั้นก็ทำการค้นหาหน้าที่จะถูกสับเปลี่ยนต่อไป

พิจารณาภาพที่ 8.13 จะเห็นว่า หน้า A ถึง H นั้นจะถูกเก็บไว้ในลิงค์ลิสต์ และถูกจัดเรียงโดยเวลาที่หน้าเหล่านี้เข้ามาในหน่วยความจำ สมมติว่าการผิดหน้าเกิดขึ้นเมื่อเวลาเท่ากับ 20 หน้าที่อยู่มานานที่สุดคือ หน้า A ซึ่งเข้ามาตั้งแต่เวลาเท่ากับ 0 (เมื่อโพรเซสเริ่มทำงาน) ถ้าบิต R ในหน้า A มีค่าเป็น 0 หน้า A จะถูกสับเปลี่ยนออกไป (โดยอาจจะมีการเขียนลงดิสก์ ถ้ามีการเปลี่ยนแปลงหรือทิ้งไปเฉย ๆ ถ้าไม่มีการเปลี่ยนแปลง ซึ่งสังเกตได้จากบิต Modified (M)) แต่ถ้าบิต R นั้นถูกกำหนดเป็น 1 หน้า A ก็จะถูกนำไปใส่ตอนท้ายของแถว และเวลาที่เข้ามาในลิสต์ก็ถูกเปลี่ยนเป็นเวลาปัจจุบัน คือ 20 พร้อมกันนั้น บิต R ก็ถูกลบเป็น 0 หน้าที่จะถูกเช็คเพื่อสับเปลี่ยนออกต่อไปก็คือ หน้า B

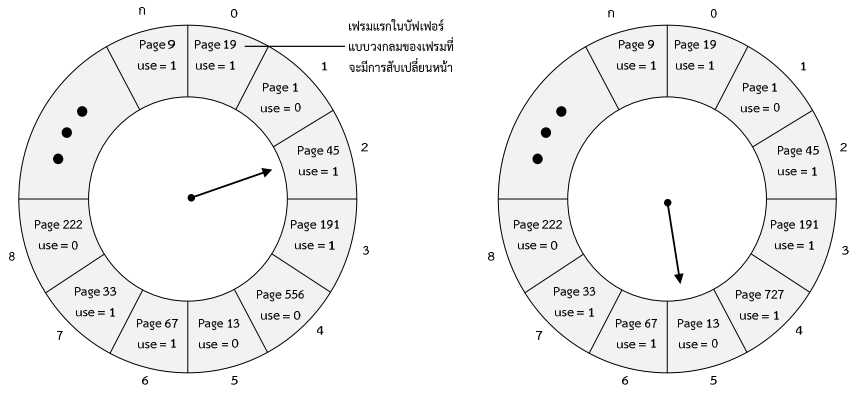


ภาพที่ 8.13 วิธีการสับเปลี่ยนหน้าแบบให้โอกาสครั้งที่สอง

หลักการคือ ค้นหาหน้าที่เก่าที่สุดและไม่ได้ถูกอ้างอิงหรือเรียกใช้งาน แต่ถ้าทุกหน้าในระบบถูกเรียกใช้งานหมด วิธีการนี้ก็จะกลายเป็นมาก่อน-ออกก่อนนั่นเอง สมมติว่าในตัวอย่างภาพที่ 8.13 บิต R ของทุกหน้าถูกกำหนดเป็น 1 ระบบปฏิบัติการจะทำการย้ายหน้าแต่ละหน้ากลับไปเข้าแถวใหม่ พร้อมกับทำการลบบิต R เป็น 0 จนท้ายที่สุดหน้า A ก็จะกลับมาที่หน้าแถวอีกครั้งหนึ่ง และถ้าบิต R มีค่าเป็น 0 หน้า A ก็จะถูกสับเปลี่ยนออกไป

8.5.3 วิธีสับเปลี่ยนแบบวงรอบนาฬิกา (Clock Page Replacement Algorithms)

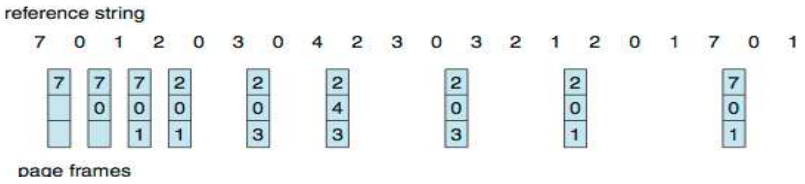
ใช้วิธีเรียงเฟรมทุกเฟรมเป็นรูปวงกลมเหมือนนาฬิกา ภาพที่ 8.14 และมีเข็มนาฬิกาชี้ไปที่หน้าที่เก่าที่สุด เมื่อเกิดการผิดหน้า หน้าที่มีเข็มนาฬิกาชี้อยู่จะถูกตรวจสอบ ถ้าบิต R มีค่าเป็น 0 หน้านั้นจะถูกสับเปลี่ยนออกไป และหน้าใหม่ก็จะถูกใส่เข้ามาในตำแหน่งเดิม พร้อมกันนั้นเข็มนาฬิกาจะทำการเลื่อนไปด้านหน้า 1 ตำแหน่ง แต่ถ้าบิต R ถูกกำหนดเป็น 1 ก็ให้ลบค่าของบิตนั้นเป็น 0 และเลื่อนเข็มไปหน้าถัดไป วิธีการดังกล่าวจะถูกทำซ้ำจนกว่าเราจะได้หน้าที่มีบิต R เป็น 0 ซึ่งวิธีการนี้จะเหมือนวิธีการแบบให้โอกาสครั้งที่ 2 เพียงแต่แตกต่างกันในวิธีการใช้งานเท่านั้น



ภาพที่ 8.14 วิธีสับเปลี่ยนแบบวงรอบนาฬิกา

8.5.4 วิธีสับเปลี่ยนแบบที่ดีที่สุด (Optimal Page Replacement Algorithms : OPT หรือ MIN)

ใช้หลักการ “ให้เลือกสับเปลี่ยนหน้าที่จะไม่ถูกเรียกใช้งาน และมีระยะเวลารอการเรียกใช้ที่นานที่สุด” วิธีนี้จะทำให้เกิดอัตราการผิดหน้าต่ำที่สุดสำหรับพื้นที่ ๆ มีจำนวนเฟรมหนึ่ง จากตัวอย่างในภาพที่ 8.15 จะเห็นว่าวิธีแบบ OPT หรือ MIN นี้จะดีที่สุด ทำให้เกิดการผิดหน้าทั้งหมด 9 ครั้ง ดังนี้



ภาพที่ 8.15 วิธีสับเปลี่ยนแบบที่ดีที่สุด

1) การเรียก 3 หน้าแรกเข้ามาใช้งานจะทำให้เกิดการผิดหน้า 3 ครั้ง

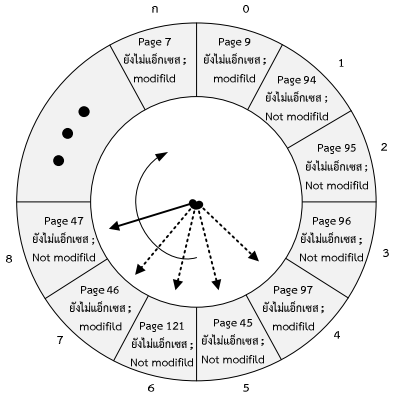
2) การเรียกหน้า 2 จะสลับหน้า 7 ออก เพราะหน้า 7 นั้นจะใช้งานอีกครั้งในลำดับที่ 18, หน้า 0 จะใช้งานอีกเป็นลำดับที่ 5 และหน้า 1 จะใช้ในลำดับที่ 14

3) การเรียกหน้า 3 ก็จะสลับหน้า 1 ออก เพราะหน้า 1 เป็นหน้าสุดท้ายที่จะถูกเรียกใช้งาน (ลำดับที่ 8 จากหน้าที่ 3) ในขณะที่หน้า 0 จะถูกเรียกเป็นลำดับที่ 1 และหน้า 2 เป็นลำดับที่ 3 เมื่อเทียบจากตำแหน่งที่หน้า 3 กำลังเข้าใช้งาน

วิธีนี้จะทำให้เกิดการผิดหน้า 9 ครั้ง ซึ่งดีกว่าวิธีแบบมาก่อน-ออกก่อนมาก (มีการผิดหน้า 15 ครั้ง) ถ้าไม่คิดการผิดหน้า 3 ครั้งแรก ซึ่งจะต้องเกิดขึ้นอย่างแน่นอน จะเห็นได้ว่าแบบที่ดีที่สุดนี้ดีกว่าแบบมาก่อน-ออกก่อน ได้ถึง 2 เท่า วิธีแบบที่ดีที่สุดนี้ยากที่จะสร้างได้จริง ๆ เพราะเราต้องรู้ในอนาคตว่า จะมีการเรียกหน้าใดเมื่อไหร่ วิธีแบบที่ดีที่สุดนี้จึงมีไว้เพื่อใช้ในการเปรียบเทียบเท่านั้น

8.5.5 วิธีสับเปลี่ยนแบบที่ไม่ได้ใช้งานออกก่อน (Not Recently Used : NRU)

วิธีนี้จะพัฒนาจากวิธีการสับเปลี่ยนแบบวงรอบนาฬิกา โดยพิจารณาบิตที่สำคัญในตารางหน้าเพิ่มขึ้นอีก 1 บิต เนื่องจากระบบปฏิบัติการสามารถทำการเก็บข้อมูลสถิติได้ว่า หน้าใดที่กำลังถูกใช้หรือไม่ได้ใช้งาน โดยได้ข้อมูลจากบิตในแถวของตารางหน้าที่แสดงสถานะการทำงานของแต่ละหน้า บิต Referenced จะถูกกำหนดเป็น 1 เมื่อหน้านั้นถูกเรียกใช้งาน และบิต Modified จะถูกกำหนดเป็น 1 เมื่อหน้านั้นมีการเปลี่ยนแปลง บิตทั้งสองนั้นเป็นฟิลด์ที่อยู่ในแถวหน้าของตารางหน้า และจะถูกเปลี่ยนแปลงค่าทุกครั้งเมื่อมีการเรียกใช้งานหรือถูกอ้างอิงถึง ดังนั้นการกำหนดค่าต่าง ๆ ควรเป็นหน้าที่ของฮาร์ดแวร์ เมื่อบิตเหล่านั้นถูกกำหนดเป็น 1 เมื่อใด บิตนั้นก็จะมีค่าเท่ากับ 1 จนกว่าระบบปฏิบัติการจะกำหนดกลับเป็น 0 โดยใช้ซอฟต์แวร์



ภาพที่ 8.16 การสับเปลี่ยนหน้าแบบ NRU

บิต R และ M ทั้งสองนี้สามารถนำมาใช้ในการสร้างวิธีการสับเปลี่ยนหน้าแบบใหม่ได้โดย เมื่อโพรเซสหนึ่ง ๆ เริ่มทำงาน บิตทั้งสองในทุก ๆ หน้าจะถูกกำหนดเป็น 0 โดยระบบปฏิบัติการ และเมื่อครบวงรอบของระยะเวลาหนึ่ง ๆ เช่น จากการแทรกของอินเทอร์รัพ บิต R จะถูกทำความสะอาด หรือถูกกำหนดค่าเป็น 0 เพื่อทำหน้าที่เป็นตัวแยกหน้าที่ไม่ได้ถูกเรียกใช้งานออกจากหน้าอื่น ๆ เมื่อมีการผิดหน้าเกิดขึ้น ระบบปฏิบัติการจะทำการตรวจสอบทุกหน้าและแบ่งหน้าเหล่านั้นออกเป็น 4 กลุ่ม ขึ้นอยู่กับค่าของบิต R และบิต M ดังนี้

กลุ่มที่ 0 : ไม่ถูกเรียกใช้งาน และ ไม่มีการเปลี่ยนแปลงค่า
กลุ่มที่ 1 : ไม่ถูกเรียกใช้งาน แต่ มีการเปลี่ยนแปลงค่า
กลุ่มที่ 2 : ถูกเรียกใช้งาน แต่ ไม่มีการเปลี่ยนแปลงค่า
กลุ่มที่ 3 : ถูกเรียกใช้งาน และ มีการเปลี่ยนแปลงค่า

แม้ว่าอาจจะไม่มีหน้าใดอยู่ในกลุ่มที่ 1 เลย แต่มันก็อาจจะเกิดขึ้นได้ในกรณีที่หน้าในกลุ่มที่ 3 นั้นถูกระบบปฏิบัติการทำการกำหนดบิต R เป็น 0 เมื่อครบวงรอบของระยะเวลาหนึ่ง ๆ ซึ่งระบบปฏิบัติการไม่ได้กำหนดบิต M เป็น 0 ด้วย เพราะเราต้องการข้อมูลที่ว่าหน้านั้นจะต้องถูกเขียนทับกลับลงไปในดิสก์ด้วยหรือไม่ จึงทำการเคลียร์บิต R แต่ไม่แตะต้องบิต M และทำให้หน้านั้นอยู่ในกลุ่มที่ 1

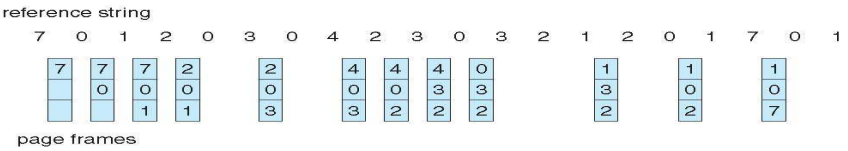
วิธี NRU นี้จะสุ่มเอาหน้าออกจากกลุ่มลำดับต่ำสุด (เช่น กลุ่มที่ 1 มีลำดับต่ำกว่ากลุ่มที่ 2 และ 3 ตามลำดับ) ที่มีหน้าของโพรเซสอยู่ นอกจากนั้นวิธีการนี้ยังดูเหมือนว่าจะพยายามที่จะนำเอาหน้าที่ถูกเปลี่ยนแปลงแต่ไม่ได้ถูกเรียกใช้งานหรืออ้างอิงถึงออก ทุกครั้งที่มีการครบวงรอบนาฬิกา (ประมาณ 20 มิลลิวินาที) มากกว่าหน้าที่ไม่มีการเปลี่ยนแปลงแต่ถูกเรียกใช้งานบ่อย ข้อเด่นของวิธี NRU คือ ง่ายที่จะทำความเข้าใจ ไม่ยากเกินไปที่จะนำมาใช้งานจริง และมีประสิทธิภาพค่อนข้างดี (แม้ว่าจะไม่ดีเท่ากับแบบที่ดีที่สุด) และเป็นวิธีที่ใช้งานในระบบปฏิบัติการของเครื่อง Macintosh

8.5.6 การสับเปลี่ยนแบบใช้งานน้อยที่สุด-ออกก่อน (Least Recently Used : LRU)

วิธีนี้จะเป็นการใช้ข้อมูลในอดีตที่ผ่านมาประมาณการอนาคตอันใกล้ โดยอาจจะสับเปลี่ยนหน้าที่ไม่ได้ถูกเรียกใช้งานเป็นเวลานานที่สุดออก ซึ่งเรียกว่าเป็นแบบใช้งานน้อยที่สุดออกก่อน (LRU) วิธี

แบบนี้จะบันทึกเวลาที่แต่ละหน้าถูกอ้างอิงครั้งล่าสุดไว้ เมื่อต้องการเลือกหน้าเพื่อสับเปลี่ยนออก ก็จะเลือกหน้าที่ไม่ได้ใช้มาเป็นเวลานานสุด (หน้าที่มีตัวเลขเวลาน้อยที่สุด)

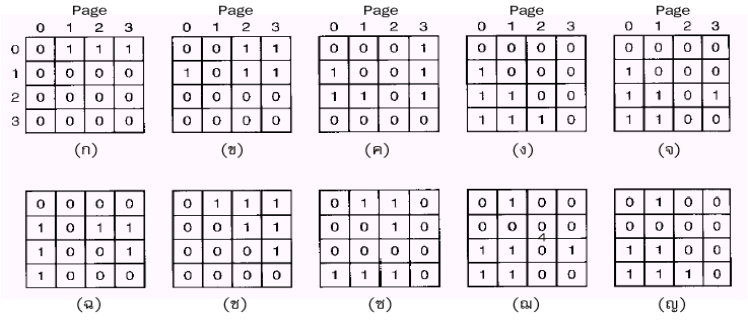
จะเห็นได้ว่าวิธีการนี้ได้มองย้อนไปในเวลาอดีต แทนที่จะมองไปในอนาคต แม้ว่า LRU น่าจะใช้งานได้ดีที่สุดในทางทฤษฎี แต่ก็ไม่ง่ายในทางปฏิบัติ เนื่องจากการใช้ LRU นั้น ระบบจะต้องสร้างลิงค์ลิสต์ของทุก ๆ หน้าในหน่วยความจำที่มีหน้าที่ใช้งานน้อยที่สุดอยู่ที่หัวแถว และหน้าที่ใช้งานบ่อยที่สุดอยู่ท้ายสุด และความยากของวิธีนี้ก็คือลิสต์จะต้องทำการปรับเปลี่ยนทุกครั้งที่มีการอ้างอิงถึง หรือมีการเรียกใช้งานในหน่วยความจำ ซึ่งการค้นหาหน้าในลิสต์ การลบ หรือการย้ายหน้าไปมานั้น ล้วนแต่เป็นการทำงานที่ต้องใช้เวลาของโพรเซสเซอร์ทั้งสิ้น



ภาพที่ 8.17 การสับเปลี่ยนหน้าแบบ NRU การสับเปลี่ยนหน้าแบบ LRU

แต่อย่างไรก็ตามถ้าเราใช้ LRU ด้วยฮาร์ดแวร์พิเศษก็อาจจะเป็นทางเลือกอีกทางหนึ่ง โดยเราอาจจะพิจารณาจากตัวอย่างที่ง่ายที่สุดก่อน ซึ่งวิธีนี้จะใช้ฮาร์ดแวร์ที่มีตัวนับ หรือ counter (C) เป็นแบบ 64 บิต ที่จะทำการเพิ่มค่าทุกครั้งที่มีการเรียกใช้งาน นอกจากนี้ในตารางหน้าของแต่ละหน้าจะต้องมีฟิลด์ขนาดใหญ่ที่สามารถใส่ค่าของตัวนับนี้ได้ หลังจากที่มีการอ้างอิงหน่วยความจำในแต่ละครั้ง ค่า C จะถูกบันทึกลงในตารางหน้าของหน้านั้น ๆ เมื่อมีการผิดหน้าระบบปฏิบัติการจะต้องทำการตรวจสอบทุกค่าของ C ในตารางหน้า เพื่อค้นหาหน้าที่มีค่า C น้อยที่สุด ซึ่งหน้านั้นก็คือหน้าที่ใช้งานน้อยที่สุดนั่นเอง

พิจารณาการใช้ LRU ด้วยฮาร์ดแวร์ โดยสมมติให้ในระบบมี n เฟรม ซึ่งทำให้ฮาร์ดแวร์นั้นจะต้องดูแลเมตริกซ์ nxn บิต ซึ่งมีค่าเริ่มต้นเป็น 0 หมด เมื่อใดก็ตามที่เฟรม k ถูกอ้างอิงถึง ฮาร์ดแวร์นั้นก็จะกำหนดให้ทุกบิตในแถว k เป็น 1 และกำหนดให้ทุกบิตในคอลัมน์ k เป็น 0 เมื่อมีการตรวจสอบ แถวที่มีค่าของบิตรวมกันน้อยที่สุด คือ แถวที่ถูกเรียกใช้งานน้อยที่สุด ภาพที่ 8.18 แสดงการทำงานของอัลกอริทึมนี้ โดยให้ระบบมี 4 เฟรม และมีการอ้างอิงถึงหน้าต่าง ๆ ตามลำดับ คือ 0 1 2 3 2 1 0 3 2 3



ภาพที่ 8.18 การสับเปลี่ยนหน้าแบบ LRU หน้าที่อยู่ในเฟรมที่ 1 ก็จะถูกเลือกออกเพราะไม่ได้ถูกเรียกใช้งานนานที่สุด

หลังจากที่หน้า 0 ถูกอ้างอิงถึง เราจะได้ดังภาพที่ 8.18 (ก) และหลังจากที่หน้า 1 ถูกอ้างอิงถึง เราก็จะได้เมตริกซ์ดังภาพ 8.18 (ข) ตามลำดับ หลังจากที่ทุกหน้าได้ทำงานเสร็จ เราจะได้เมตริกซ์ดังภาพที่ 8.18 (ญ) ซึ่งมีค่าของแถวที่ 1 รวมกันน้อยที่สุด ดังนั้นเมื่อเกิดการผิดหน้าขึ้น หน้าที่อยู่ในเฟรมที่ 1 ก็จะถูกเลือกออก เพราะไม่ได้ถูกเรียกใช้งานนานที่สุด แต่ถ้าเมตริกซ์สุดท้ายได้ดังภาพที่ 8.19 ก็ให้เราหาผลรวมของเฟรม 0 และเฟรมที่ 3 ดังนี้

เฟรมที่	0	1	2	3
0	0	1	0	0
1	0	0	1	1
2	1	1	0	0
3	0	0	1	0

ภาพที่ 8.19 ในกรณีที่ในตารางมีค่าของแถวเท่ากัน

ผลรวมของเฟรม 0 และ เฟรมที่ 3 มีค่าดังนี้

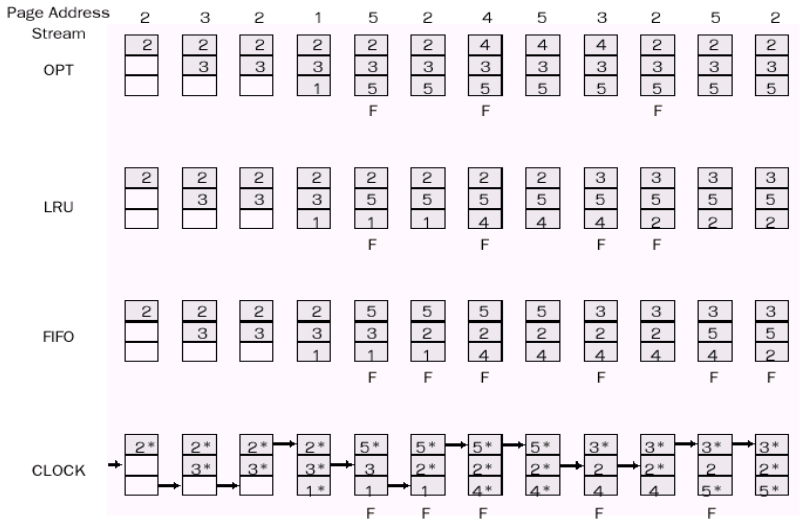
Frame 0 = (0 1 0 0) = 4 (ฐานสิบ)

Frame 3 = (0 0 1 0) = 2 (ฐานสิบ)

ก็ให้เลือกหน้าในเฟรมที่ 3 ออก เพราะมีค่าน้อยกว่าเฟรมที่ 0

8.5.7 เปรียบเทียบวิธีการสับเปลี่ยนหน้าแบบต่าง ๆ

ถ้าสมมติว่าในระบบมีทั้งหมด 3 เฟรม ภาพที่ 8.20 จะแสดงการเปรียบเทียบวิธีการสับเปลี่ยนหน้าแบบ OPT, LRU, FIFO และ CLOCK ดังนี้



ภาพที่ 8.20 เปรียบเทียบการสับเปลี่ยนหน้าแบบต่าง ๆ

8.6 สรุป

หนึ่งในงานที่สำคัญและซับซ้อนของการออกแบบระบบปฏิบัติงานก็คือ การจัดการกับหน่วยความจำ การจัดการกับหน่วยความจำก็จะเกี่ยวข้องกับการดูแลหน่วยความจำหลักเสมือนเป็นทรัพยากรที่จะต้องถูกจัดสรรและแบ่งปันระหว่างโพรเซสต่าง ๆ การใช้ซีพียูและอุปกรณ์รับ-ส่งข้อมูลให้มีประสิทธิภาพนั้น เราจำเป็นต้องนำโพรเซสเข้าไปใช้งานในหน่วยความจำให้มากที่สุดเท่าที่จะทำได้ และการอนุญาตของระบบให้โปรแกรมเมอร์สามารถเขียนโปรแกรมทำงานที่มีขนาดเท่าใดก็ได้ การที่เราจะทำตามข้อเสนอที่กล่าวมาได้นั้น เราจะต้องใช้หน่วยความจำเสมือน

เพราะในระบบที่มีหน่วยความจำเสมือนนั้น ตำแหน่งหน่วยความจำที่อ้างอิงถึงจะเป็นตำแหน่งหน่วยความจำทางตรรกะ ซึ่งสามารถเปลี่ยนแปลงเป็นตำแหน่งหน่วยความจำจริง เมื่อโพรเซสกำลังทำงานได้ (At run-time) การจัดการดังกล่าวทำให้ซีพียูสามารถใช้ข้อมูลที่อยู่ในหน่วยความจำหลักหรืออยู่ในที่ใดก็ได้ (เช่น ในฮาร์ดดิสก์) ที่สามารถทำการสลับโพรเซสเหล่านั้นให้เข้ามาทำงานได้ แนวความคิดของหน่วยความจำเสมือนนี้ จะอนุญาตให้โพรเซสถูกแยกส่วนออกเป็นชิ้นเล็ก ๆ ได้ และชิ้นส่วนเล็ก ๆ ของโพรเซส ไม่จำเป็นที่จะต้องอยู่ติดกันในหน่วยความจำหลักขณะที่ทำงาน และยิ่งกว่านั้นคือ โพรเซสยังสามารถทำงานได้ โดยไม่จำเป็นต้องให้ชิ้นส่วนเล็ก ๆ ทั้งหมดของโพรเซสนั้นอยู่ในหน่วยความจำหลักด้วย

แบบฝึกหัดท้ายบทที่ 8

จงตอบคำถามต่อไปนี้

1) จงอธิบายแนวคิดของหน่วยความจำเสมือนคืออะไรมีข้อดีอย่างไร

2) เมื่อเกิดการผิดหน้า (Page fault) จงอธิบายว่าระบบปฏิบัติการมีวิธีการดำเนินการอย่างไร

3) การสนับสนุนอุปกรณ์ทางฮาร์ดแวร์ เพื่อการจัดสรรหน้าตามคำร้องขอ มีวิธีการอย่างไร

4) ค่าบิตสถานะมีค่าเป็นใช้ได้ (Valid) แสดงว่าหน้านั้นอยู่ในหน่วยความจำหลัก แต่ถ้าบิตสถานะมีค่าเป็นใช้ไม่ได้ (Invalid) แสดงว่าหน้านั้นอยู่ในจานบันทึก จงอธิบายถึงประสิทธิภาพของระบบจัดสรรหน้าตามคำร้องขอคืออะไร และโอกาสที่จะทำให้เกิด Page fault มีสาเหตุจากอะไรบ้าง

5) ถ้าเวลาเฉลี่ยในการเข้าถึงหน่วยความจำเท่ากับ 2000 นาโนวินาที และเวลาเฉลี่ยของการจัดการการผิดหน้าเป็น 10 มิลลิวินาที จงหาเวลาเฉลี่ยในการอ้างอิงหน่วยความจำให้มีหน่วยเป็น นาโนวินาที

6) จงอธิบายถึงสาเหตุที่สำคัญของเทคนิคการบันทึกข้อมูลด้วยการคัดลอกข้อมูล

7) จงเปรียบเทียบการสับเปลี่ยนระหว่าง วิธีสับเปลี่ยนแบบที่ไม่ได้ใช้งานออกก่อน กับ วิธีการสับเปลี่ยนแบบใช้งานน้อยที่สุด-ออกก่อน ว่าแตกต่างกันอย่างไร

8) จงพิจารณาค่าการอ้างอิงถึงหน้าต่าง ๆ ดังนี้ 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. และตอบคำถาม มีจำนวน Page faults เกิดขึ้นกี่เฟรมในกรณีใช้อัลกอริทึมต่อไปนี้

8.1) LRU replacement

8.2) FIFO replacement

8.3) Optimal replacement

เอกสารอ้างอิง

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

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

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

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

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

M.Sc TU. Blog. Retrieved April 2, 2014 from http://msctu.blogspot.com/2010/03/