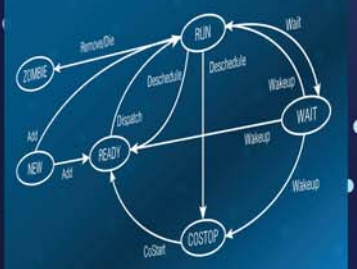
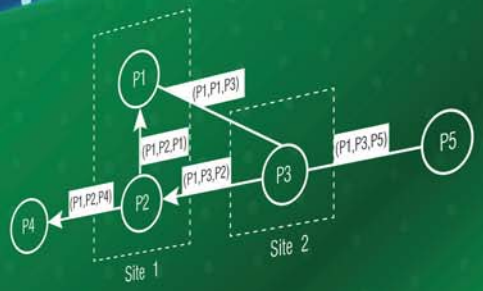
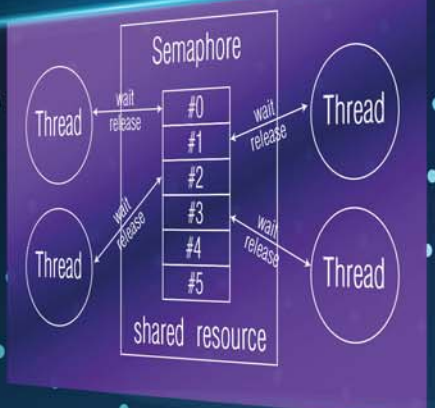
OPERATING SYSTEM

ระบบปฏิบัติการ







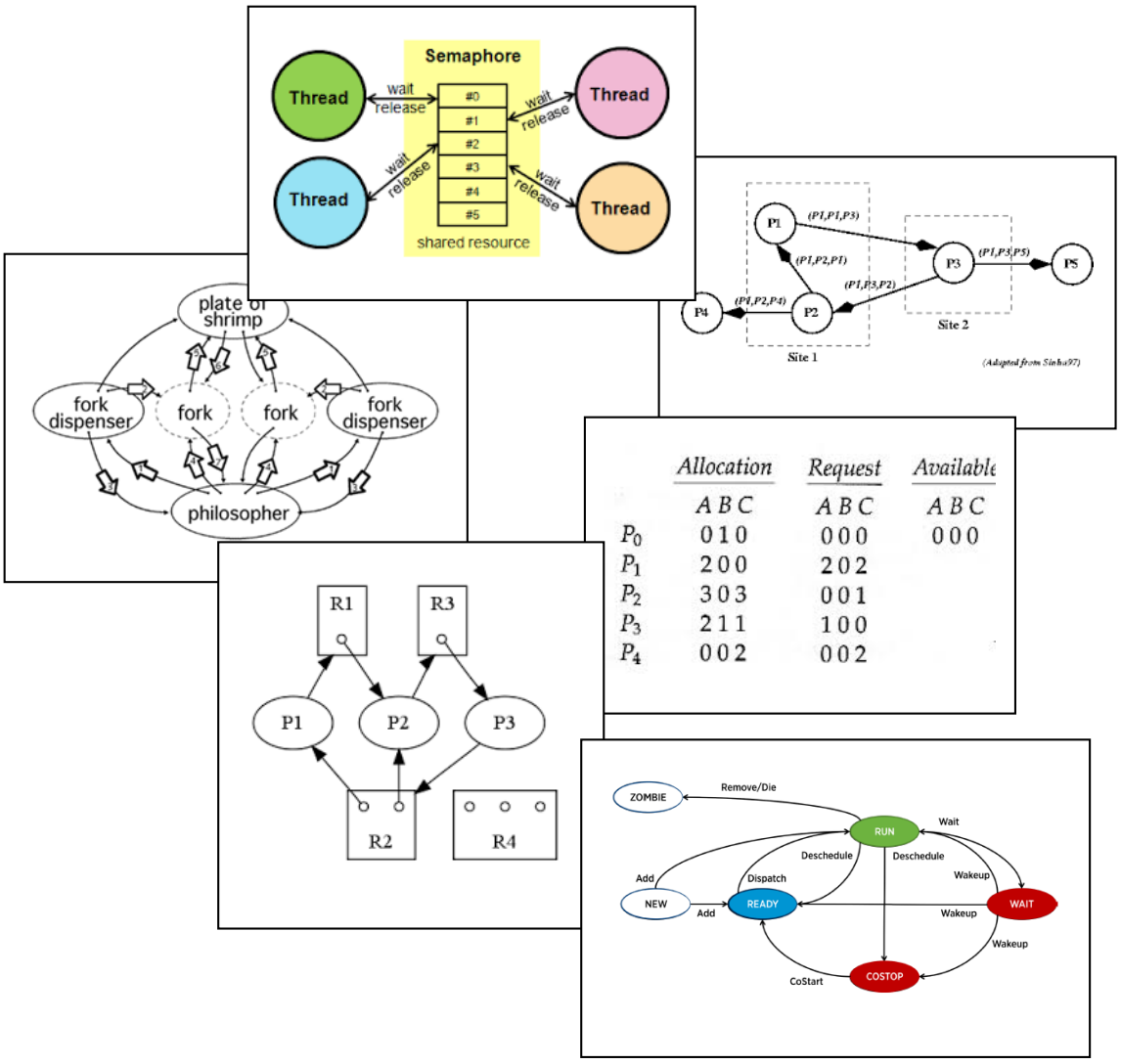
	Allocation	Request	Available
	A B C	A B C	A B C
			0 0 0
P_0	0 1 0	0 0 0	
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 2	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

ผศ.ปฏิคม ทองจริง

คณะวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ
มหาวิทยาลัยราชภัฏรำไพพรรณี

Operating Systems

ผศ. ปฏิคม ทองจริง



csit@rbru.ac.th

คำนำ

แม้ว่าในปัจจุบันระบบคอมพิวเตอร์จะมีความหลากหลายและเกิดขึ้นอย่างมากมาย ทั้งในเครื่องคอมพิวเตอร์ เครื่องคอมพิวเตอร์แม่ข่าย และอุปกรณ์แบบพกพา แต่หลักการขั้นพื้นฐานยังคงเป็นพื้นฐานหลักที่สามารถเป็นฐานความรู้ สามารถนำไปประยุกต์ใช้ได้อยู่เสมอ ทั้งนี้เพื่อแต่หลักการเหล่านี้จะนำไปถูกใช้ในเทคโนโลยีใดและตามความต้องการของผู้ใช้ระบบในแต่ละช่วงเวลา ดังนั้นการเรียนระบบปฏิบัติการคอมพิวเตอร์จึงถือเป็นการวางรากฐานความคิดความเข้าใจในระบบคอมพิวเตอร์

หนังสือระบบปฏิบัติการคอมพิวเตอร์เหมาะกับสาขาวิทยาการคอมพิวเตอร์ เทคโนโลยีสารสนเทศ ในระดับปริญญาตรี เพื่อให้เป็นแบบเรียนและแบบทดสอบความรู้ของนักศึกษา ในการเรียนรู้ระบบปฏิบัติการคอมพิวเตอร์ โดยในหนังสือว่าด้วยหลักการขั้นพื้นฐานที่สำคัญและการดำเนินการของระบบปฏิบัติการ ขั้นตอนการทำงานของโพรเซสต่าง ๆ การจัดการในการเข้าใช้หน่วยประมวลผล การจัดสรรหน่วยความจำ การจัดการแฟ้มข้อมูลและอุปกรณ์อินพุตเอาท์พุต เพื่อนำไปสู่ความรู้และความเข้าใจในเรื่องระบบปฏิบัติการเป็นอย่างดี

ทั้งนี้โครงสร้างของหนังสือได้แบ่งเนื้อหาไว้ 10 บท ประกอบด้วยเนื้อหาเรื่อง การจัดการโพรเซส การกำหนดใช้ซีพียู การจัดการหน่วยความจำ การจัดการแฟ้มข้อมูลและโครงสร้างของหน่วยเก็บข้อมูล ในทุกๆ บทของเอกสารประกอบการเรียนการสอนชุดนี้ จะมีแบบฝึกหัดที่เป็นโจทย์ปัญหาด้านต่างๆ เพื่อสำหรับเป็นการให้ผู้อ่านได้ฝึกฝน ทบทวน ค้นคว้า และทำความเข้าใจการแก้ไขปัญหาที่ให้ไว้ในแบบฝึกหัดท้ายบทในแต่ละบท

ต้องขอขอบพระคุณเป็นอย่างสูงต่ออาจารย์ทุกท่าน ที่ให้ความรู้และอบรมสั่งสอนในทุกระดับการศึกษา และขอขอบคุณต่อเจ้าของผลงานเอกสารตำราทุกท่าน ที่ผู้เขียนได้ใช้ศึกษาค้นคว้าและอ้างอิงในการเขียนครั้งนี้ หวังเป็นอย่างยิ่งหนังสือเล่มนี้สามารถใช้เป็นจุดเริ่มต้น ในการเรียนระบบปฏิบัติการ และหากได้อ่านทบทวนจะทำให้เกิดความรู้ความเข้าใจเป็นอย่างดี หากมีข้อผิดพลาดประการใดผู้เขียนต้องขออภัยไว้ ณ ที่นี้

ปฏิคม ทองจริง

กุมภาพันธ์ 2560

สารบัญ

หน้า

บทที่ 1
แนะนำพื้นฐานการทำงานระบบปฏิบัติการคอมพิวเตอร์

ปัจจุบันเครื่องคอมพิวเตอร์ เป็นอุปกรณ์ที่เข้ามามีส่วนเกี่ยวข้องกับการทำงานในด้านต่าง ๆ ไม่ว่าจะเป็นด้านการศึกษา ด้านการแพทย์ ด้านธุรกิจ ด้านวิทยาศาสตร์และเทคโนโลยี ฯลฯ พจนานุกรมฉบับราชบัณฑิตยสถาน พ.ศ. 2525 ให้ความหมายของคอมพิวเตอร์ไว้ว่า “เครื่องอิเล็กทรอนิกส์แบบอัตโนมัติทำหน้าที่เหมือนสมองกล ใช้สำหรับแก้ปัญหาต่าง ๆ ที่ง่ายและซับซ้อนโดยวิธีทางคณิตศาสตร์” ดังนั้นคอมพิวเตอร์จึงเป็นเครื่องจักรอิเล็กทรอนิกส์ที่ถูกสร้างขึ้นมาเพื่อใช้ทำงานแทนมนุษย์ในด้านการคิดคำนวณ และสามารถจำข้อมูลทั้งตัวเลขและตัวอักษรได้ เพื่อการเรียกใช้งานในครั้งต่อไป

นอกจากนี้เครื่องคอมพิวเตอร์ยังสามารถจัดการกับสัญลักษณ์ได้ด้วยความเร็วสูง โดยปฏิบัติตามขั้นตอนของโปรแกรม และมีความสามารถในด้านต่าง ๆ อีกมาก อาทิเช่น การเปรียบเทียบทางตรรกศาสตร์ การรับส่งข้อมูล การจัดเก็บข้อมูลในตัวเครื่องและสามารถประมวลผลจากข้อมูลต่าง ๆ ได้ สิ่งที่ทำให้เครื่องคอมพิวเตอร์ทำงานได้ต้องประกอบไปด้วย ฮาร์ดแวร์ ซอฟต์แวร์ และผู้ใช้งาน คำสั่งต่าง ๆ ที่สั่งให้คอมพิวเตอร์ทำงานคือสิ่งที่เราเรียกว่า ภาษาคอมพิวเตอร์ ดังนั้นนักศึกษาที่ศึกษาด้านคอมพิวเตอร์จึงมีความจำเป็นในการเรียนการเขียนโปรแกรมภาษาคอมพิวเตอร์ เพื่อให้อุปกรณ์อิเล็กทรอนิกส์นี้สามารถทำงานตามกระบวนการตามคำสั่งที่เราต้องการ

1.1 องค์ประกอบระบบคอมพิวเตอร์

เมื่อพิจารณาเครื่องคอมพิวเตอร์รวมถึงดูหลักการทำงานแล้ว จะพบว่าระบบคอมพิวเตอร์มีองค์ประกอบที่สำคัญอยู่ 4 องค์ประกอบ เพื่อทำให้เครื่องคอมพิวเตอร์สามารถทำงานได้ คือ

1.1.1 ฮาร์ดแวร์ (Hardware)

หมายถึง ส่วนประกอบเป็นเครื่องคอมพิวเตอร์ที่สามารถมองเห็นและสัมผัสได้ ถือเป็นทรัพยากรพื้นฐานที่สำคัญที่ต้องมีการเตรียมพร้อมสำหรับการเข้าถึงเพื่อการใช้งาน ซึ่งสามารถแบ่งได้ 4 ส่วน คือ

1. **หน่วยรับข้อมูล (Input unit)**

ทำหน้าที่รับข้อมูลหรือคำสั่งเข้าสู่เครื่องคอมพิวเตอร์ แล้วส่งไปเก็บไว้ในหน่วยความจำ เพื่อให้หน่วยประมวลผลกลางประมวลผล ตัวอย่างของหน่วยรับข้อมูล เช่น เมาส์ คีย์บอร์ด สแกนเนอร์ กล้องดิจิตอล และไมโครโฟน เป็นต้น

2. **หน่วยแสดงผล (Output unit)**

ทำหน้าที่แสดงผลลัพธ์ที่ได้จากการประมวลผลของหน่วยประมวลผลกลางที่เก็บอยู่ในหน่วยความจำ โดยรูปแบบการแสดงผลจะขึ้นอยู่กับความต้องการของผู้ใช้ ตัวอย่างของหน่วยแสดงผล เช่น จอภาพ เครื่องพิมพ์ ลำโพง และเครื่องฉายภาพสไลด์ เป็นต้น

3. หน่วยประมวลผลกลาง (Central Processing Unit : CPU)

หรือซีพียู หรือเรียกอีกชื่อหนึ่งว่า โปรเซสเซอร์ (Processor) เป็นอุปกรณ์ที่มีความสำคัญมากที่สุดของฮาร์ดแวร์ ทำหน้าที่ประมวลผลคำสั่งและควบคุมการทำงานทั้งหมดของระบบคอมพิวเตอร์

4. หน่วยความจำ (Memory unit)

ทำหน้าที่จัดเก็บข้อมูลหรือคำสั่งที่รับเข้ามา เพื่อส่งต่อไปยังซีพียูและเมื่อซีพียูประมวลผลเสร็จแล้ว จะนำผลลัพธ์ที่ได้มาเก็บไว้ในหน่วยความจำ เพื่อนำไปแสดงผลทางอุปกรณ์แสดงผล หรือจัดเก็บลงหน่วยความจำสำรอง เพื่อเรียกข้อมูลนำกลับมาใช้งานต่อไป หน่วยความจำแบ่งเป็น 2 ชนิด คือ

1. **หน่วยความจำหลัก (Primary Storage/Main Memory)** ทำหน้าที่เก็บข้อมูลและคำสั่งต่าง ๆ ในโปรแกรมเพื่อรอการประมวลผล และเก็บสารสนเทศที่ผ่านการประมวลผลแล้ว ก่อนที่จะส่งไปยังอุปกรณ์ส่งออก (Output device) นอกจากนี้หน่วยความจำหลักยังแบ่งออกเป็น 2 ประเภทหลัก คือ **ROM (Read Only Memory)** เป็นหน่วยความจำที่เก็บข้อมูลไว้แบบถาวร (Nonvolatile memory) ไม่สามารถลบได้ด้วยวิธีธรรมดาทั่วไป ข้อมูลใน ROM จะยังคงถูกเก็บอยู่ได้โดยไม่ต้องมีไฟฟ้าไปเลี้ยง เช่น BIOS เป็นต้น และ **RAM (Random Access Memory)** เป็นหน่วยความจำที่เก็บข้อมูลไว้แบบชั่วคราว (Volatile memory) เมื่อไม่มีกระแสไฟหรือเมื่อปิดเครื่อง ข้อมูลที่อยู่ใน RAM จะหายไป โดย RAM ใช้เก็บข้อมูลหรือชุดคำสั่งจากโปรแกรมใน ระหว่างที่เครื่องคอมพิวเตอร์กำลังทำงาน เช่น DDR RAM และนอกจากนี้ยัง มีหน่วยความจำขนาดเล็กที่เรียกว่าหน่วยความจำ Cache ทำให้หน่วยประมวลผลกลางสามารถเข้าถึงและค้นหาข้อมูลได้เร็วกว่าหน่วยความจำหลัก
2. **หน่วยความจำสำรอง (Secondary Storage)** เป็นอุปกรณ์ที่สามารถเก็บข้อมูลและโปรแกรมไว้ได้ แม้ว่าจะปิดเครื่องคอมพิวเตอร์ไปแล้วก็ตาม สื่อที่ใช้ในการเก็บหน่วยความจำ สำรองที่นิยมใช้ในปัจจุบัน คือ ฮาร์ดดิสก์ แฟลชไดรฟ์ (Flash drive) เป็นต้น

1.1.2 ซอฟต์แวร์ (Software)

ซอฟต์แวร์ คือ โปรแกรมหรือชุดคำสั่งที่สั่งให้ฮาร์ดแวร์ทำงานรวมไปถึงการควบคุมการทำงานของอุปกรณ์แวดล้อมต่าง ๆ เช่น ฮาร์ดดิสก์ ดิสก์ไดร์ฟ ซีดีรอม การ์ดอินเตอร์เฟสต่าง ๆ เป็นต้น ซอฟต์แวร์เป็นชุดคำสั่งหรือโปรแกรมที่ใช้สั่งงานให้คอมพิวเตอร์ทำงาน ดังนั้นจึงเป็นสิ่งที่จำเป็นและมีความสำคัญมาก และเป็นส่วนประกอบหนึ่งที่ทำให้ระบบสารสนเทศเป็นไปได้ตามที่ต้องการ แบ่งเป็น 2 ประเภทคือ

1. ระบบปฏิบัติการ (Operating System)

เมื่อผู้ใช้ต้องการติดต่อหรือใช้งานแอปพลิเคชั่นโปรแกรม ระบบปฏิบัติการ (Operating System : OS) จะช่วยให้แอปพลิเคชั่นโปรแกรมต่าง ๆ สามารถติดต่อกับฮาร์ดแวร์ของเครื่องได้ ซอฟต์แวร์ระบบเป็นซอฟต์แวร์ที่ทำงานอยู่เบื้องหลัง การดำเนินการของคอมพิวเตอร์ช่วยให้คอมพิวเตอร์จัดการกับทรัพยากรภายในเครื่องได้ ระบบปฏิบัติการทำหน้าที่ประสานงานทรัพยากรประเภทต่าง ๆ ในคอมพิวเตอร์ จัดเตรียมส่วนติดต่อระหว่างผู้ใช้กับคอมพิวเตอร์ รวมถึงการดำเนินงานกับแอพพลิเคชั่น

โปรแกรมซอฟต์แวร์ประยุกต์ ระบบปฏิบัติการที่รู้จักกันแพร่หลายมากที่สุดสำหรับผู้ใช้เครื่องคอมพิวเตอร์ เช่น ระบบปฏิบัติการวินโดว์ (Windows) ระบบปฏิบัติการไอโอเอส (IOS) และระบบปฏิบัติการรหัสเปิดต่าง ๆ (Open sources) เช่น ระบบปฏิบัติการลินุกส์ (Linux) เป็นต้น

ภาพที่ 1.2 รายชื่อและสัญลักษณ์ของระบบปฏิบัติการต่าง ๆ

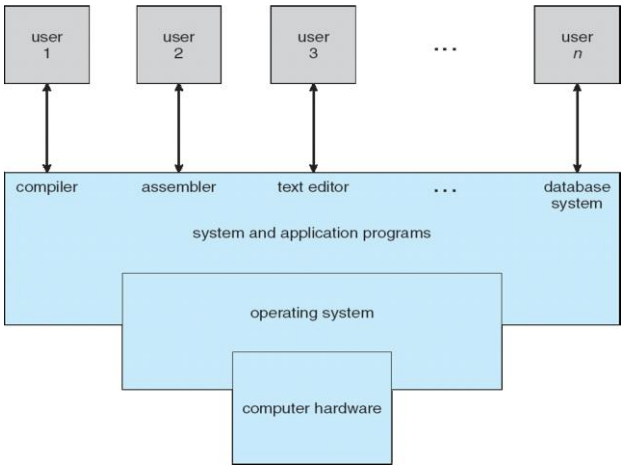
2. โปรแกรมประยุกต์ (Application Programs)

คือซอฟต์แวร์หรือโปรแกรมที่ถูกเขียนขึ้นเพื่อการทำงานเฉพาะอย่างที่เราต้องการ เช่น งานส่วนตัว งานทางด้านธุรกิจ งานทางด้านวิทยาศาสตร์ โปรแกรมทางธุรกิจ เกมส์ต่าง ๆ ระบบฐานข้อมูล ตลอดจนตัวแปลภาษา เราอาจเรียกโปรแกรมประเภทนี้ว่า User's Program โปรแกรมประเภทนี้โดยส่วนใหญ่ มักใช้ภาษาระดับสูงในการพัฒนา เช่น ภาษา C, C++, Java ฯลฯ ตัวอย่างของโปรแกรมที่พัฒนาขึ้นเพื่อใช้งาน เช่น โปรแกรมไมโครซอฟต์ออฟฟิศ โปรแกรมตกแต่งภาพ โปรแกรมสแกนไวรัส โปรแกรมในทางธุรกิจ เช่น โปรแกรมระบบบัญชีเงินเดือน (Payroll program) โปรแกรมระบบสินค้าคงคลัง (Stock program) ฯลฯ ซึ่งแต่ละโปรแกรมก็จะมีเงื่อนไขตามความต้องการหรือกฎเกณฑ์ของแต่ละหน่วยงานที่ใช้ โปรแกรมประเภทนี้เราสามารถดัดแปลงแก้ไขเพิ่มเติม (Modifications) ในบางส่วนของโปรแกรมเองได้ เพื่อให้ตรงกับความต้องการของผู้ใช้งานโปรแกรม โปรแกรมเหล่านี้เป็นตัวกำหนดแนวทางในการใช้ทรัพยากรระบบ เพื่อทำงานต่าง ๆ ให้แก่ผู้ใช้หลากหลายประเภท ซึ่งอาจเป็นได้ทั้งบุคคล โปรแกรม หรือเครื่องคอมพิวเตอร์ เช่นตัวแปลภาษาต้องใช้ทรัพยากรระบบในการแปลโปรแกรมภาษาระดับสูงให้เป็นภาษาเครื่องแก่โปรแกรมเมอร์ ดังนั้น ระบบปฏิบัติการต้องควบคุมและประสานงานในการใช้ทรัพยากรระบบของผู้ใช้ให้เป็นไปอย่างถูกต้อง

ภาพที่ 1.3 ผลิตภัณฑ์ซอฟต์แวร์ประยุกต์ต่าง ๆ

3. ผู้ใช้ (Users)

แม้ว่าระบบคอมพิวเตอร์จะประกอบด้วยองค์ประกอบทั้งทางด้านฮาร์ดแวร์และซอฟต์แวร์ แต่ระบบคอมพิวเตอร์จะไม่สามารถทำงานได้ ถ้าขาดอีกองค์ประกอบหนึ่ง ซึ่งได้แก่ องค์ประกอบทางด้านบุคลากรที่จะเป็นผู้จัดการและควบคุมระบบคอมพิวเตอร์ให้สามารถปฏิบัติงานได้อย่างราบรื่น คอยแก้ไขปัญหาต่าง ๆ ที่เกิดขึ้นกับระบบคอมพิวเตอร์ พัฒนาโปรแกรมประยุกต์ต่าง ๆ รวมไปถึงการใช้งานโปรแกรมประยุกต์ที่ถูกพัฒนาขึ้น



ภาพที่ 1.4 องค์ประกอบของระบบคอมพิวเตอร์
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.4)

4. ข้อมูลและสารสนเทศ

ข้อมูล หมายถึง ข้อเท็จจริงหรือเหตุการณ์ที่เกิดขึ้นแล้วใช้ตัวเลขตัวอักษรหรือสัญลักษณ์ต่าง ๆ ทำความหมายแทนสิ่งเหล่านั้น เช่น คะแนนสอบวิชาภาษาอังกฤษของนักศึกษา อายุของพนักงานในบริษัทคอมพิวเตอร์จำกัด ราคาขายของหนังสือในร้านหนังสือดอก และคำตอบที่ผู้ถูกสำรวจตอบในแบบสอบถาม เป็นต้น

สารสนเทศ (Information) หมายถึง ข้อสรุปต่าง ๆ ที่ได้จากการนำข้อมูลมาทำการวิเคราะห์หรือผ่านวิธีการที่ได้กำหนดขึ้น ทั้งนี้เพื่อนำข้อสรุปไปใช้งานหรืออ้างอิง เช่น เกรดเฉลี่ยของวิชาภาษาอังกฤษของนักศึกษา อายุเฉลี่ยของพนักงานในบริษัทคอมพิวเตอร์จำกัด ราคาขายสูงสุดของหนังสือในร้านหนังสือ และข้อสรุปจากการสำรวจคำตอบในแบบสอบถาม เป็นต้น

1.2 ความหมายระบบปฏิบัติการ

กลุ่มโปรแกรมที่ได้รับการจัดระเบียบเพื่อทำหน้าที่ควบคุมการทำงานของระบบ และสนับสนุนการทำงานในส่วนของฮาร์ดแวร์ โดยใช้เป็นตัวเชื่อมโยงระหว่างเครื่องคอมพิวเตอร์และผู้ใช้ ทั้งนี้เพื่อ

อำนวยความสะดวกในการพัฒนาและการใช้งานโปรแกรมต่าง ๆ รวมถึงการจัดสรรทรัพยากรต่าง ๆ ในระบบให้มีประสิทธิผลที่ดี ในลักษณะที่ผู้ใช้ไม่จำเป็นต้องทราบกลไกการทำงานหรือฮาร์ดแวร์ของระบบ

1.2.1 หน้าที่ของระบบปฏิบัติการ

หน้าที่ของระบบปฏิบัติการสามารถแบ่งหน้าที่ออกเป็น 3 หัวข้อหลักด้วยกัน คือ

1. จัดสรรทรัพยากรที่ใช้ร่วมกัน

คือ การจัดการทรัพยากรทั้งหมดไม่ว่าจะเป็นหน่วยความจำ หน่วยประมวลผล ฯลฯ ให้สามารถใช้ทรัพยากรเหล่านี้ร่วมกันได้ ดังนั้นการเข้าใช้ทรัพยากรเหล่านี้ร่วมกันในลักษณะของระบบ Multiprogramming ระบบปฏิบัติการจะต้องมีหน้าที่ในการทำให้เกิดความยุติธรรมต่อการเข้าใช้ทรัพยากรและขจัดหรือป้องกันความขัดแย้งต่าง ๆ ทั้งนี้หากมีของผิดพลาด เพียงโปรแกรมหนึ่งจะต้องไม่มีผลกระทบกับส่วนอื่น ๆ ในระบบคอมพิวเตอร์ โดยคำนึงประสิทธิภาพโดยรวมของระบบเป็นหลักสำคัญในการควบคุมการทำงานของโปรแกรม

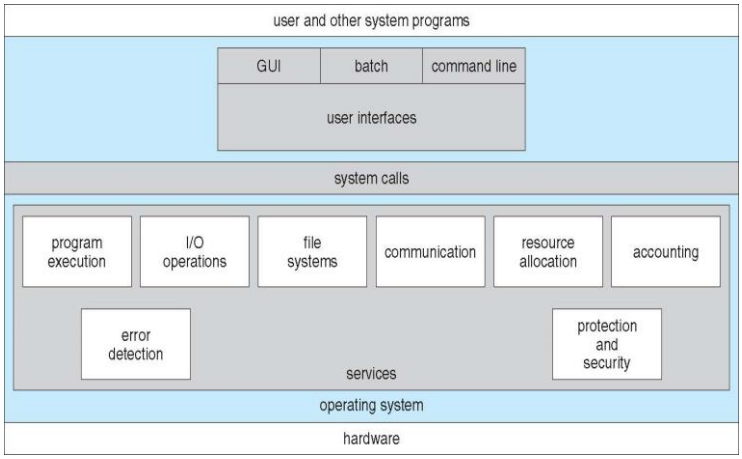
2. ควบคุมการทำงานของโปรแกรมต่าง

เพื่อป้องกันข้อผิดพลาดของโปรแกรมและ การใช้งานที่ไม่เหมาะสมในเครื่องคอมพิวเตอร์ ระบบปฏิบัติการมีโปรแกรมย่อยมากมาย ที่ทำหน้าที่ควบคุมการทำงานของอุปกรณ์ต่าง ๆ โดยเฉพาะอุปกรณ์รับข้อมูลและแสดงผลของระบบคอมพิวเตอร์ เช่น เมาส์ คีย์บอร์ด เครื่องพิมพ์ จอภาพ ผู้ใช้งานไม่จำเป็นต้องเขียนโปรแกรมเพื่อควบคุมอุปกรณ์ดังกล่าวด้วยตนเอง โดยสามารถเรียกใช้งานโปรแกรมย่อยนั้น ๆ ในส่วนเคอร์เนล (Kernel) ของระบบปฏิบัติการ

3. เป็นโปรแกรมที่ทำงานอยู่ตลอดเวลาบนเครื่องคอมพิวเตอร์

เคอร์เนลจะเป็นโปรแกรมแก่นสำคัญของระบบปฏิบัติการ ซึ่งคอยดูแลบริหารทรัพยากรของระบบ ติดต่อกับฮาร์ดแวร์และซอฟต์แวร์ เนื่องจากว่าเป็นส่วนประกอบพื้นฐานของระบบปฏิบัติการ เป็นฐานล่างสุดในการติดต่อกับทรัพยากรต่าง ๆ เช่น หน่วยความจำ หน่วยประมวลผลกลาง และ อุปกรณ์อินพุตเอาต์พุต โดยภายในเคอร์เนลจะประกอบไปด้วยโมดูล (module) ต่าง ๆ และบางครั้งเราอาจจะเรียกโมดูลเหล่านี้ว่า ไดรเวอร์ (Driver) ซึ่งมีหน้าที่เป็นตัวกลางในการติดต่อกันระหว่างแอพพลิเคชั่นหรือระบบปฏิบัติการกับอุปกรณ์ฮาร์ดแวร์ทั้งหมด ทั้งภายในและนอกเครื่องคอมพิวเตอร์ (ตัวสั่งการที่ทำงานควบคู่กับฮาร์ดแวร์ตลอดเวลา)

เคอร์เนล ได้รวมถึง Interrupt handler ซึ่งดูแลคำขอหรือประมวลผลการทำงานของ Input/Output เพื่อจัดลำดับการทำงานให้เคอร์เนล และดูแลแต่ละขั้นตอนเมื่อประมวลผลเคอร์เนล มักจะรวมถึงการจัดการตำแหน่งของระบบปฏิบัติการ ในหน่วยความจำและอุปกรณ์เก็บข้อมูล เพื่อจัดสรรสำหรับส่วนประกอบต่าง ๆ และโปรแกรมประยุกต์ เรียกว่า System call หรือ Monitor call

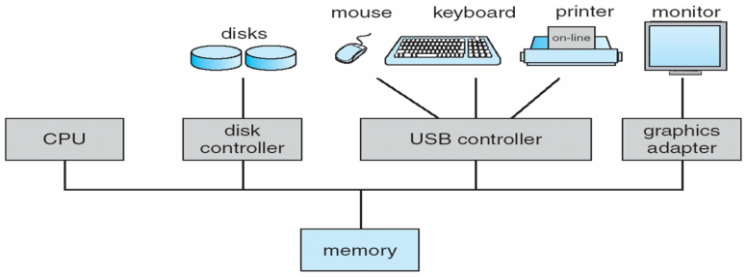


ภาพที่ 1.5 บริการต่าง ๆ ของระบบปฏิบัติการ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p56)

1.3 การทำงานของระบบคอมพิวเตอร์

เมื่อเปิดเครื่องคอมพิวเตอร์ จะมีการโหลดโปรแกรมขึ้นมาทำงานเรียกโปรแกรมนี้ว่า “Bootstrap program” โดยปกติโปรแกรมนี้จะถูกเก็บไว้ใน ROM หรือ EPROM หรือที่เรารู้จักโปรแกรมนี้ในชื่อว่า “Firmware” การทำงานของโปรแกรมนี้เมื่อเราเริ่มเปิดหรือรีบู๊ตเครื่องคอมพิวเตอร์ โปรแกรมจะเริ่มทำการตรวจสอบคุณสมบัติของอุปกรณ์ภายในเครื่องคอมพิวเตอร์ว่าพร้อมทำงานหรือไม่ เช่น หน่วยความจำ ฮาร์ดดิสก์ และเมื่อผ่านกระบวนการตรวจคุณสมบัติของฮาร์ดแวร์เรียบร้อยแล้ว จะโหลดระบบปฏิบัติการและส่วนของเคอร์เนลเข้าสู่หน่วยความจำคอมพิวเตอร์เพื่อพร้อมในการทำงานต่อไป

ระบบคอมพิวเตอร์ยุคใหม่จะมีหน่วยประมวลผลกลาง ตัวควบคุมอุปกรณ์ (Device controller) ซึ่งเชื่อมโยงกันผ่านระบบบัส (Common bus) โดยมีการใช้หน่วยความจำร่วมกัน (Share memory) ดังแสดงในภาพที่ 1.6



ภาพที่ 1.6 ส่วนประกอบของระบบคอมพิวเตอร์ในปัจจุบัน
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.7)

จากภาพที่ 1.6 สามารถอธิบายได้ว่าหน่วยประมวลผลกลาง หรือซีพียู ตัวควบคุมอุปกรณ์เชื่อมต่อกับระบบบัส เพื่อเป็นเส้นทางในการรับส่งข้อมูลเข้าสู่หน่วยความจำที่ใช้ร่วมกัน ดังนั้นการประมวลผลพร้อมกันของซีพียูและอุปกรณ์ ขึ้นอยู่กับรอบการเข้าใช้หน่วยความจำ

1.3.1 การดำเนินการของระบบคอมพิวเตอร์

การดำเนินการของระบบคอมพิวเตอร์ ตัวอุปกรณ์อินพุตเอาท์พุต และซีพียูสามารถถูกดำเนินการพร้อมกันได้ ตัวควบคุมอุปกรณ์แต่ละตัวเป็นตัวควบคุมเฉพาะประเภทของอุปกรณ์นั้น ๆ เช่น Disk controller จะควบคุมการทำงานของดิสก์ USB controller ควบคุมรับส่งข้อมูลของอุปกรณ์ที่เชื่อมต่อ USB ตัวควบคุมอุปกรณ์แต่ละตัวจะมีหน่วยความจำของตัวเองหรือที่เรียกว่า บัฟเฟอร์ (Buffer) ซีพียูเคลื่อนย้ายข้อมูลเข้าออกจากหน่วยความจำหลัก และหน่วยความจำหลักเคลื่อนย้ายข้อมูลเข้าออกจาก Local buffer ของตัวควบคุมอุปกรณ์ ข้อมูลจากอุปกรณ์อินพุตและเอาท์พุตนำเข้าสู่ Local buffer ของตัวควบคุมอุปกรณ์ ตัวควบคุมอุปกรณ์ทำการแจ้งซีพียูให้ทราบหลังการทำงานได้เสร็จสิ้น โดยการทำให้เกิดสัญญาณการขัดจังหวะหรือเรียกว่า การอินเตอร์รัพ (Interrupt)

อินเตอร์รัพเป็นสัญญาณจากอุปกรณ์ที่ต่อกับเครื่องคอมพิวเตอร์ หรือรูปแบบโปรแกรมภายในเครื่องคอมพิวเตอร์ ที่ทำให้โปรแกรมหลักหรือระบบปฏิบัติการซึ่งควบคุมเครื่องคอมพิวเตอร์หยุด และตรวจสอบข้อมูลข่าวสารจากสัญญาณการอินเตอร์รัพให้ทำอะไรต่อไป เครื่องคอมพิวเตอร์เกือบทั้งหมดในปัจจุบัน เป็นระบบ Interrupt driven โดยระบบ เริ่มทำงานตามรายการคำสั่งต่อไปจนกระทั่งเสร็จสิ้นการทำงาน หรือเริ่มทำงานตามรายการคำสั่งต่อไปจนกระทั่ง เมื่อสัญญาณอินเตอร์รัพถูกส่งออกมาจำเป็นต้องหยุดรอเพื่อดูคำสั่งจากสัญญาณอินเตอร์รัพนั้น และภายหลังการทำงานตามสัญญาณอินเตอร์รัพเสร็จสิ้นคอมพิวเตอร์กลับมาทำงานต่อไปใหม่โดยให้โปรแกรมทำงานต่อไป หรือให้โปรแกรมอื่นเริ่มต้นทำงาน

เมื่อซอฟต์แวร์หรือฮาร์ดแวร์ต้องการให้ซีพียูพักจากงานที่ทำอยู่เพื่อมาทำงานตามการร้องขอของตนก่อน ซอฟต์แวร์หรือฮาร์ดแวร์เหล่านี้จะทำการส่งสัญญาณที่เรียกว่า การร้องขออินเตอร์รัพ (Interrupt request) ให้แก่คอมพิวเตอร์ เมื่อได้รับสัญญาณการร้องขออินเตอร์รัพนั้นแล้ว จะทำ การสนองตอบสัญญาณโดยการข้ามไปทำการประมวลผลโปรแกรมย่อยที่เรียกว่า Interrupt Service Routine (ISR) ตำแหน่งเริ่มต้นของ ISR แต่ละอย่างจะมีค่าขึ้นอยู่กับชนิดของอินเตอร์รัพที่เกิดขึ้น โดยค่าตำแหน่งที่อยู่เริ่มต้นเหล่านี้จะถูกระบุอยู่ใน Interrupt Vector แยกตามชนิดของอินเตอร์รัพที่เกิดขึ้น

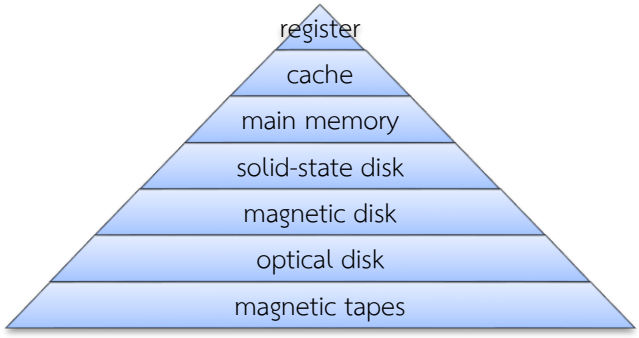
ระบบปฏิบัติการมักจะมีรหัสที่เรียกว่าอินเตอร์รัพแฮนเดิล (Interrupt handle) ซึ่งทำการตรวจสอบก่อนว่าสัญญาณนั้นเป็นสัญญาณขัดจังหวะประเภทใด จากนั้นก็โยกย้ายการควบคุมไปยังตำแหน่งเริ่มต้นของส่วนการบริการการขัดจังหวะประเภทนั้น ๆ ซึ่ งการจัดการงานนี้ต้องทำอย่างรวดเร็ว โดยใช้ตารางเก็บตำแหน่งส่วนบริการการขัดจังหวะ (Interrupt vector table) ที่อยู่ในหน่วยความจำหลัก ซึ่งข้อมูลในตารางนี้เป็นตำแหน่งที่อยู่เริ่มต้นของส่วนบริการการขัดจังหวะแต่ละประเภท ระบบปฏิบัติการยูนิกซ์และไมโครซอฟต์ดอส ก็อาศัยกลไกการขัดจังหวะนี้

1.3.2 โครงสร้างหน่วยเก็บข้อมูล (Storage Structure)

ในทางอุดมคติ เราต้องการให้โปรแกรมและข้อมูลอาศัยอยู่ในหน่วยความจำหลัก (Main memory) อย่างถาวร แต่ในการทำงานจริงไม่สามารถทำได้ด้วยเหตุผล 2 ประการ คือ 1) หน่วยความจำ

หลัก มีขนาดเล็กเกินไปที่จะสามารถเก็บโปรแกรมและข้อมูลที่จำเป็นทั้งหมดได้อย่างถาวร 2) หน่วยความจำหลัก เป็นอุปกรณ์ที่ใช้เก็บข้อมูลแบบชั่วขณะ เป็นหน่วยความจำแบบลบเลือนได้ คือ ถ้าเป็นหน่วยความจำที่เก็บข้อมูลไว้แล้ว หากไม่มีไฟฟ้าจ่ายให้กับวงจรหน่วยความจำ ข้อมูลที่เก็บไว้จะหายไปหมด เพราะเมื่อปิดเครื่องข้อมูลก็หาย

ดังนั้นคอมพิวเตอร์จึงต้องมีหน่วยเก็บข้อมูลสำรอง (Secondary storage) หน่วยความจำไม่ลบเลือนความหมายคือ หน่วยความจำเก็บข้อมูลได้ โดยไม่ขึ้นกับไฟฟ้าที่เลี้ยงวงจร เพื่อเก็บข้อมูลมาก ๆ ได้อย่างถาวร ทำให้คอมพิวเตอร์ในปัจจุบันจะมีหน่วยเก็บข้อมูลสำรองซึ่งสามารถเก็บข้อมูลจำนวนมาก อย่างไรก็ดีความเร็วในการอ่านและบันทึกข้อมูลของหน่วยเก็บข้อมูลสำรองจะต่ำกว่าหน่วยความจำหลัก โดยสามารถจัดลำดับของหน่วยความจำตามปัจจัยหลักคือ ความเร็ว ต้นทุน และประเภทแบบ Volatile หรือ Nonvolatile ได้ดังภาพที่ 1.7



ภาพที่ 1.7 ระดับชั้นของหน่วยเก็บข้อมูลสำรอง

1. เทปแม่เหล็ก (Magnetic Tape)

จะอ่านข้อมูลตามลำดับก่อนหลังตามที่ได้บันทึกไว้ เรียกหลักการนี้ว่าการเข้าถึงข้อมูลตามลำดับ (Sequential access) การทำงานลักษณะนี้จึงเป็นข้อเสียของการใช้เทปแม่เหล็กบันทึกข้อมูล คือทำให้อ่านข้อมูลได้ช้า เนื่องจากต้องอ่านข้อมูลในม้วนเทปไปเรื่อย ๆ จนถึงตำแหน่งที่ต้องการ ผู้ใช้จึงนิยมนำเทปแม่เหล็กมาสำรองข้อมูลเท่านั้น ส่วนข้อมูลที่กำลังใช้งานจะถูกเก็บอยู่บนหน่วยเก็บข้อมูลแบบจานแม่เหล็ก (Magnetic disk) เพื่อให้เรียกใช้ได้ง่าย และนำเฉพาะข้อมูลที่สำคัญและไม่ถูกเรียกใช้บ่อยมาเก็บสำรอง (Backup) ไว้ในเทปแม่เหล็ก เพื่อป้องกันการสูญหายของข้อมูล

2. จานแม่เหล็ก (Magnetic Disk)

จานแม่เหล็กสามารถเก็บข้อมูลได้เป็นจำนวนมาก และมีคุณสมบัติในการเข้าถึงข้อมูลโดยตรง (Direct access) จานแม่เหล็กจะต้องใช้คู่กับตัวขับจานแม่เหล็ก หรือ ดิสก์ไดรฟ์ (Disk drive) ซึ่งเป็นอุปกรณ์สำหรับอ่านเขียนจานแม่เหล็ก จานแม่เหล็กเป็นสื่อที่ใช้หลักการของการเข้าถึงข้อมูลแบบสุ่ม (Random access) นั่นคือ ถ้าต้องการข้อมูลลำดับที่ 52 หัวอ่านก็จะตรงไปที่ข้อมูลนั้นและอ่านข้อมูลนั้นขึ้นมาใช้งานทันที ทำให้มีความเร็วในการอ่านและบันทึกที่สูงกว่าเทปแม่เหล็กมาก หัวอ่านของดิสก์ไดรฟ์เรียกว่า หัวอ่านและบันทึก (Read/Write head) เมื่อผู้ใช้ใส่แผ่นจานแม่เหล็กเข้าไปในดิสก์ไดรฟ์ ก่อนที่จะใช้แผ่นจานแม่เหล็กเก็บข้อมูล จะต้องผ่านขั้นตอนของการฟอร์แมต (Format) ก่อน เพื่อเตรียมแผ่นจานแม่เหล็กให้พร้อมสำหรับเครื่องรุ่นที่จะใช้ การฟอร์แมตจัดเป็นงานพื้นฐานหนึ่งของระบบปฏิบัติการ

3. ออปติคัลดิสก์ (Optical Disk)

ใช้เทคโนโลยีของแสงเลเซอร์ ทำให้สามารถเก็บข้อมูลได้จำนวนมาก ในปัจจุบันจะมีออปติคอลอยู่หลายแบบ ซึ่งใช้เทคโนโลยีที่แตกต่างกันไป พัฒนาจนถึง ดีวีดี (Digital Versatile Disk : DVD) เป็นเทคโนโลยีที่ได้รับความนิยมอย่างมาก แผ่นดีวีดีสามารถเก็บข้อมูลได้ต่ำสุดที่ 4.7 จิกะไบต์ ซึ่งเพียงพอสำหรับเก็บภาพยนตร์เต็มเรื่องด้วยคุณภาพสูงสุดทั้งภาพและเสียง (ในขณะที่ CD-ROM หรือ Laser disk ต้องใช้จำนวนหลายแผ่น) ทำให้ดีวีดีจะสามารถมีความจุได้ตั้งแต่ 4.7 กิกะไบต์ ถึง 17 กิกะไบต์ และมีความเร็วในการเข้าถึง (Access time) อยู่ที่ 600 กิโลไบต์ต่อวินาที ถึง 1.3 เมกะไบต์ต่อวินาที

4. Solid-State Disk (SSD)

เป็นการใช้ชิปหน่วยความจำเก็บข้อมูลแทนจานแม่เหล็กในฮาร์ดดิสก์ เหมือนแฟลชไดรฟ์ ที่เราใช้กันอยู่ในทุกวันนี้มีส่วนประกอบสำคัญ 2 ส่วน คือ ชิปหน่วยความจำ กับชิปคอนโทรลเลอร์ ข้อดีของ SSD ใช้เวลาเข้าถึงข้อมูลน้อยกว่าฮาร์ดดิสก์ เพราะไม่มีหัวอ่าน ไม่มีจานแม่เหล็ก ไม่ต้องหมุน สามารถเข้าถึงข้อมูลได้เลย ปัจจุบันมีความเร็วในการอ่านถึง 120 เมกะไบต์/วินาที และความเร็ว ในการเขียนที่ 100 เมกะไบต์/วินาที ซึ่งเกือบจะเร็วเท่าฮาร์ดดิสก์ที่เร็วที่สุดอยู่แล้ว และสามารถพัฒนาไปได้อีก เสียงเงียบเพราะไม่มีชิ้นส่วนที่เคลื่อนไหวได้เหมือนฮาร์ดดิสก์ ทนแรงกระแทกและการสั่นสะเทือน ทนต่ออุณหภูมิที่สูงกว่า มีน้ำหนักเบา ที่สำคัญคือ ปัญหาเรื่องการแตกกระจายของการจัดเก็บแฟ้มข้อมูล (File fragmentation) ไม่มีผลต่อความเร็วของ SSD

5. หน่วยความจำแคช (Cache memory)

เป็นหน่วยความจำแบบ Random access ซึ่งซีพียูสามารถเข้าถึงข้อมูลได้เร็วกว่าหน่วยความจำ (RAM) แบบปกติ ในการประมวลผลข้อมูล ซีพียูจะมองหาข้อมูลในหน่วยความจำแคช (Cache memory) ก่อน เพื่อทำให้การประมวลผลเร็วขึ้น หน่วยความจำแคชในบางครั้งมีการอธิบายว่าเป็นหน่วยความจำระดับที่ใกล้และเข้าถึงได้ง่ายจากซีพียู ในส่วน L1 และ L2 cache ที่อยู่บนซีพียูมักจะเป็น Static RAM (SRAM) ในขณะที่หน่วยความจำหลักเป็น Dynamic RAM (DRAM) ในส่วนของ SRAM จะไม่มีการ Refresh ตัวเอง

6. รีจิสเตอร์ (Register)

รีจิสเตอร์ คือ หน่วยเก็บข้อมูลขนาดจิ๋ว ซึ่งเป็นส่วนหนึ่งของซีพียู และใช้เป็นที่เก็บข้อมูลชั่วคราวสำหรับการส่งผ่านผลลัพธ์ของคำสั่งหนึ่งไปยังคำสั่งถัดไปในซีพียู หรือโปรแกรมอีกโปรแกรมหนึ่งซึ่งอยู่ภายใต้การควบคุมของระบบปฏิบัติการ รีจีสเตอร์จำเป็นต้องมีขนาดใหญ่เพียงพอที่จะเก็บคำสั่งได้ในบางกรณี รีจีสเตอร์อาจมีขนาดเล็ก เช่น มีขนาดเป็นครึ่งหนึ่งของคำสั่ง ขึ้นอยู่กับการออกแบบสถาปัตยกรรมของซีพียูนั้น ๆ

1.3.3 โครงสร้างอินพุตและเอาต์พุต (Input/Output Structure)

เมื่ออินพุตและเอาท์พุตหรือเขียนย่อว่า I/O เริ่มทำงาน ซีพียูจะโหลดรีจิสเตอร์ที่จำเป็นมาไว้ในตัวควบคุมอุปกรณ์ ซึ่งตัวควบคุมอุปกรณ์จะทำการตรวจสอบรีจิสเตอร์เหล่านั้น เพื่อกำหนดว่าจะทำงานอะไร โดยมีการรับส่งข้อมูล 2 แบบคือ

1. การรับ-ส่งข้อมูลแบบสัมพันธ์

เมื่อการรับส่งข้อมูลเริ่มขึ้น จะทำการโยกย้ายการควบคุมให้กับโปรแกรมของผู้ใช้ การรับส่งข้อมูลจะทำได้อีกครั้งหลังจากเสร็จการรับส่งข้อมูลเท่านั้น ในการรอรับส่งข้อมูลเสร็จมี 2 วิธี คือ 1) คอมพิวเตอร์บางเครื่องมีชุดคำสั่ง Wait พิเศษ ซึ่งปล่อยให้ซีพียูว่าง จนกระทั่งเกิดสัญญาณขัดจังหวะถัดไป 2) เครื่องจักรที่ไม่มีชุดคำสั่งดังกล่าว อาจจะมี Wait loop ถ้าซีพียูต้องรอให้การรับส่งข้อมูลเสร็จก่อนเสมอ แสดงว่าต้องมีการร้องขอของอินพุตหรือเอาท์พุตอยู่หนึ่งตัวที่เด่นอยู่ตลอดเวลา ดังนั้นเมื่อเกิดสัญญาณขัดจังหวะการรับส่งข้อมูล ระบบปฏิบัติการจะรู้ทันทีว่าอุปกรณ์กำลังถูกขัดจังหวะ แต่ไม่สามารถประมวลผลอินพุตหรือเอาท์พุตของอุปกรณ์หลาย ๆ ตัวพร้อมกันได้

2. การรับส่งข้อมูลแบบไม่สัมพันธ์

เมื่อการรับส่งข้อมูลเริ่มขึ้น การโยกย้ายการควบคุมให้กับโปรแกรมของผู้ใช้ ทำได้โดยไม่ต้องรอให้การรับส่งข้อมูลเสร็จ คำสั่งร้องขอระบบปฏิบัติการ อนุญาตให้โปรแกรมของผู้ใช้รอคอยให้รับส่งข้อมูลเสร็จ ตารางที่ระบบปฏิบัติการใช้เก็บบันทึกของอุปกรณ์รับส่งข้อมูลแต่ละตัว คือ Device status table ซึ่งใช้แสดงประเภทของอุปกรณ์ ที่อยู่ และสถานะ (ว่าง กำลังทำงาน หรือเสีย)

1.3.4 การเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access : DMA)

โดยปกติเมื่อมีการส่งข้อมูลให้กับอุปกรณ์ต่าง ๆ ข้อมูลนั้นจะถูกเก็บไว้ในหน่วยความจำ จากนั้นซีพียูจะอ่านข้อมูลจากหน่วยความจำ เพื่อส่งไปให้อุปกรณ์ที่กำหนด ในทางกลับกันเมื่ออุปกรณ์ต้องการส่งข้อมูลให้โพรเซส ข้อมูลจะถูกส่งผ่านซีพียูไปยังหน่วยความจำส่วนนี้ จากนั้นโพรเซสจึงจะนำข้อมูลไปใช้ได้ ซึ่งจะเห็นว่าการทำงานในลักษณะนี้ทำได้ช้าและเปลืองเวลาการทำงานของซีพียู ดังนั้นจึงมีแนวคิดใหม่ในการรับส่งข้อมูลจากอุปกรณ์อินพุตและเอาท์พุตเข้าไปยังหน่วยความจำโดยตรงโดยไม่ต้องส่งผ่านซีพียู ซึ่งจะทำให้การรับส่งข้อมูลทำได้เร็วขึ้น และยังจะสามารถใช้ซีพียูในการทำงานโพรเซสอื่น ๆ ได้ วิธีการเช่นนี้เรียกว่า การเข้าถึงหน่วยความจำโดยตรง

การรับข้อมูลแบบเข้าถึงหน่วยความจำโดยตรงต้องอาศัยแชนแนล โดยแชลแนลเป็นฮาร์ดแวร์รูปแบบหนึ่งซึ่งติดตั้งอยู่ในตัวควบคุมอุปกรณ์ แชลแนลจะทำหน้าที่แทนซีพียูในงานที่เกี่ยวข้องกับการรับส่งข้อมูลแบบเข้าถึงหน่วยความจำโดยตรง (ในกรณีที่อุปกรณ์นั้นสามารถเข้าถึงหน่วยความจำโดยตรงได้) ซึ่งเมื่อระบบมีความต้องการรับส่งข้อมูลแบบเข้าถึงหน่วยความจำโดยตรง แชลแนลจะส่งสัญญาณไปบอกซีพียูให้รับรู้ว่ามีเหตุการณ์ที่ต้องการจะรับส่งข้อมูล เมื่อซีพียูทราบว่าจะมีการรับส่งข้อมูลแบบเข้าถึงหน่วยความจำโดยตรง ซีพียูสั่งให้แชลแนลทำงานในรูทีนที่เกี่ยวข้องกับการรับส่งข้อมูล (มอบให้แชลแนลดำเนินการ) จากนั้นซีพียูจะไปทำงานอื่น (เนื่องจากไม่ต้องมาคอยควบคุมการทำงานในส่วนที่ แชลแนลรับผิดชอบ) และเมื่อแชลแนลทำการรับส่งข้อมูลเสร็จ แชลแนลจะส่งสัญญาณบอกให้ซีพียูรับรู้อีกครั้งว่าการทำงานเสร็จสิ้นแล้ว สำหรับกรณีอุปกรณ์ที่ไม่มีคุณสมบัติในการเข้าถึงหน่วยความจำโดยตรงได้ การรับส่งข้อมูลระหว่างอุปกรณ์นั้นกับหน่วยความจำดำเนินการผ่านซีพียูตามปกติ

1.4 สถาปัตยกรรมระบบคอมพิวเตอร์

1.4.1 ระบบหน่วยประมวลผลเดี่ยว (Single Processor System)

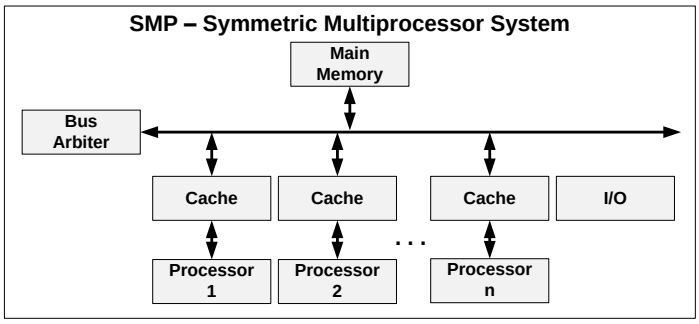
ปกติโดยทั่วไปของระบบคอมพิวเตอร์ที่ใช้ในการประมวลผลข้อมูล และทำงานตามขั้นตอนของโปรแกรม ใช้เพียง 1 ซีพียูในการทำงานซึ่งเพียงพอต่อปริมาณงาน แต่ปัจจุบันมีการใช้งานด้าน Graphic และด้าน Multimedia กันมาก มีการประมวลผลมากขึ้น จึงทำให้เกิดปัญหาคอขวดที่เรียกกันว่าปัญหา Bottom neck (ซีพียูประเมินผลไม่ทันกับข้อมูลที่ถูกส่งเข้ามา) ประกอบกับการเพิ่มความถี่ให้กับซีพียูยังมีข้อจำกัดด้านความร้อน จึงเกิดเทคโนโลยีระบบหลายหน่วยประมวลผล (Multiprocessor systems) ขึ้นเพื่อมาช่วยในการประมวลผลมากกว่าเพิ่มสัญญาณความถี่ให้กับซีพียูเหมือนที่ผ่านมา

1.4.2 ระบบหลายหน่วยประมวลผล (Multiprocessor System)

เป็นระบบคอมพิวเตอร์ที่มีการเพิ่มแกน (Core) ในการประมวลผลให้มีมากกว่า 1 แกนบน Chip เดียวกัน แต่ละตัวทำงานเป็นอิสระจากกัน มีการใช้ทรัพยากรของระบบร่วมกัน เช่น Dual-Core processor, Quad-Core processor ประโยชน์ของระบบหลายหน่วยประมวลผล ทำให้เพิ่มปริมาณงาน (Throughput) เพราะว่าระบบคอมพิวเตอร์ที่มี 2 ซีพียูและแต่ละซีพียูทำงานต่างกัน ดังนั้นในเวลาที่เท่ากัน ระบบที่ใช้จำนวนซีพียูมากกว่า ย่อมให้ปริมาณงานที่มากกว่า ทำให้ระบบมีความน่าเชื่อถือมากขึ้น (Reliability) ด้วยการกำหนดให้ทุกซีพียูทำงานเดียวกัน เพื่อเป็นการตรวจสอบความถูกต้องในการทำงาน และมีซีพียูสำรอง ในกรณีที่เกิดความเสียหายกับซีพียูหลัก และประหยัดค่าใช้จ่าย ด้วยการให้ซีพียูหลายตัวใช้ทรัพยากรของระบบร่วมกัน ระบบหลายหน่วยประมวลผลแบ่งเป็น 2 ประเภท

1. ประมวลผลแบบสมมาตร (Symmetric Multiprocessing)

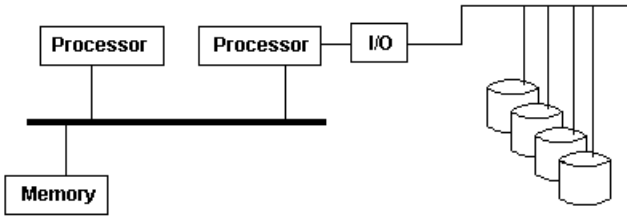
เป็นการประมวลผลโดยใช้ซีพียูมากกว่า 1 ตัว โดยที่แต่ละซีพียูทำงานเท่ากัน ไม่มีซีพียูตัวใดรับโหลดหรือทำงานมากกว่าตัวอื่น และใช้ระบบปฏิบัติการเดียวกันทุกซีพียู



ภาพที่ 1.8 ไดอะแกรมของระบบหลายหน่วยประมวลผล แบบแบบสมมาตร
ที่มา: Wikipedia, the free encyclopedia. File:SMP - Symmetric Multiprocessor System.svg

2. ประมวลผลแบบไม่สมมาตร (Asymmetric Multiprocessing)

เป็นการประมวลผลโดยใช้ซีพียูมากกว่า 1 ตัว โดยมีซีพียูตัวหนึ่งเป็นตัวหลัก ทำหน้าที่บริหารจัดสรรทรัพยากร และแบ่งงานให้ซีพียูตัวอื่น ๆ ทำงาน



ภาพที่ 1.9 ไดอะแกรมของระบบระบบหลายหน่วยประมวลผล แบบไม่สมมาตร
ที่มา: Wikipedia, the free encyclopedia. Retrieved June 2, 2014 from http://en.wikipedia.org/wiki/File:Asmp_2.gif

1.4.3 ระบบคลัสเตอร์ (Clustered Systems)

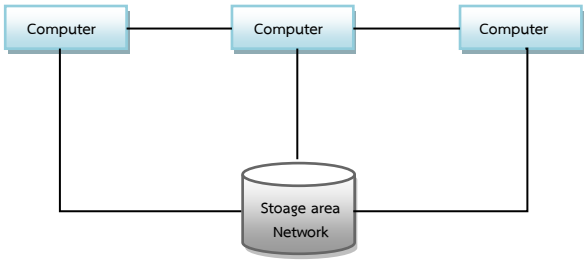
เป็นการเชื่อมต่อระบบการทำงานของกลุ่มคอมพิวเตอร์เข้าด้วยกัน ภายใต้ระบบเครือข่ายความเร็วสูง มีความสามารถในการกระจายงานที่ทำไปยังเครื่องภายในระบบ เพื่อให้การประมวลผลมีประสิทธิภาพสูงขึ้น โดยอาจเทียบเท่าซุปเปอร์คอมพิวเตอร์หรือสูงกว่า สำหรับการประมวลผลงานที่มีความซับซ้อนโดยเฉพาะงานด้านวิทยาศาสตร์ เช่น การจำลองโครงสร้างของโมเลกุลทางเคมี การวิเคราะห์เกี่ยวกับตำแหน่งการเกิดพายุสุริยะ การวิเคราะห์ข้อมูลที่มีขนาดใหญ่เป็นต้น ถ้าดูตามโครงสร้างแล้ว ระบบคลัสเตอร์ คือคอมพิวเตอร์แบบขนานที่มีหน่วยความจำแยกนั่นเอง โครงสร้างของระบบคลัสเตอร์แบ่งเป็น 2 ชนิด คือ

1. ระบบคลัสเตอร์แบบปิด

คลัสเตอร์จะซ่อนระบบทั้งหมดและจะต่อผ่านเกตเวย์สู่โลกภายนอก ข้อดี คือ มีความปลอดภัยสูงและใช้อินเตอร์เน็ตแอดเดรสเพียงแอดเดรสเดียวเท่านั้น ข้อเสีย คือ แต่ละโหนดในระบบไม่สามารถช่วยกันบริหารข้อมูลจากภายนอกได้

2. ระบบคลัสเตอร์แบบเปิด

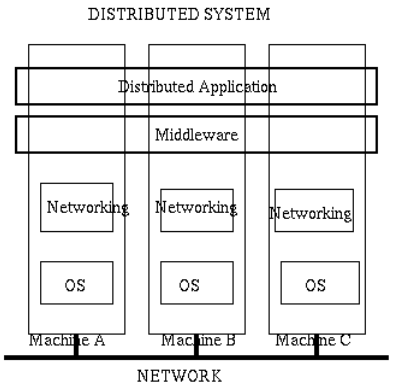
คลัสเตอร์จะติดต่อกับเครือข่ายภายนอกโดยตรง ทำให้ผู้ใช้เข้าถึงทุกโหนดในระบบได้โดยตรง ระบบซอฟแวร์ที่ถูกใช้ในระบบคลัสเตอร์ควรเป็นโปรแกรมแบบขนาน การทำงานโปรแกรมแบบขนานบนระบบคลัสเตอร์นั้นจะใช้วิธีการที่เรียกว่า โปรแกรมแบบส่งผ่านข้อความ (Message passing) การทำงานของโปรแกรมในลักษณะนี้ทำได้โดยการกระจายงานขนาดใหญ่ไปยังหลาย ๆ เครื่องให้ทำงานพร้อมกัน และใช้การแลกเปลี่ยนข่าวสารผ่านเครือข่ายในการติดต่อระหว่างกลุ่มของโปรแกรมที่ช่วยกันทำงาน ระบบโปรแกรมแบบขนานที่ใช้ในปัจจุบันคือ แบบ MPI (Message Passing Interface) และ แบบ PVM (Parallel Virtual Machine) ซึ่งมีการทำงานที่แตกต่างกัน ทำให้สามารถสร้างและใช้ขีดความสามารถของระบบคลัสเตอร์ได้เพิ่มมากขึ้นด้วย



ภาพที่ 1.10 โครงสร้างทั่วไปของระบบ Cluster Systems

1.4.4 ระบบแบบกระจาย (Distributed System)

เป็นระบบคอมพิวเตอร์ที่แต่ละซีพียูมีทรัพยากรเป็นของตัวเอง มีการนำคอมพิวเตอร์แต่ละเครื่องมาเชื่อมต่อกันด้วยระบบเครือข่าย แล้วแจกจ่ายงานให้กับซีพียูที่มีอยู่ ประโยชน์ของระบบแบบกระจาย คือ การแชร์ทรัพยากร (Resource sharing) เป็นการลดค่าใช้จ่ายในการซื้ออุปกรณ์ เนื่องจากสามารถใช้ทรัพยากรร่วมกันได้ เช่น เครื่องพิมพ์ มีการทำงานเร็วขึ้นถ้าเครื่องคอมพิวเตอร์ที่ทำงานอยู่มีงานโอเวอร์โหลด (Overload) จะทำการส่งงานบางส่วนไปยังคอมพิวเตอร์เครื่องอื่น เรียกวิธีการนี้ว่า “Load sharing” ทำให้มีความน่าเชื่อถือถ้าเครื่องคอมพิวเตอร์เครื่องใดเครื่องหนึ่งในระบบเสีย สามารถโอนข้อมูลไปทำงานที่คอมพิวเตอร์เครื่องอื่น ๆ ได้โดยไม่ต้องรอ



ภาพที่ 1.11 ไดอะแกรมของระบบแบบกระจาย
ที่มา: Temple University. CIS 307: Introduction to Distributed Systems. Retrieved June 2, 2014 from http://www.cis.temple.edu/~giorgio/cis307/readings/client-server.html

1.5 โครงสร้างระบบคอมพิวเตอร์

ระบบปฏิบัติการมีการพัฒนาควบคู่ไปกับระบบคอมพิวเตอร์ จากคอมพิวเตอร์ยุคแรกที่มีขนาดใหญ่ ใช้หลอดสุญญากาศ และไม่มีการติดตั้งระบบปฏิบัติการ พัฒนาจนถึงยุคที่คอมพิวเตอร์มีขนาดเล็ก มี

โครงสร้างของระบบปฏิบัติการที่สามารถนำมาใช้งานได้อย่างมีประสิทธิภาพ โดยสามารถแบ่งระบบปฏิบัติการตามคุณสมบัติการทำงานได้ ดังนี้

1.5.1 ระบบที่ไม่มีระบบปฏิบัติการ (Non Operating System)

อยู่ในช่วงเวลาของคอมพิวเตอร์ยุคแรก ๆ คอมพิวเตอร์จะไม่มีระบบปฏิบัติการ ผู้ใช้ต้องเขียนโปรแกรมเพื่อควบคุมการทำงานทั้งหมด ทำให้ใช้ประโยชน์จากคอมพิวเตอร์ได้น้อยมาก (Low utilization) งานที่ได้จะขาดความน่าเชื่อถือ (Low reliability) ต้องมีโอเปอเรเตอร์ (Operator) เพื่อทำหน้าที่รวบรวมงานและเตรียมระบบสำหรับผู้ใช้งาน

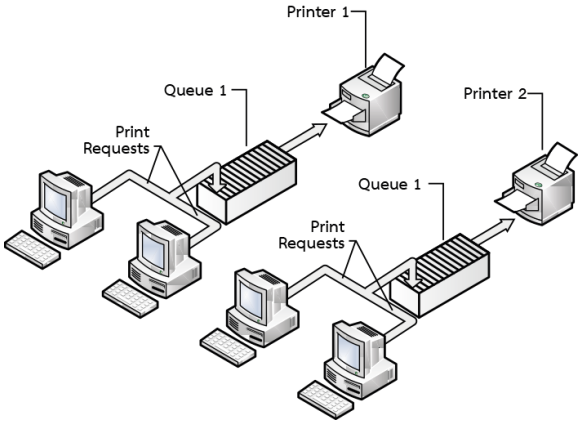
1.5.2 ระบบกลุ่มอย่างง่าย (Simple Batch System)

คอมพิวเตอร์มีขนาดใหญ่ ทำการรับข้อมูลจากคอนโซล (Console) มีการพัฒนาอุปกรณ์สำหรับนำเข้าข้อมูล และอุปกรณ์สำหรับนำข้อมูลออก เช่น เครื่องอ่านบัตร เครื่องพิมพ์ เทปไดรฟ์ ผู้ใช้ไม่ได้ติดต่อกับระบบคอมพิวเตอร์โดยตรง แต่เป็นเพียงผู้เตรียมข้อมูลเขียนโปรแกรม ข้อมูลสำหรับควบคุมระบบ มีโอเปอเรเตอร์ทำหน้าที่รวบรวมงานที่จะประมวลผล จัดเรียงลำดับงานเป็นกลุ่ม งานมักอยู่ในรูปบัตรเจาะรู แล้วจึงส่งงานทั้งหมดเข้าระบบ โดยจะถูกประมวลผลทีละงาน หรือทีละกลุ่มงาน (Batch) การทำงานในระบบนี้ ซีพียู จะว่าง (Idle) บ่อยมาก เนื่องจากความเร็วของซีพียูและอุปกรณ์รับส่งมีความแตกต่างกันมาก ทำให้ซีพียูต้องหยุดรอ เพื่อให้อุปกรณ์ในการอ่านข้อมูลทำงานเสร็จก่อน การใช้ประโยชน์จากซีพียูอยู่ในระดับต่ำมาก และมีการหน่วงเวลา (Delay) เกิดขึ้นระหว่างช่วงที่รันงานจนถึงงานเสร็จ เรียกว่า “Turnaround time”

เนื่องจากการทำงานแบบ Batch system การใช้ประโยชน์ของหน่วยประมวลผลต่ำมาก จึงมีการปรับแก้ไขปัญหานี้ได้แก่การปรับอัตรา (Buffering) ในการทำงานจะให้อุปกรณ์รับส่งข้อมูลทำงานไปพร้อม ๆ กับการประมวลผลของซีพียู โดยขณะที่ซีพียูประมวลผลคำสั่งหนึ่ง อุปกรณ์รับส่งข้อมูลจะนำเข้าข้อมูลที่ต้องการใช้งานต่อไปที่ซีพียู นำเข้าไปเก็บไว้ในหน่วยความจำเรียกว่า บัพเฟอร์ ทำให้ซีพียูสามารถทำงานต่อได้ทันที โดยไม่ต้องหยุดรออุปกรณ์รับส่งข้อมูล มีปัญหาในเรื่องความแตกต่างระหว่างความเร็วของซีพียูกับอุปกรณ์รับส่งอยู่ เนื่องจากซีพียูมีความเร็วสูงกว่าอุปกรณ์รับส่งข้อมูลมาก และประเภทของงานที่ซีพียูประมวลผลนั้นอาจเป็นงานที่เน้นการใช้งานซีพียู (CPU bound) หรือเน้นการใช้งานอุปกรณ์รับส่งข้อมูล (I/O bound)

1.5.3 ระบบสพูลลิ่ง (Spooling System)

พยายามแก้ปัญหาความแตกต่างระหว่างความเร็วของซีพียูกับอุปกรณ์รับส่งข้อมูลโดยให้มีการถ่ายข้อมูลไปยังอุปกรณ์รับส่งข้อมูลที่มีความเร็วสูงกว่า เช่น เทปแม่เหล็ก เมื่อโปรแกรมต้องการใช้ข้อมูล ระบบปฏิบัติการจะสั่งให้ซีพียูไปอ่านข้อมูลที่เทปแม่เหล็กแทน ทำให้ประสิทธิภาพในการทำงานของซีพียูสูงขึ้นเล็กน้อย แต่การทำงานของโปรแกรมต้องผ่านขั้นตอนมากขึ้น และการเข้าถึงข้อมูลบนเทปแม่เหล็กต้องเป็นแบบลำดับ ต่อมามีการคิดค้นดิสก์ขึ้นมาจึงได้เปลี่ยนไปใช้ดิสก์แทนเทปแม่เหล็ก ซึ่งทำให้สามารถเข้าถึงข้อมูลได้โดยตรงทำให้ระบบทำงานได้รวดเร็วมากขึ้น มีการนำเอาระบบ Spooling ไปประยุกต์ใช้งานด้านอื่น ๆ เพื่อเพิ่มประสิทธิภาพการทำงานของระบบ เช่น การสั่งพิมพ์งานออกทางเครื่องพิมพ์

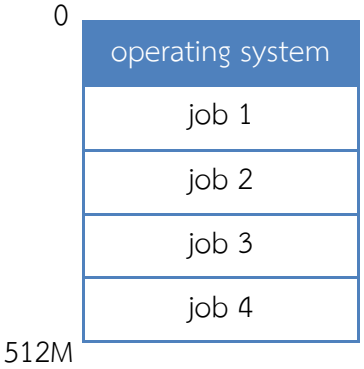


ภาพที่ 1.12 การประยุกต์ใช้ระบบ Spooling ในการพิมพ์ออกเครื่องพิมพ์
ที่มา: Aficionado. Retrieved June 5, 2014 from
http://aficionadous.blogspot.com/2009/04/spooling-os-spoofing-internet.html

1.5.4 ระบบรองรับการทำงานหลายโปรแกรม (Multiprogramming System)

จากการใช้ทรัพยากรอย่างไม่เต็มประสิทธิภาพ เพราะสามารถทำงานได้เพียงครั้งละ 1 งาน เพื่อให้สามารถทำได้หลายงานพร้อม ๆ กัน โดยในการทำงานจะมีโปรแกรมประมวลผลมากกว่า 1 งาน อยู่ภายในหน่วยความจำหลัก

ระบบปฏิบัติการจะทำหน้าที่เลือกงานหรือโปรแกรมเข้าไปประมวลผลที่ซีพียูทันทีที่ซีพียูว่าง จากนั้นอาจต้องรอการอ่านเทปหรือรอการทำงานของอินพุตเอาท์พุต ระบบปฏิบัติการจะสวิตซ์ (Switch) ไปทำงานที่สอง เมื่องานที่สองต้องรอ ทำให้ซีพียูจะสลับไปทำงานอีกงานต่อไปเรื่อย ๆ จนวนมาถึงคิวของงานแรก ทำให้ซีพียูไม่มีช่วงเวลาว่างเลย ระบบหลายโปรแกรมช่วยให้มีการใช้งานทรัพยากรของระบบโดยเฉพาะซีพียูอย่างเต็มประสิทธิภาพ (High utilization)



ภาพที่ 1.13 แสดงงานในหน่วยความจำหลักในระบบ Multiprogramming System

1.5.5 ระบบแบ่งส่วนเวลา (Time Sharing System)

ปัญหาของระบบหลายโปรแกรม คือ หากโปรแกรมหรืองานที่เข้าไปทำงานที่ซีพียูมีขนาดใหญ่ หรือมีการทำงานที่ซีพียูเป็นเวลานาน จะทำให้โปรแกรมอื่น ๆ ที่จะเข้าไปทำงานที่ซีพียูต้องรอ จึงกำหนดให้มีระบบการแบ่งเวลา (Time sharing) สำหรับแต่ละโปรแกรมหรือแต่ละงาน ในการเข้าไปทำงานที่ซีพียูในระยะเวลาที่กำหนด ระบบคอมพิวเตอร์ที่มีการทำงานในระบบหลายโปรแกรมร่วมกับระบบการแบ่งเวลานั้น จะช่วยให้ระบบสามารถให้บริการผู้ใช้ได้หลายคนพร้อม ๆ กัน โดยผู้ใช้แต่ละคนจะสลับกันเข้าไปใช้งานซีพียู เนื่องจากซีพียูทำงานด้วยความเร็วสูง ทำให้ผู้ใช้งานรู้สึกเหมือนเป็นเจ้าของระบบทั้งหมด

1.5.6 ระบบโต้ตอบฉับพลัน (Real Time System)

เป็นระบบที่ใช้ในงานเฉพาะเจาะจง เช่น งานทดลองวิทยาศาสตร์ ระบบภาพทางการแพทย์ งานควบคุมทางอุตสาหกรรม คำนึงถึงอัตราเวลาการตอบสนอง (Response time) เป็นสำคัญ โดยมีการกำหนดระยะเวลาที่จะต้องทำงานให้เสร็จภายในเวลาที่กำหนด มีผลให้ซีพียูมีการใช้งานที่ต่ำมาก เนื่องจากระบบต้องให้ซีพียูว่างหรือเกือบว่างตลอดเวลา เพื่อที่ระบบจะได้สามารถประมวลผลงานทันที เมื่อมีข้อมูลเข้ามาในระบบโต้ตอบฉับพลัน แบ่งเป็น 2 ประเภท คือ

1. Hard Real-Time System เป็นระบบที่กำหนดเวลาไว้แน่นอน เพื่อให้ระบบทำงานได้เสร็จ หากว่าระบบไม่สามารถทำงานเสร็จได้ตามเวลาที่กำหนด จะเกิดปัญหาร้ายแรง

2. Soft Real-Time System เป็นระบบที่กำหนดเวลาไว้แน่นอนเช่นกัน แต่ถ้าระบบทำงานไม่เสร็จภายในเวลาที่กำหนดไว้ จะไม่เกิดปัญหาร้ายแรงเท่ากับระบบที่ทำงานแบบ Hard Real-Time

1.6 สรุป

ในส่วนของระบบปฏิบัติการเป็นกลุ่มของโปรแกรมที่จัดการและเตรียมพร้อมส่วนของฮาร์ดแวร์เพื่อให้ผู้ใช้งานสามารถใช้โปรแกรมได้อย่างมีประสิทธิภาพ มีองค์ประกอบที่สำคัญอยู่ 4 องค์ประกอบเพื่อทำให้เครื่องคอมพิวเตอร์สามารถทำงานได้ คือ ฮาร์ดแวร์ ระบบปฏิบัติการ โปรแกรมประยุกต์ และผู้ใช้งาน ซึ่งจะต้องมีการทำงานร่วมกันอยู่ตลอดเวลา ระบบคอมพิวเตอร์ยุคใหม่จะมีซีพียู ตัวควบคุมอุปกรณ์ ซึ่งเชื่อมโยงกันผ่าน Common bus ซึ่งมีการใช้หน่วยความจำร่วมกัน

โครงสร้างของหน่วยความจำในทางอุดมคติ โปรแกรมและข้อมูลอาศัยอยู่ในหน่วยความจำหลักอย่างถาวร แต่ในการทำงานจริงไม่สามารถทำได้ ด้วยเหตุผลหน่วยความจำหลักมีขนาดเล็กเกินไปที่จะสามารถเก็บโปรแกรมและข้อมูลที่จำเป็นทั้งหมดได้อย่างถาวร หน่วยความจำหลักเป็นอุปกรณ์ที่ใช้เก็บข้อมูลแบบชั่วขณะ เป็นหน่วยความจำแบบลบเลือนได้จึงต้องมีหน่วยเก็บข้อมูลสำรอง หน่วยความจำไม่ลบเลือนคือ หน่วยความจำเก็บข้อมูลได้ โดยไม่ขึ้นกับไฟฟ้าที่เลี้ยงวงจรเพื่อเก็บข้อมูลมาก ๆ ได้อย่างถาวร ทำให้คอมพิวเตอร์ในปัจจุบันจะมีหน่วยเก็บข้อมูลสำรองซึ่งสามารถเก็บข้อมูลจำนวนมาก ตามปัจจัยหลัก คือ ความเร็ว ต้นทุน และประเภทของหน่วยความจำแบบ Volatile หรือแบบ Nonvolatile

สถาปัตยกรรมระบบคอมพิวเตอร์ ปกติโดยทั่วไปของระบบคอมพิวเตอร์ที่ใช้ในการประมวลผลข้อมูลและรันโปรแกรม ที่ใช้เพียง 1 ซีพียูในการทำงานซึ่งเพียงพอต่อปริมาณงาน ดังนั้นในเวลาที่เท่ากันระบบที่ใช้จำนวนซีพียูมากกว่า ย่อมให้ปริมาณงานที่มากกว่า เพิ่มความน่าเชื่อถือของระบบด้วยการกำหนดให้ทุกซีพียูทำงานเดียวกัน ระบบหลายหน่วยประมวลผลแบ่งเป็น 2 ประเภท คือ ประมวลผลแบบสมมาตร และประมวลผลแบบไม่สมมาตร โครงสร้างระบบคอมพิวเตอร์ได้มีวิวัฒนาการมาอย่างต่อเนื่อง ทั้งนี้เพื่อให้การทำงานของระบบคอมพิวเตอร์มีประสิทธิภาพมากยิ่งขึ้น และในปัจจุบันได้นำระบบ ระบบแบบกระจาย ได้เป็นระบบที่มีการศึกษาและนำมาใช้งานมาก เช่น Cloud computing ซึ่งเป็นสิ่งที่จะต้องเรียนรู้ต่อไป

แบบฝึกหัดท้ายบทที่ 1

1. จงบอกรายละเอียดของระบบคอมพิวเตอร์มีองค์ประกอบที่สำคัญกี่องค์ประกอบ อะไรบ้าง

2. จงอธิบายระบบปฏิบัติการทำงานอย่างไร และบอกระบบปฏิบัติการที่รู้จักทั้งแบบรหัสเปิดและเชิงพาณิชย์มา 10 ระบบปฏิบัติการ

3. จงอธิบายขั้นตอนการทำงานของ Bootstrap program

4. จงอธิบายถึงความสามารถในการจัดลำดับของหน่วยความจำ ตามปัจจัยหลักด้านใดบ้าง

5. จงอธิบายการอินเตอร์รัพคืออะไรมีหลักการทำงานอย่างไร

6. Dual Core, Core 2 Dual และ Quad Core แตกต่างกันอย่างไร และทำงานอย่างไร จงอธิบายมาให้เข้าใจ

7. จงอธิบายระบบคลัสเตอร์คืออะไร และบอกระบบปฏิบัติการที่สนับสนุนระบบนี้

8. จงอธิบายการประมวลผลแบบสมมาตรกับการประมวลผลแบบไม่สมมาตร ทำงานแตกต่างกันอย่างไร

9. จงอธิบายระบบแบบกระจายคืออะไรทำงานอย่างไร

10. จงอธิบายระบบ Cloud computing คืออะไรมีองค์ประกอบอะไรบ้าง

บรรณานุกรมประจำบทที่ 1

บุญสืบ โพธิ์ศรีและคณะ. (2550). **เทคโนโลยีสารสนเทศเบื้องต้น.** นนทบุรี : เจริญรุ่งเรืองการพิมพ์.

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

วีกีพีเดีย. Retrieved March 2, 2014 from http://th.wikipedia.org/wiki/คอมพิวเตอร์

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

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

บทที่ 2
การจัดการโพรเซส

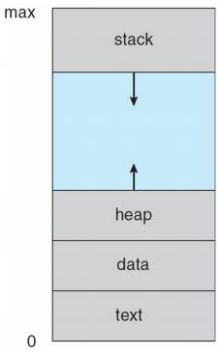
2.1 แนวคิดเรื่องโพรเซส

ระบบคอมพิวเตอร์ในสมัยยุคแรก ๆ มีการทำงานในลักษณะแบบโมโนโปรแกรมมิ่ง (Mono programming) คือ ในขณะเวลาใดเวลาหนึ่งจะมีเพียงโพรเซสเดียวเท่านั้นที่ทำงานได้ ข้อเสียคือ ระบบไม่สามารถใช้ทรัพยากรได้อย่างเต็มประสิทธิภาพ เนื่องจากมีบางช่วงที่ทรัพยากรของระบบว่าง แต่ไม่สามารถใช้ทรัพยากรเหล่านั้นได้ ต่อมามีการพัฒนาระบบคอมพิวเตอร์ที่สามารถทำได้หลายงานพร้อมกัน และสามารถใช้ทรัพยากรร่วมกัน โดยมีการแบ่งงานหรือโปรแกรมออกเป็นส่วนย่อย และทำงานแตกต่างกัน เรียกโปรแกรมส่วนย่อยเหล่านี้ว่า กระบวนการ หรือเรียกทับศัพท์ว่า โพรเซส (Process)

(ยรรยง เต็งอำนวย, 2533: 24) นิยามโดยทั่วไปของคำว่าโพรเซสคือ โปรแกรมที่กำลังดำเนินการอยู่ ถ้ากำหนดให้ชัดเจนกว่านี้ต้องบอกว่าโพรเซสคือ กลุ่มของช่องหน่วยความจำ (Memory cells) ซึ่งเปลี่ยนไปตามกฎเกณฑ์หนึ่ง โดยที่กฎเกณฑ์เหล่านี้เรียกว่า โปรแกรม และอุปกรณ์ที่ตีความโปรแกรมเหล่านี้คือ หน่วยประมวลผล (Processor) โพรเซสหมายถึง โปรแกรมที่อยู่ระหว่างการทำงาน ซึ่งในการทำงานจำเป็นต้องใช้ทรัพยากรต่าง ๆ ของระบบ เช่น ซีพียู อุปกรณ์รับ/ส่งข้อมูล หน่วยความจำหลัก ในระบบคอมพิวเตอร์อาจมีโพรเซสมากกว่าหนึ่งโพรเซส และแต่ละโพรเซสอาจมีสถานะการทำงานที่แตกต่างกัน ระบบปฏิบัติการจึงต้องเข้ามาจัดการเกี่ยวกับโพรเซส และจัดการทรัพยากรของระบบที่มีอยู่อย่างจำกัดให้แก่โพรเซส หน้าที่สำคัญของระบบปฏิบัติการ คือ การจัดการโพรเซส ตั้งแต่การสร้างโพรเซส การจัดลำดับการใช้ซีพียูของแต่ละโพรเซส และการทำลายโพรเซส เป็นต้น

ดังนั้นโปรแกรมจะกลายเป็นโพรเซสเมื่อเรารันไฟล์นั้นขึ้นมาและถูกโหลดขึ้นสู่หน่วยความจำ โดยหน่วยความจำ Stack เป็นพื้นที่ที่ถูกจองไว้ให้โปรแกรมที่ถูกเรียกใช้งานสำหรับเก็บฟังก์ชัน ตัวแปร และพารามิเตอร์ต่าง ๆ ของโปรแกรม ส่วนพื้นที่ Heap เป็นหน่วยความจำที่สงวนไว้ให้โปรแกรมใช้งานสำหรับเป็นหน่วยความจำชั่วคราว ซึ่งขนาดของหน่วยความจำที่ถูกร้องขอในระหว่างที่โปรแกรมกำลังทำงานอยู่จะไม่สามารถทราบได้จนกว่าจะมีการทำงานของโปรแกรม โปรแกรมที่ถูกเรียกใช้งานสามารถขอจองหน่วยความจำที่ว่างอยู่ได้ หลังจากใช้งานเสร็จแล้วก็สามารถของยกเลิกหน่วยความจำส่วนนั้นได้

แม้ว่าเราจะทำงานโปรแกรมเดียวกันในระบบคอมพิวเตอร์ โพรเซสจะถูกแยกออกมาสองโพรเซสและมีการสลับสับเปลี่ยนโพรเซสในการทำงาน และในบางกรณีโพรเซสอาจจะทำงานอยู่บน Virtual Machine เช่น โปรแกรมของภาษาจาวาที่เมื่อคอมไพล์ตัวโค้ดของภาษาจาวาแล้ว จะได้ไฟล์ชื่อโปรแกรมนามสกุล.class และเมื่อรันโปรแกรมนี้ขึ้นมา โพรเซสจะทำงานอยู่บนตัว Java Virtual Machine (ซึ่งจะแปลงคำสั่งต่าง ๆ เป็นภาษาเครื่อง) เพื่อการทำงานของโพรเซสต่อไป



ภาพที่ 2.1 แสดงการจัดเก็บข้อมูลในหน่วยความจำ

2.2 คุณสมบัติของโพรเซส

โพรเซสแต่ละตัวจะถูกกำหนดความสำคัญขึ้น (Priority) ขณะที่โพรเซสถูกสร้างขึ้นความสำคัญนี้อาจเปลี่ยนแปลงได้หรือไม่ได้ขึ้นอยู่กับระบบปฏิบัติการ โพรเซสที่มีความสำคัญมากระบบปฏิบัติการจะให้สิทธิพิเศษมากกว่าโพรเซสที่มีความสำคัญน้อย เช่น ให้เวลาในการครอบครองซีพียู ได้นานกว่าอำนาจหน้าที่ (Authority) เป็นสิ่งที่บ่งบอกว่าโพรเซสนั้น ๆ สามารถทำอะไรได้บ้าง ใช้อุปกรณ์ชิ้นไหนได้บ้าง เป็นต้น ตัวอย่างเช่นโพรเซส A ไม่สามารถเข้าใช้ดิสก์ใด ๆ ทั้งสิ้น แต่สามารถรับข้อมูลจากทุก ๆ โพรเซสในระบบได้คุณสมบัติอื่น ๆ ที่ระบบปฏิบัติการกำหนดให้มี

2.3 สภาวะของโพรเซส (Process State)

ในขณะที่แต่ละโพรเซสกำลังทำงานอยู่ จะมีการเปลี่ยนแปลงสถานะของโพรเซสเกิดขึ้น โดยสถานะของโพรเซส (Process state) ที่สำคัญมี 5 สถานะ คือ

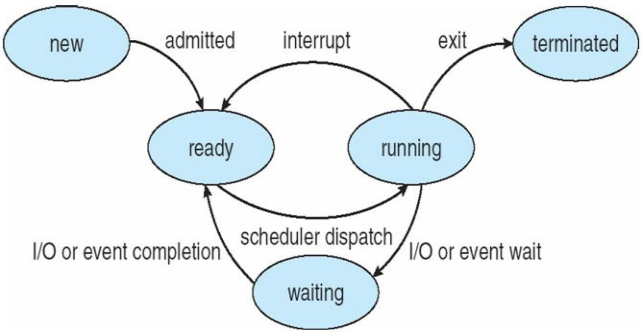
1. สถานะสร้าง (New state) หมายถึง โพรเซสใหม่กำลังถูกสร้างขึ้น

2. สถานะพร้อม (Ready state) หมายถึง สถานะที่โพรเซสพร้อมจะทำงาน หากได้รับการจัดสรรซีพียู

3. สถานะการทำงาน (Running state) หมายถึง สถานะที่โพรเซสได้ครอบครองซีพียู และทำการประมวลผล

4. สถานะรอ (Waiting state) หมายถึง สถานะที่โพรเซสรอเหตุการณ์บางอย่าง

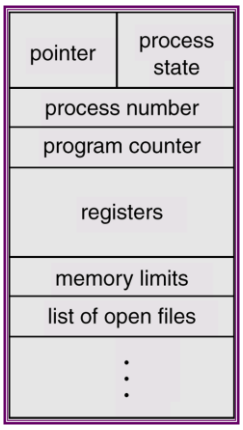
5. สถานะเสร็จสิ้น (Terminate state) หมายถึง สถานะที่โพรเซสเสร็จสิ้นการทำงาน



ภาพที่ 2.2 แผนภาพสถานะของโพรเซส
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.108)

2.4 ตารางข้อมูลการประมวลผล (Process Control Block)

โพรเซส เป็นการทำงานของคำสั่งหรือกลุ่มคำสั่งของโปรแกรม และอาจมีการสลับการทำงานระหว่างโพรเซสในระบบ จึงต้องมีการบันทึกรายละเอียดของโพรเซสแต่ละโพรเซส รวมถึงข้อมูลต่าง ๆ ที่จำเป็นต้องใช้ในการทำงาน จึงใช้โครงสร้างข้อมูลพิเศษที่เรียกว่า Process Control Block (PCB) หรืออาจเรียกว่า Task Control Block โดยมีรายละเอียดของโครงสร้างข้อมูลดังนี้



ภาพที่ 2.3 โครงสร้างข้อมูลของตารางข้อมูลการประมวลผล

1. ตัวชี้ (Pointer) เป็นตัวชี้ไปยังหน่วยความจำหลักที่กันไว้สำหรับโพรเซสนั้น ๆ
2. สถานะของกระบวนการ (Process state) จะเก็บสถานะของโพรเซส ซึ่งจะแสดงสถานะปัจจุบันของโพรเซส
3. ชื่อและหมายเลขประจำตัว (Process number) ของโพรเซสต้องไม่ซ้ำกับโพรเซสอื่น
4. ตัวชี้โปรแกรม (Program counter) เก็บตำแหน่งที่อยู่ของคำสั่งโปรแกรมต่อไปที่จะถูกประมวลผล

5. รีจิสเตอร์ของหน่วยประมวลผล (CPU registers) หน่วยความจำภายในซีพียูทำหน้าที่เก็บสถานะของระบบเมื่อมีอินเทอร์รัพเกิดขึ้น ซึ่งรีจีสเตอร์จะมีข้อมูลที่ถูกเก็บอยู่ในรีจีสเตอร์ภายในโพรเซสเซอร์ ซึ่งถูกนำมาใช้ในขณะที่โพรเซสเซอร์กำลังประมวลผล

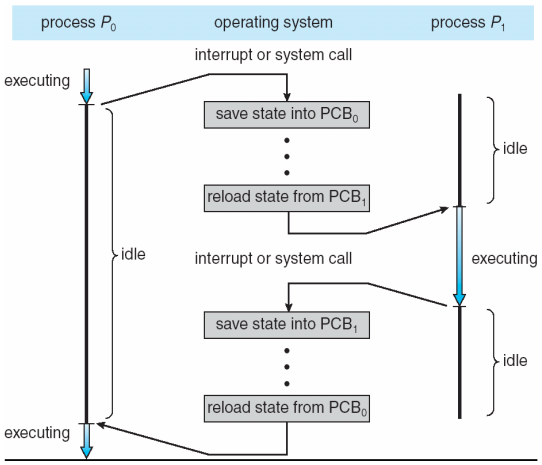
6. ข้อมูลในการจัดตารางทำงานของประมวลผลกลาง (CPU scheduling information) เก็บข้อมูลการจัดตารางเวลาของซีพียู เป็นข้อมูลแสดงลำดับความสำคัญของโพรเซสที่ถูกกำหนดโดยระบบปฏิบัติการ

7. สารสนเทศเกี่ยวกับการจัดการหน่วยความจำ (Memory management information) เก็บข้อมูลการจัดการหน่วยความจำ เป็นข้อมูลเกี่ยวกับหน่วยความจำที่ระบบปฏิบัติการกำหนดไว้ เช่น ขนาดของหน่วยความจำ ค่าของรีจีสเตอร์ ตารางหน่วยความจำเพจเทเบิล (Page table) ตารางเซกเม้นท์ (Segment table)

8. ข้อมูลทางการบัญชี (Accounting information) หมายถึงการเก็บข้อมูลการใช้ทรัพยากร ประกอบด้วย จำนวนเวลาที่ซีพียูใช้ในการทำงาน หมายเลขบัญชี หมายเลขงานหรือหมายเลขโพรเซส และอื่น ๆ

9. ข้อมูลสถานะการรับส่งข้อมูล (I/O status information) เก็บข้อมูลสถานะของ I/O อุปกรณ์ที่โพรเซสปัจจุบันใช้งานอยู่ รายการแฟ้มที่ถูกเปิดอ่าน-เขียนหรือถูกใช้งาน และอื่น ๆ

แต่ละโพรเซสจะมีบล็อกควบคุมเพื่อเก็บสถานะการทำงานของตนเองเรียกว่า PCB โดยโพรเซสแต่ละโพรเซสจะมีหมายเลขเฉพาะเพื่อใช้ในการแสดงตน PCB เป็นพื้นที่หน่วยความจำที่ระบบปฏิบัติการกำหนดไว้เพื่อเก็บข้อมูลที่สำคัญของโพรเซส ดังนั้นเมื่อระบบปฏิบัติการมอบเวลาของซีพียูให้กับโพรเซสอื่นครอบครองเพื่อประมวลผล ถ้าโพรเซสนั้นได้สลับกลับมาครอบครองเวลาของซีพียูใหม่อีกครั้ง โพรเซสจะนำข้อมูลที่อยู่ใน PCB ของตัวมันเองมาใช้เพื่อให้ทราบสถานะการทำงานเดิมของตน



ภาพที่ 2.4 แผนภาพแสดงการสับเปลี่ยนซีพียู จากโพรเซสหนึ่งไปสู่อีกโพรเซสหนึ่ง
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.109)

จากภาพที่ 2.4 สามารถอธิบายการทำงานของโพรเซส P0 และ P1 ที่ต้องการทำงานพร้อมกัน จึงจำเป็นต้องสลับการเข้าใช้ซีพียู สามารถอธิบายลำดับการทำงานได้ดังนี้

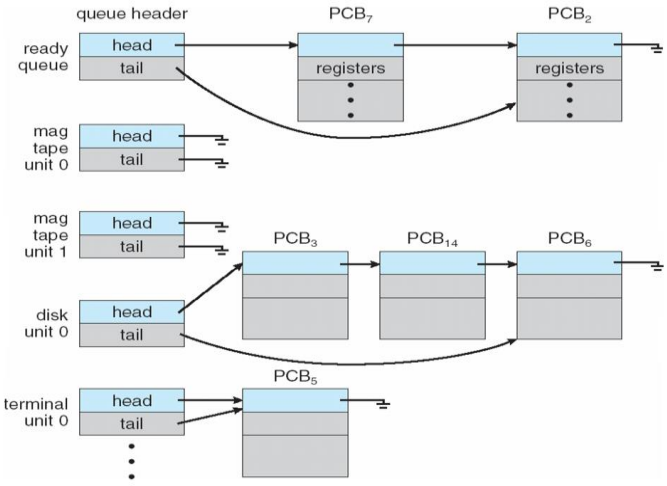
1. เมื่อโพรเซส P0 ทำงานได้ระยะเวลาหนึ่ง ระบบปฏิบัติการจะต้องสลับการทำงานจากโพรเซส P0 ไปยังโพรเซส P1 (เนื่องจากเกิดการขัดจังหวะด้วยสาเหตุใด ๆ)
2. ระบบปฏิบัติการจะทำการเก็บสถานะของโพรเซส P0 ไว้ใน PCB0
3. สลับการทำงานของซีพียูไปทำงานในโพรเซส P1 โดยใช้ข้อมูลของโพรเซส P1 ที่เก็บไว้ใน PCB1
4. เมื่อโพรเซส P1 ทำงานไปแล้วช่วงระยะเวลาหนึ่ง และถูกขัดจังหวะ ระบบปฏิบัติการจะต้องบันทึกข้อมูลของโพรเซส P1 เก็บไว้ใน PCB1 และสลับไปทำงานในโพรเซสอื่นต่อไป

2.5 การจัดตารางของโพรเซส

โดยปกติการทำงานของระบบคอมพิวเตอร์แบบโมโนโปรแกรมมิ่งจะมีเพียง 1 โพรเซสเท่านั้นที่สามารถใช้งานซีพียูได้ ในปัจจุบันการทำงานของระบบคอมพิวเตอร์แบบมัลติโปรแกรมมิ่ง ที่มีหลาย ๆ โพรเซสสามารถทำงานพร้อมกันได้ ถ้ามีซีพียูเพียงตัวเดียว ระบบจำเป็นต้องให้แต่ละโพรเซสสลับกันใช้ซีพียู จุดประสงค์ของการทำงานแบบนี้คือ การพยายามที่จะทำให้หน่วยประมวลผลกลางทำงานอย่างมีประสิทธิผลสูงสุด โดยจัดให้มีโพรเซสเข้าไปทำงานในหน่วยประมวลผลตลอดเวลา แต่สำหรับระบบที่มีหน่วยประมวลผลเดียว จะมีเพียงหนึ่งโพรเซสเท่านั้นที่สามารถเข้าใช้หน่วยประมวลผลกลางได้ และในกรณีที่มีหลายโพรเซสต้องการเข้าใช้หน่วยประมวลผลกลาง งานเหล่านั้นต้องรอจนกว่าหน่วยประมวลผลกลางจะว่างลง แล้วจึงมีการจัดตารางการใช้หน่วยประมวลผลกลางใหม่อีกครั้ง

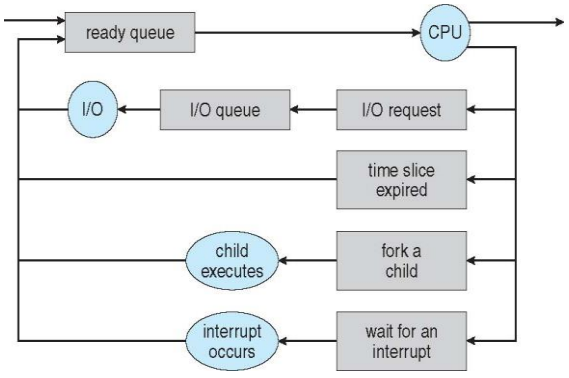
2.5.1 การจัดตารางแถวคอย (Scheduling Queue)

(พิเชษฐ์ ศิริรัตนไพศาลกุล, 2548: 75) เมื่อโพรเซสเข้าสู่ระบบจะถูกจัดให้อยู่ในแถวคอย (Job queue) ในหน่วยเก็บข้อมูลขนาดใหญ่ เมื่อโพรเซสได้เข้าสู่หน่วยความจำหลักแล้ว โพรเซสนั้นจะถูกเก็บในรายการแบบเชื่องโยง (Link list) ที่เรียกว่าแถวพร้อม (Ready queue) ซึ่งโพรเซสภายในแถวพร้อมนี้เป็นโพรเซสที่อยู่ในหน่วยความจำหลัก และพร้อมที่จะทำงาน ทั้งนี้เพียงแค่กำลังรอคอยที่จะเข้าใช้หน่วยประมวลผลกลางเพื่อทำงานต่อไป โดยที่แถวพร้อมจะเก็บตัวชี้ที่ชี้ไปยังตารางข้อมูลของโพรเซสแรกและโพรเซสสุดท้ายในแถวพร้อม และในแต่ละ PCB ก็จะมีตัวชี้ไปยัง PCB ที่อยู่ถัดไปด้วย เมื่อโพรเซสได้เข้าใช้หน่วยประมวลผลกลางและทำงานไปได้ระยะเวลาหนึ่ง โพรเซสนั้นอาจต้องหยุดทำงาน เป็นเพราะงานเสร็จหรือหยุดรอเหตุการณ์บางอย่างให้เกิดขึ้น เช่น รออุปกรณ์รับ-ส่งข้อมูล เนื่องจากมีหลายโพรเซสทำงานอยู่ในระบบ ดังนั้นโพรเซสทั้งหลายอาจต้องเข้าแถวคอยเพื่อที่จะใช้อุปกรณ์รับ-ส่งข้อมูลต่าง ๆ ซึ่งแถวคอยอุปกรณ์ดังกล่าว เรียกว่า “แถวอุปกรณ์” (Device queue) โดยอุปกรณ์แต่ละตัวจะมีแถวคอยเป็นของตัวเอง ดังภาพที่ 2.5



ภาพที่ 2.5 แสดงลำดับงานที่พร้อมทำงานและลำดับงานของอุปกรณ์ประเภทต่าง ๆ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.111)

ในกรณีที่อุปกรณ์ตัวนั้นเป็นอุปกรณ์ที่ใช้เฉพาะงานใดงานหนึ่ง เช่น เครื่องขับเทป แถวคอยของอุปกรณ์ตัวนั้นก็จะมีเพียง 1 โพรเซสเท่านั้น แต่ถ้าเป็นอุปกรณ์ที่สามารถใช้ร่วมกันได้ เช่น จานบันทึก ภายในแถวคอยของอุปกรณ์ตัวนั้นก็อาจมีหลายโพรเซสคอยอยู่ได้ ภาพถัดไป แสดง “แผนภาพของแถวคอย” (Queuing diagram) รูปสี่เหลี่ยมผืนผ้าแทนแถวคอย 2 แบบ คือ แถวพร้อมและแถวอุปกรณ์ วงกลมแทนทรัพยากรซึ่งเป็นเจ้าของแถวคอยนั้น ๆ เส้นลูกศรแสดงทิศทางการไหลของโพรเซสในระบบ



ภาพที่ 2.6 แผนภาพแสดงแถวลำดับที่คอยใช้แทนการจัดลำดับโพรเซส
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.112)

เมื่อมีโพรเซสใดโพรเซสหนึ่งเข้าสู่แถวพร้อม โพรเซสนั้นจะรออยู่ในแถวพร้อม จนกระทั่งถูกเลือกให้เป็นผู้ใช้หน่วยประมวลผลกลาง และหลังจากที่โพรเซสได้ถูกจัดให้เข้าใช้หน่วยประมวลผลกลางแล้ว ขณะที่โพรเซสกำลังทำงานอยู่ อาจมีเหตุการณ์หนึ่งเหตุการณ์ใดเกิดขึ้น ดังนี้คือ

1. โพรเซสร้องขออุปกรณ์รับ-ส่งข้อมูล ดังนั้นจึงถูกจัดให้รอในแถวอุปกรณ์
2. โพรเซสสร้างโพรเซสย่อยขึ้นมา และรอจนโพรเซสย่อยทำงานเสร็จ
3. โพรเซสถูกขัดจังหวะโดยระบบ ทำให้ต้องหยุดการทำงาน และถูกย้ายไปไว้ในแถวพร้อมอีกครั้ง

แม้ใน 2 กรณีแรก เมื่ออุปกรณ์ทำงานเสร็จหรือโพรเซสย่อยทำงานเสร็จ โพรเซสก็สามารถเปลี่ยนจากสถานะรอคอยไปเป็นสถานะพร้อม และถูกย้ายไปไว้ในแถวพร้อมในที่สุด โพรเซสหนึ่ง ๆ จะทำงานสลับกันไปเช่นนี้เรื่อย ๆ จนกว่าจะสิ้นสุดการทำงานและออกไปจากระบบ

2.5.2 การจัดตารางการทำงาน (Schedulers)

การนำโพรเซสไปต่อท้ายคิว หรือนำเอาโพรเซสออกจากคิว จะต้องมีโปรแกรมที่ช่วยกำหนด และจัดลำดับของโพรเซส เรียกว่า กำหนดการ (Scheduler) โดยแบ่งออกได้เป็น 3 ประเภท คือ

1. กำหนดการระยะยาว (Long-term Scheduler)

กำหนดการระยะยาวหรือเรียกอีกชื่อหนึ่งว่า Job Scheduler ทำหน้าที่เลือกโพรเซสที่รออยู่ที่หน่วยความจำสำรอง และต้องการเข้าไปทำงานที่ซีพียู ให้เข้าไปอยู่ในหน่วยความจำหลักที่คิวพร้อมเพื่อรอจนกว่าซีพียูว่าง

2. กำหนดการระยะสั้น (Short-term Scheduler)

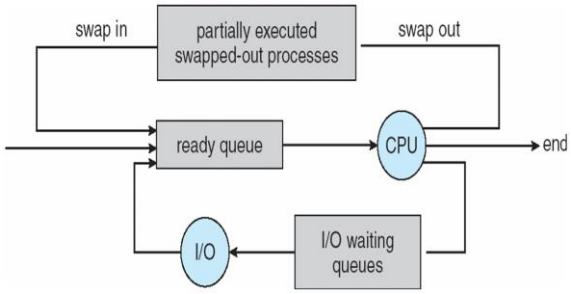
กำหนดการระยะสั้นหรือเรียกอีกชื่อหนึ่งว่า CPU Scheduler ทำหน้าที่เลือกโพรเซสจากคิวพร้อมในหน่วยความจำหลัก และมอบหมายซีพียูให้กับโพรเซสแต่ละโพรเซส เป็นผลให้โพรเซสเปลี่ยนสถานะจากพร้อมเป็นทำงาน

3. กำหนดการระยะกลาง (Medium-term Scheduler)

สามารถถูกนำมาใช้เพิ่มเติม ในกรณีถ้ามีความจำเป็นใช้โพรเซสในระบบมัลติโปรแกรมมิ่ง เพื่อลดโพรเซสที่อยู่ในหน่วยความจำ โดยการย้ายโพรเซสออกจากหน่วยความจำและ Swap ลงดิสก์ และนำโพรเซสนั้นกลับมาใช้หน่วยความจำอีกครั้งเมื่อต้องการให้โพรเซสนั้นทำงานต่อไป

ข้อแตกต่างระหว่างกำหนดการระยะยาวและกำหนดการระยะสั้น คือ ความถี่ของการถูกเรียกใช้งาน กำหนดการระยะสั้นจะถูกเรียกใช้งานมากกว่ากำหนดการระยะยาว เนื่องจากซีพียูทำงานด้วยความเร็วสูง ทำให้โพรเซสถูกทำงานในระยะเวลาสั้นมาก (2-3 มิลลิวินาที) จากนั้นโพรเซสจะถูกขัดจังหวะเพื่อไปทำงานด้านการรับส่งข้อมูล หรืองานอาจจะเสร็จเรียบร้อยทำให้ซีพียูว่าง กำหนดการระยะสั้น จะเลือกโพรเซสต่อไปจากคิวพร้อม และมอบหมายซีพียูให้โพรเซสที่เลือกเข้ามา

โดยปกติกำหนดการระยะสั้นจะทำงานอย่างน้อยทุก ๆ 10 มิลลิวินาที ส่วนกำหนดการระยะยาวมีความถี่ในการทำงานน้อยมาก เนื่องจากการสร้างโพรเซสใหม่จำเป็นต้องใช้เวลาเป็นนาที ในการเลือกโพรเซสของกำหนดการระยะยาวจำเป็นต้องพิจารณาว่าเป็นโพรเซสที่เน้นการรับส่งข้อมูล หรือโพรเซสที่เน้นซีพียู เพื่อให้ระบบมีสัดส่วนการทำงานที่พอดี ถ้าเลือกโพรเซสที่เน้นการรับส่งข้อมูล จะทำให้คิวพร้อมว่างและกำหนดการระยะสั้นถูกเรียกใช้งานน้อยมาก ถ้าเลือกโพรเซสที่เน้นงานด้านซีพียู จะทำให้คิวรอว่างอุปกรณ์รับส่งข้อมูลไม่ถูกใช้งาน



ภาพที่ 2.7 การเพิ่มตัวจัดลำดับระยะกลาง
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.114)

ในปัจจุบันระบบมัลติทาสกิ้ง (Multitasking) หมายถึง ระบบที่สมรรถภาพของเครื่องคอมพิวเตอร์สามารถทำงานสองโปรแกรมขึ้นไปได้ในเวลาเดียวกัน ตอนเริ่มต้นของระบบมัลติทาสกิ้งบนมือถือ จะยอมให้ทำงานเพียงหนึ่งโพรเซสเท่านั้น นอกนั้นจะให้หยุดรอไว้ก่อนโดยตัวอย่างเช่น

1. ในระบบปฏิบัติการไอโอเอส ในส่วนของการดำเนินการโพรเซสส่วนหน้า (Single foreground processes) ยอมให้ทำงานเพียงหนึ่งโพรเซส โดยควบคุมผ่านส่วนที่ติดต่อผู้ใช้งาน ในส่วนของโพรเซสที่ทำงานอยู่เบื้องหลัง (Multiple background processes) จะทำงานอยู่บนหน่วยความจำ โดยไม่แสดงออกมาและมีข้อจำกัดในการทำงาน
2. ระบบปฏิบัติการแอนดรอยด์ โพรเซสที่ทำงานอยู่เบื้องหลังถูกใช้เพื่อปฏิบัติงานด้านต่าง ๆ การบริการยังคงดำเนินการได้ แม้ว่าโพรเซสที่ทำงานเบื้องหลังจะถูกระงับการทำงาน โพรเซสในการบริการไม่มีในส่วนของการติดต่อกับผู้ใช้ จึงใช้หน่วยความจำน้อย

2.5.3 การสลับการทำงานของซีพียู (Context Switch)

การสลับการทำงานของซีพียูหมายถึง การที่ซีพียูสลับไปประมวลผลโพรเซสอื่น โดยจะทำการบันทึกสถานะของโพรเซสปัจจุบันเอาไว้ จากนั้นจะเรียกสถานะของโพรเซสที่ถูกบันทึกไว้กลับขึ้นมาประมวลผลต่ออีกครั้ง เมื่อย้อนกลับมาทำงานเดิมซีพียูจะทำการประมวลผลโพรเซสจนเสร็จ จากนั้นก็จะสลับไปยังโพรเซสอื่นถัดไปที่อยู่ในคิว ระหว่างที่ซีพียูสลับการประมวลผลนั้นซีพียูจะว่างและไม่มีการทำงาน เวลาระหว่างนี้จะสูญเปล่า เวลาที่ใช้ในการสลับจะเปลี่ยนไปตามเครื่อง ซึ่งขึ้นอยู่กับความเร็วของหน่วยความจำ จำนวนรีจีสเตอร์ที่ถูกคัดลอก และคำสั่งพิเศษที่มีอยู่ในระบบ

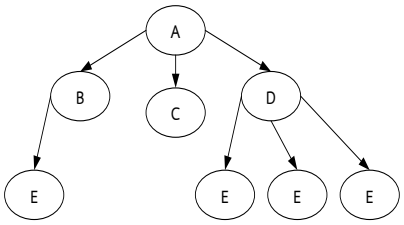
ดังนั้นการสลับการทำงานของซีพียูจึงขึ้นอยู่กับการสนับสนุนของฮาร์ดแวร์ด้วย เช่น ซีพียูบางตัวมีรีจีสเตอร์อยู่หลายชุดก็จะเปลี่ยนตัวชี้ไปยังชุดของรีจีสเตอร์ แต่ถ้าหากมีจำนวนโพรเซสมากกว่า ชุดของรีจีสเตอร์ ระบบจะคัดลอกข้อมูลระหว่างรีจีสเตอร์กับหน่วยความจำ

การสลับการทำงานนี้จะเข้ามาแก้ปัญหาคอขวดของระบบ (Performance bottleneck) ซึ่งการใช้ Context switch กลายเป็นสิ่งที่เพิ่มประสิทธิภาพให้กับระบบเกิดเป็นโครงสร้างใหม่ที่เรียกว่า เธรด (Threads) ที่นำมาใช้เพื่อหลีกเลี่ยงปัญหาคอขวด ในเรื่องของเธรดจะอธิบายในบทที่ 4 ต่อไป

2.6 การดำเนินการของโพรเซส

ในระบบปฏิบัติการมีโพรเซสมากมาย สามารถทำงานได้พร้อมกัน และระบบปฏิบัติการจะมีวิธีการในการสร้างหรือทำลายโพรเซสเป็นปกติอยู่เสมอ กลไกในการสร้างและทำลายโพรเซสสามารถอธิบายลำดับชั้นของโพรเซส (Process hierarchy) ได้ดังนี้

1. เมื่อผู้ใช้ส่งงานให้กับระบบเพื่อทำงาน ระบบปฏิบัติการจะทำการสร้างโพรเซสสำหรับงานนั้นขึ้นมา
2. การทำงานของระบบปฏิบัติการก็ถือว่าเป็นงานของระบบ ดังนั้นจะมีการสร้างโพรเซสขึ้นเหมือนกัน
3. นอกจากนั้นโพรเซสที่ถูกสร้างขึ้นก็สามารถสร้างโพรเซสย่อยได้
4. โพรเซสที่ให้กำเนิด เราเรียกว่า โพรเซสแม่ (Parent process)
5. โพรเซสย่อยที่เกิดขึ้น เราเรียกว่า โพรเซสลูก (Child process)
6. โดยทั่วไป เมื่อโพรเซสแม่จบลง โพรเซสต่าง ๆ ที่อยู่ภายใต้ตัวมันก็จะจบลงตามไปด้วย
7. ระบบปฏิบัติการบางตัวยอมให้โพรเซสแม่จบลง โดยที่โพรเซสลูกไม่ต้องจบลงตามไปด้วย ในกรณีนี้โพรเซสลูกก็จะไม่มีโพรเซสแม่



ภาพที่ 2.8 แสดงลำดับชั้นของโพรเซส

จากตัวอย่างในภาพที่ 2.8 โพรเซส A จะมีโพรเซสลูก 3 โพรเซส คือ B, C และ D ถึงแม้ว่า โพรเซส A เป็นโพรเซสแม่ของโพรเซส B, C และ D แต่โพรเซส A ไม่ได้เป็นผู้ที่สร้างโพรเซส B, C และ D ผู้ที่สร้างโพรเซสทั้งหมดได้แก่ระบบปฏิบัติการ ซึ่งระบบปฏิบัติการจะมีโพรเซสหนึ่งทำหน้าที่สร้างและยุติโพรเซส คือ ตัวจัดคิวระยะยาว

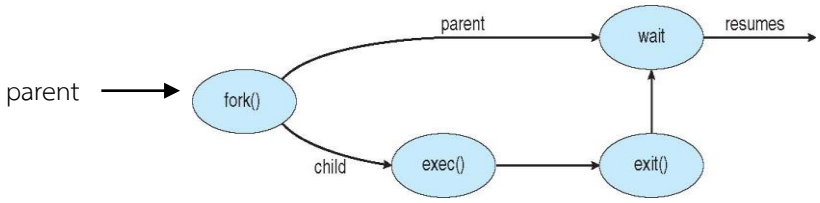
2.6.1 การสร้างโพรเซส (Process Creation)

โพรเซสใด ๆ สามารถสร้างโพรเซสใหม่ได้ด้วยการเรียกใช้คำสั่งระบบ (System command) ของระบบปฏิบัติการ หรือผ่านทาง System call ที่ชื่อ Fork (ในระบบปฏิบัติการ Unix) โพรเซสที่สร้าง โพรเซสอื่นเรียกว่า โพรเซสแม่ เมื่อสร้างโพรเซสลูกแล้วสามารถทำงานต่อไปพร้อมกับโพรเซสลูก หรือหยุดรอจนกว่าโพรเซสลูกจะทำงานเสร็จ โพรเซสที่ถูกสร้างเรียกว่า โพรเซสลูก สามารถร้องขอทรัพยากรจากระบบปฏิบัติการ หรือถูกจำกัดให้ใช้ได้เฉพาะทรัพยากรของโพรเซสแม่เท่านั้น ทั้งนี้โพรเซสแม่จำเป็นต้องทราบหมายเลขโพรเซสลูกทั้งหมด

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
 pid_t pid;

  /* fork a child process */
 pid = fork();

 if (pid < 0) { /* child process */
  fprintf(stderr, “Fork Failed”);
  return 1;
}
  else if (pid ==0) { /*child process */
   execlp (“/bin/ls”, “ls, NULL);
}
else {  /* parent process */
 /* parent will wait for the child to complete */
wait (NULL);
printf(“Child Complete”);
}
   return 0;
}
```

ภาพที่ 2.9 ตัวอย่างโค้ดภาษาซีในการสร้างโพรเซส

โค้ดภาษาซีที่แสดงในภาพที่ 2.9 เป็นตัวอย่างของขั้นตอนการสร้างโพรเซสและการดำเนินการของโพรเซส จะเห็นว่ามีโปสเซสที่แตกต่างกันรันอยู่บนโปรแกรมเดียวกัน (ในบรรทัด pid = fork()) เมื่อค่า pid เป็นศูนย์ ระบบจะมีคำสั่ง System call คือคำสั่ง execlp() ซึ่งในส่วนของโปรแกรมจะเป็นการแสดงรายการไดเรกทอรี (ls เป็นคำสั่งในระบบ Unix) โพรเซสแม่จะรอจนโพรเซสลูกทำงานจนเสร็จ (คำสั่ง wait()) และระบบจะคืนข้อมูลหรือผลลัพธ์ของโพรเซสลูกไปให้โพรเซสแม่ผ่านทาง System call ที่ชื่อ Wait (Unix) และเมื่อโพรเซสลูกทำงานจนเสร็จสิ้น โพรเซสแม่จะกลับมาทำงานต่อไป



ภาพที่ 2.10 การสร้างโพรเซสโดยระบบเรียกฟังก์ชั่น fork( )
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.119)

2.6.2 การสิ้นสุดของโพรเซส (Process Termination)

ในการทำลายโพรเซส โพรเซสจะสิ้นสุดลงเมื่อสิ้นสุดการทำงานในคำสั่งสุดท้าย และแจ้งให้ระบบปฏิบัติการลบมันออกไปโดยใช้ System call ที่ชื่อ Exit (ในระบบปฏิบัติการ Unix) หรือยกเลิก

โพรเซสเมื่อทำงานเสร็จสิ้น เมื่อโพรเซสแม่ทำงานเสร็จสิ้น โพรเซสลูกและโพรเซสที่ถูกสร้างโดยโพรเซสลูกจะต้องสิ้นสุดตามไปด้วย เมื่อโพรเซสแม่สิ้นสุดไปแล้วโพรเซสลูกทั้งหมดก็จะสิ้นสุดไปด้วย สิ่งที่เกิดขึ้นในลักษณะนี้เรียกว่า การสิ้นสุดเป็นขั้น ๆ (Cascading termination) นอกจากนี้โพรเซสใดโพรเซสหนึ่งสามารถเป็นสาเหตุให้โพรเซสอื่นสิ้นสุดการทำงาน โดยผ่านทาง System call ที่เหมาะสมเช่น Abort (ใช้ในระบบปฏิบัติการ Unix) โดยโพรเซสแม่สามารถยกเลิกโพรเซสลูกได้ ในกรณีที่โพรเซสลูกใช้ทรัพยากรของโพรเซสแม่จนหมด ทำให้ทรัพยากรไม่พอใช้ โพรเซสแม่ไม่ต้องการใช้โพรเซสลูกอีกต่อไป โพรเซสแม่ทำงานเสร็จสิ้นแล้ว และระบบปฏิบัติการไม่ต้องการให้โพรเซสลูกทำงานต่อไป

2.7 การสื่อสารระหว่างโพรเซส

โพรเซสที่ทำงานในระบบปฏิบัติการอาจจะเป็นโพรเซสอิสระ (Independent processes) หรือโพรเซสที่ต้องทำงานร่วมกัน (Cooperation processes) โพรเซสอิสระ คือ โพรเซสที่ไม่มีผลกระทบต่อโพรเซสอื่นในระบบเช่น โพรเซสซึ่งไม่แบ่งข้อมูล (ชั่วคราวหรือถาวร) ให้กับโพรเซสอื่น โพรเซสที่ต้องทำงานร่วมกัน คือ โพรเซสที่มีผลกระทบต่อโพรเซสอื่นในระบบ เช่น โพรเซสที่ต้องแบ่งข้อมูลให้กับโพรเซสอื่นที่เป็นโพรเซสร่วม มีเหตุผลต่าง ๆ มากมายที่ทำให้ต้องจัดเตรียมสิ่งแวดล้อมให้กับโพรเซสที่ต้องทำงานร่วมกันดังนี้

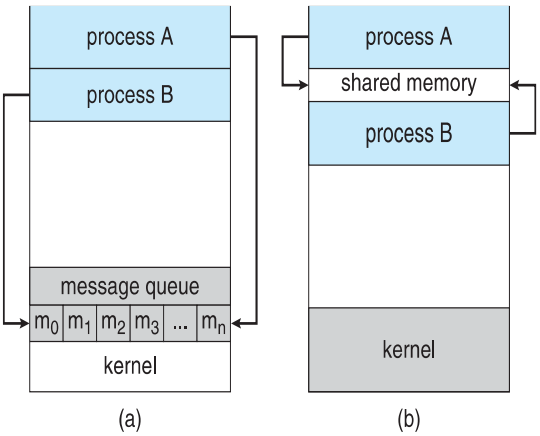
1. การร่วมกันใช้ข้อมูลข่าวสาร (Information Sharing) เมื่อผู้ใช้หลายคนสนใจข้อมูลข่าวสารชิ้นเดียวกัน (ตัวอย่างเช่น แฟ้มข้อมูลที่ถูกแชร์) จำเป็นต้องจัดเตรียมสิ่งแวดล้อม โดยอนุญาตให้เข้าถึงทรัพยากรเหล่านี้ร่วมกันได้

2. การคำนวณรวดเร็วขึ้น (Computation Speedup) ถ้าต้องการให้งานปกติสามารถทำงานได้เร็วขึ้น จำเป็นจะต้องแตกงานเหล่านั้นเป็นส่วนย่อย ๆ แล้วให้แต่ละส่วนทำงานขนานกันไป แต่ความเร็วในการคำนวณจะสูงขึ้นได้ก็ต่อเมื่อ ระบบมีอุปกรณ์ที่ใช้คำนวณหลาย ๆ ตัว เช่น มีซีพียูหลาย ๆ ตัว หรือมีหน่วยคำนวณหลาย ๆ ตัว

3. ระบบย่อย (Modularity) บางทีระบบอาจต้องการที่จะสร้างระบบให้อยู่ในรูปแบบของระบบย่อยหรือโมดูล โดยอาจแบ่งหน้าที่งานต่าง ๆ ของระบบไปเป็นหน้าที่ละหนึ่งโพรเซส

4. ความสะดวกสบาย (Convenience) ผู้ใช้แต่ละคนอาจจะมีงานหลาย ๆ งานที่ต้องทำงานในเวลาเดียวกัน เช่น ต้องการแก้ไขข้อมูล พิมพ์ข้อมูล และแปลภาษาไปพร้อม ๆ กัน ซึ่งสามารถกระทำได้ในเวลาเดียวกัน

กลไกที่สนับสนุนให้โพรเซสสามารถประสานกันได้ คือ การสื่อสารระหว่างโพรเซส (Inter Process Communication : IPC) และการประสานโพรเซส (Synchronize process) การทำงานของโพรเซสที่มีการประสานกับโพรเซสอื่น จำเป็นต้องใช้บัฟเฟอร์ โดยระบบปฏิบัติการจะต้องทำการแชร์หน่วยความจำไว้ใช้งาน และจะต้องมีกลไกที่สนับสนุนให้สามารถประสานได้ กลไกที่ว่าคือการติดต่อระหว่างโพรเซส ดังนั้นวิธีการสื่อสารระหว่างโพรเซสที่ยอมโพรเซสเหล่านั้นให้มีการแลกเปลี่ยนข้อมูลหรือสารสนเทศ พื้นฐานของ IPC มีอยู่ 2 แบบคือ แบบ Share Memory และแบบ Message Passing



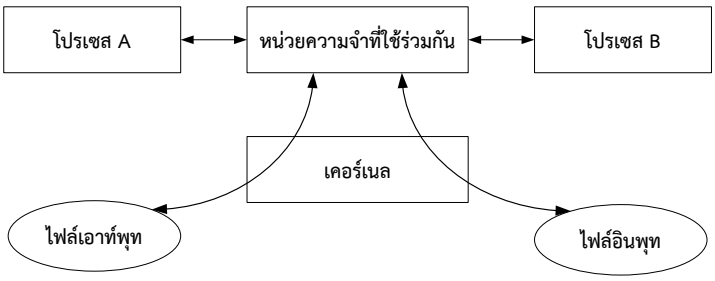
ภาพที่ 2.11 แสดงรูปแบบการสื่อสาร (a) Message passing. (b) Shared memory
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.124)

2.7.1 Shared-Memory Systems

ระบบ Share memory เป็นเทคนิคการสื่อสารข้อมูลในกระบวนการที่มีหลาย ๆ โพรเซสเข้ามาใช้หน่วยความจำที่เดียวกัน โดยจะสามารถทำได้ในระบบปฏิบัติการแบบ Multitasking การใช้งานหน่วยความจำร่วมกันเป็นปัจจัยหนึ่งที่จะทำให้โพรเซส 2 โพรเซส หรือมากกว่าใช้ข้อมูลในหน่วยความจำร่วมกันได้โดยตรง ไม่ผ่านเคอร์เนล (ออบเจ็ก IPC ที่ผ่านมาเช่น PIPE ,FIFO จะทำงานผ่านเคอร์เนล) โดยโพรเซสเหล่านั้นจะต้องจัดการการทำงานร่วมกันด้วยตัวเอง

หลักการทำงานของระบบ Shared Memory หน่วยความจำที่ถูกกำหนดให้ใช้งานร่วมกัน ปกติแล้วก็คือ ตำแหน่งแอดเดรสที่ว่างของหน่วยความจำ เพื่อกำหนดเป็นพื้นที่ใช้หน่วยความจำร่วมกัน โพรเซสสามารถเข้ามาใช้ข้อมูลในหน่วยความจำ เพื่อการสื่อสารข้อมูลหรือแลกเปลี่ยนข้อมูลเหล่านั้น เสมือนกับว่าพื้นที่ของโพรเซสเอง การเปลี่ยนแปลงใด ๆ ที่เกิดขึ้นไม่ว่ามาจากโพรเซสใด จะมีผลต่อโพรเซสทั้งหมดที่กำลังใช้หน่วยความจำในขณะนั้น

หน่วยความจำที่ถูกใช้งานร่วมกันไม่มีหน้าที่กำหนดจังหวะการทำงานร่วมกันของโพรเซสต่าง ๆ เช่น การอ่านข้อมูลขณะที่โพรเซสอื่นกำลังเขียนข้อมูลอยู่ รวมทั้งไม่มีฟังก์ชันอื่นใดที่จะมาทำหน้าที่นี้ ดังนั้นโปรแกรมเมอร์จะต้องการทำงานนี้ด้วยตนเอง



ภาพที่ 2.12 แสดงการส่งผ่านข้อมูลระหว่างไคลเอนท์และเซิร์ฟเวอร์เมื่อใช้งานหน่วยความจำที่ใช้ร่วมกัน

ดังนั้นจึงต้องมีการประสานงานระหว่างโพรเซสเพื่อป้องกันไม่ให้มีโพรเซสใดใส่ข้อมูลลงในบัฟเฟอร์ที่เต็มแล้ว หรือโพรเซสใดพยายามดึงข้อมูลจากบัฟเฟอร์ทั้ง ๆ ที่ยังไม่มีข้อมูลอยู่ ตัวอย่างของแนวทาง การทำงานร่วมกันของโพรเซสนี้ สามารถพิจารณาจากกรณีปัญหาผู้ผลิตและผู้บริโภค (Producer Consumer Problem) ซึ่งเป็นหลักการพื้นฐานของการใช้หน่วยความจำร่วมกันระหว่างโพรเซส เช่น มีการแชร์บัฟเฟอร์เป็นส่วนความจำที่คั่นระหว่างงานสองงาน ซึ่งต้องส่งผ่านข้อมูลระหว่างกัน แต่มีความเร็วในการรับส่งข้อมูลไม่เท่ากัน งานทั้งสองจะต้องทำงานร่วมกันในลักษณะผู้ผลิตและผู้บริโภค (Producer-consumer) การทำงานร่วมกันคือ ผู้ผลิตจะต้องสามารถผลิตข้อมูลเข้าสู่บัฟเฟอร์ได้ และผู้บริโภคก็สามารถบริโภคข้อมูลต่าง ๆ เหล่านั้นได้ ดังนั้นผู้ผลิตจะมีหน้าที่ใส่ข้อมูลลงในส่วนของบัฟเฟอร์และผู้บริโภคจะนำข้อมูลออกจากบัฟเฟอร์ โดยผู้ผลิตจะไม่มีการใส่ข้อมูลทับข้อมูลเก่า และผู้บริโภคจะไม่พยายามนำข้อมูลที่ยังผลิตไม่เสร็จออกไปใช้ บัฟเฟอร์นี้จะว่างก็ต่อเมื่อไม่มีการผลิตและผู้บริโภคได้บริโภคข้อมูลจนหมดแล้ว ประเภทของบัฟเฟอร์ที่สามารถถูกนำมาใช้มี 2 ประเภทคือ

1. บัฟเฟอร์ข้อมูลมีขนาดไม่จำกัด (Unbounded-buffer) โพรเซสที่ต้องการบริโภคข้อมูลอาจต้องคอยจากผู้ผลิต แต่โพรเซสผู้ผลิตข้อมูลสามารถผลิตข้อมูลได้ตลอดเวลา (ไม่มีวันเต็ม)
2. บัฟเฟอร์มีขนาดจำกัด (Bounded-buffer) โพรเซสผู้ผลิตอาจต้องรอถ้าบัฟเฟอร์เต็ม และโพรเซสที่ต้องการบริโภคข้อมูลอาจต้องรอถ้าบัฟเฟอร์ว่าง

เพื่อให้เกิดความเข้าใจมีตัวอย่างของโค้ดโปรแกรมภาษาซี ในการใช้ลักษณะของบัฟเฟอร์ที่มีขนาดจำกัด ในรูปแบบ Circular buffer หรือเรียกเทคนิคนี้ว่า Circular queue โดยประกาศตัวแปรที่อยู่ในหน่วยความจำเพื่อใช้งานร่วมกัน (ตัวแปรแบบ Structure) ทั้งผู้ผลิตและผู้บริโภคดังนี้

```
#define BUFFER_SIZE 10
typedef struct {
        . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

ภาพที่ 2.13 แสดงโค้ดโปรแกรมการประกาศโครงสร้างข้อมูล

การแชร์บัฟเฟอร์ คือ ส่วนความจำที่คั่นระหว่างงานสองงาน ซึ่งต้องส่งผ่านข้อมูลระหว่างกัน ถูกประกาศเป็นอาร์เรย์แบบวงกลม และตัวแปรที่กำหนดเป็นตัวชี้ตำแหน่งของอาร์เรย์คือ in และ out โดยบัฟเฟอร์ว่างก็ต่อเมื่อค่า in ชี้ในตำแหน่งอาร์เรย์เดียวกันกับ out และ บัฟเฟอร์เต็มเมื่อตรวจสอบพบว่า ((in + 1) % BUFFER SIZE) == out หมายความว่าเมื่อข้อมูลในช่องถัดไปของค่าที่ชี้ตำแหน่งข้อมูลเข้า in มีค่าเท่ากับค่าชี้ตำแหน่งออก out

```
while (true) {
        /* produce an item in next produced */
        while (((in + 1) % BUFFER_SIZE) == out)
                ; /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
}
```

ภาพที่ 2.14 แสดงโค้ดโปรแกรมของโพรเซสผู้ผลิต

```
item next_consumed;
while (true) {
        while (in == out)
                ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        /* consume the item in next consumed */
}
```

ภาพที่ 2.15 แสดงโค้ดโปรแกรมของโพรเซสผู้บริโภค

ตัวแปรเป็นโลคอล next_produced เป็นการสร้างข้อมูลเพื่อนำไปใส่ให้กับอาร์เรย์ และ next_consumed เป็นการนำข้อมูลจากอาร์เรย์ไปใช้

2.7.2 Message-Passing Systems

ฟังก์ชั่นระบบข่าวสารจะอนุญาตให้โพรเซสสามารถสื่อสารกับกระบวนการอื่นได้ โดยไม่จำเป็นต้องมีการใช้ทรัพยากรหรือข้อมูลร่วมกัน โดยไอพีซีจะจัดเตรียมการดำเนินงานอย่างน้อย 2 ขั้นตอน คือ Send / Receive Message สำหรับการสื่อสาร โดยอาศัยช่องทางที่เรียกว่า การเชื่อมโยงการสื่อสาร Communication Link กระบวนการที่ทำการส่งจะต้องสร้างการเชื่อมโยงกับโพรเซสที่ทำหน้าที่รับเสียก่อน ส่วนการเชื่อมโยงการสื่อสารนั้นสามารถทำได้หลายวิธี โดยอาจเป็นแบบกายภาพ (Physical) เช่น ใช้หน่วยความจำร่วมกัน ระบบฮาร์ดแวร์บัส ระบบเครือข่ายร่วมกัน หรือแบบตรรกะ Logical ก็ได้

ถ้าใช้แบบตรรกะก็สามารถทำได้หลายวิธี โดยอาศัยการเชื่อมโยงและการดำเนินการแบบ Send / Receive เข้ามาช่วย ได้แก่

1. การสื่อสารทางตรงและทางอ้อม (Direct / Indirect communication)
2. การสื่อสารแบบสมมาตรและแบบอสมมาตร (Symmetric / Asymmetric)
3. การปรับอัตรา (Buffering) แบบอัตโนมัติหรือแบบชัดเจนข่าวสารแบบขนาดคงที่หรือแปรผัน

ในการใช้งานการเชื่อมโยง (Link) เส้นทางการสื่อสารระหว่างโพรเซสนั้นจำเป็นต้องพิจารณาประเด็นคำถามดังต่อไปนี้

- จะทำการติดตั้งสายการเชื่อมโยงได้อย่างไร
- สายการเชื่อมโยงหนึ่งเส้น สามารถใช้สื่อสารได้มากกว่าหนึ่งโพรเซสหรือไม่
- ต้องมีจำนวนสายการเชื่อมโยงกระบวนการกี่เส้น
- ความจุของสายควรมีขนาดเท่าใด
- ขนาดของข่าวสารที่ใช้สื่อสารกันจะมีรูปแบบคงที่หรือแปรผัน
- สายการเชื่อมโยงจะเป็นแบบเดี่ยว (Unidirectional) หรือแบบคู่ (Bi-directional)

การตั้งชื่อ (Naming) กระบวนการที่ต้องการสื่อสารกับโพรเซสอื่นจะต้องมีวิธีในการอ้างอิงถึงซึ่งสามารถสื่อสารกันได้ทั้งทางตรงและทางอ้อม โดยการสื่อสารทางตรง (Direct communication) โพรเซสที่ต้องการสื่อจะต้องใช้ชื่อที่ชัดเจนทั้งฝั่งรับ และฝั่งส่งในรูปแบบดังนี้

- send (P, message) ส่งข่าวสารไปยังโพรเซส P
- receive (Q, message) รับข่าวสารจากโพรเซส Q

การเชื่อมโยงการสื่อสารจะต้องมีคุณสมบัติ ดังนี้

- ทั้งฝั่งรับและฝั่งส่งจะต้องติดตั้งการเชื่อมโยงระหว่างกันโดยอัตโนมัติ
- การเชื่อมโยงจะทำเฉพาะคู่ของฝั่งรับและฝั่งส่งเท่านั้น
- ฝั่งรับและฝั่งส่งจะมีการเชื่อมโยงเพียงเส้นเดียวเท่านั้น
- สายการเชื่อมโยงสามารถใช้ได้ทั้งแบบทางเดี่ยวและทางคู่ แต่โดยทั่วไปใช้แบบทางคู่

ในการกำหนดตำแหน่งที่อยู่แบบสมมาตร ทั้งผู้รับและผู้ส่งจะต้องมีชื่อสำหรับการสื่อสารกัน ในขณะที่ตำแหน่งที่อยู่แบบอสมมาตรนั้น ผู้ส่งระบุเพียงชื่อของผู้รับเท่านั้น แต่ผู้รับไม่จำเป็นต้องระบุชื่อผู้ส่งตามรูปแบบข้างล่าง ดังนี้

- send (P, message) ส่งข่าวสารไปยังกระบวนการ P
- receive (id, message) รับข่าวสารที่ถูกส่งมาจากกระบวนการใด ๆ ก็ได้ โดยตัวแปร id จะแทนกลุ่มของชื่อกระบวนการที่สื่อสารกัน

ข้อด้อยของวิธีแบบสมมาตรและแบบอสมมาตรก็คือ มีข้อจำกัดเกี่ยวกับชื่อมากเกินไป ถ้ามีการเปลี่ยนชื่อของกระบวนการจะต้องแจ้งไปยังสมาชิกทั้งหมด เวลาจะแก้ไขชื่อใหม่จะต้องหาชื่อเดิมที่อ้างอิงเสียก่อน ซึ่งถือว่าไม่ยืดหยุ่น

2.7.3 การติดต่อทางตรง (Direct Communication)

การติดต่อแบบนี้แต่ละโพรเซสที่ต้องการติดต่อกันจะต้องกำหนดชื่อเฉพาะที่ใช้ในการติดต่อทั้งผู้รับและผู้ส่ง ยกตัวอย่างเช่น ถ้าต้องการส่งเมสเสจจาก A ไป B จะมีการกำหนดรูปแบบคือ

- send (B, message) จะเป็นการส่งเมสเสจไปยังโพรเซส B
- receive (A, message) จะเป็นการรับเมสเสจจากโพรเซส A

ลิงค์แบบนี้จะมีคุณสมบัติดังนี้

1. การสร้างลิงค์จะเป็นแบบอัตโนมัติระหว่างคู่ของโพรเซสที่ต้องการติดต่อ (ในที่นี้คือ A กับ B) โพรเซสจะทราบหมายเลขโพรเซสที่จะติดต่อด้วย
2. ลิงค์หนึ่ง ๆ จะมีความสัมพันธ์เฉพาะโพรเซสสองโพรเซสเท่านั้น
3. ระหว่างโพรเซสแต่ละคู่นั้นจะมีเพียงลิงค์เดียวเท่านั้น
4. ลิงค์นี้เป็นได้ทั้งทิศทางเดียว และสองทิศทาง แต่ปกติจะเป็นแบบสองทิศทาง

วิธีการรับ-ส่งเมสเสจแบบนี้ ระบบปฏิบัติการจะไม่ดำเนินการติดต่อให้กับโพรเซสทั้งสอง โดบทั้งสองโพรเซสจะต้องจัดการเอง ตัวอย่างการส่งเมสเสจจากโพรเซส A ไปโพรเซส B หลังจากสร้างลิงค์เรียบร้อยแล้ว ทั้งโพรเซส A และโพรเซส B จะจองหน่วยความจำที่มีการกำหนดให้สามารถใช้ร่วมกัน (โพรเซสทั้งสองจะทราบว่าหน่วยความจำนั้นอยู่ตำแหน่งใด) โพรเซส A จะนำข้อมูลไปวางบนตำแหน่งหน่วยความจำที่จองไว้ โพรเซส B จะต้องคอยตรวจสอบว่าโพรเซส A วางข้อมูลหรือยัง ถ้ายังไม่มีการวางก็จะยังไม่ดึงข้อมูล รอจนกระทั่งโพรเซส A วางเรียบร้อยแล้วจึงจะดึงข้อมูลนั้นไปใช้งาน ส่วนโพรเซส A ก็จะต้องตรวจสอบเช่นกันว่าข้อมูลที่ตนเองวางนั้นโพรเซส B ดึงไปใช้งานหรือยัง ถ้ายังไม่ดึง โพรเซส A จะต้องรอก่อน ยังไม่มีการวางข้อมูลใหม่ เพื่อป้องกันการวางข้อมูลใหม่ทับข้อมูลเดิมซึ่งจะมีผลให้การทำงานไม่สมบูรณ์ (ข้อมูลหายไป) รอจนกระทั่งการดึงข้อมูลเรียบร้อยจึงวางข้อมูลใหม่ได้ ทำเช่นนี้เรื่อยไปจนกว่าข้อมูลจะหมด กลไกที่ใช้ในการตรวจสอบเวลาที่เหมาะสมในการรับ-ส่งข้อมูลนี้เรียกว่า Process synchronization

2.7.4 การสื่อสารทางอ้อม (Indirect Communication)

ด้วยการสื่อสารทางอ้อมข่าวสารจะถูกส่งและรับผ่านทางตู้ไปรษณีย์ หรือเรียกทับศัพท์ว่าเมลล์บ็อกซ์ (Mail box) หรือพอร์ต (Port) โดยเมลล์บ็อกซ์จะมีเลขที่ซึ่งไม่ซ้ำกันกำกับไว้ ทุกโพรเซสสามารถติดต่อสื่อสารถึงกันโดยอาศัยเมลล์บ็อกซ์ดังกล่าวนี้ ในบางกรณีอาจมีมากกว่าหนึ่งโพรเซสที่สามารถใช้เมลล์บ็อกซ์ร่วมกันได้ การปฏิบัติงานเกี่ยวกับเมลล์บ็อกซ์ที่สามารถทำได้มีดังนี้

1. การสร้างเมลล์บ็อกซ์ใหม่
2. การส่งข่าวสารผ่านเมลล์บ็อกซ์
3. การลบเมลล์บ็อกซ์

การส่ง / รับ ที่สามารถทำได้มีดังนี้

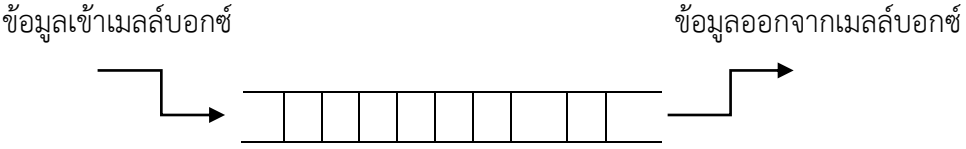
- send (A, message) ส่งข่าวสารไปยังเมลล์บ็อกซ์ A
- receive (A, message) รับข่าวสารจากเมลล์บ็อกซ์ A

การเชื่อมโยงการสื่อสารด้วยวิธีดังกล่าว จะต้องมีการสร้างความเชื่อมโยงไว้ระหว่างกันทั้งสองโพรเซสที่ใช้เมลล์บ็อกซ์ร่วมกัน สายเชื่อมโยงหนึ่งเส้นสามารถรองรับการเชื่อมโยงของโพรเซสได้มากกว่าสองโพรเซส โพรเซสแต่ละคู่สามารถใช้สายเชื่อมโยงเพื่อติดต่อกันได้หลายเส้นทาง สายการเชื่อมโยงอาจใช้ได้ทั้งแบบทางเดี่ยวและทางคู่

แต่เนื่องจากเมลล์บ็อกซ์มีการใช้งานร่วมกันมากกว่าหนึ่งโพรเซส จึงอาจเกิดปัญหาในการสื่อสารขึ้นมาได้ สมมติว่ากระบวนการ P1, P2 และ P3 ต่างใช้ตู้ไปรษณีย์ A ร่วมกัน เมื่อ P1 ส่งข่าวสารไปยังตู้ไปรษณีย์ A | P2 หรือ P3 จะเป็นผู้ได้รับข่าวสารจาก P1 เพื่อแก้ปัญหาดังกล่าว ระบบจะต้องมีข้อกำหนดต่าง ๆ ดังนี้

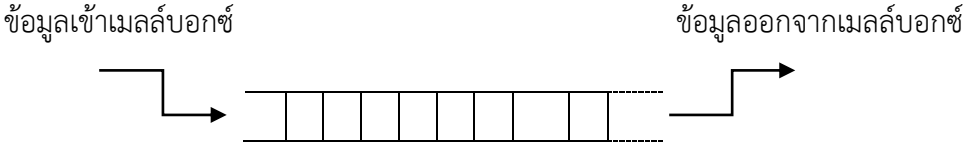
1. อนุญาตให้สายการเชื่อมโยงหนึ่งเส้นรองรับกระบวนการได้มากที่สุดเพียงสองโพรเซสเท่านั้น
2. อนุญาตให้มีเพียงกระบวนการเดียวที่สามารถรับข่าวสารจากเมลล์บ็อกซ์ ณ ขณะใดขณะหนึ่ง ให้ระบบเป็นผู้ตัดสินใจชี้ขาดว่าจะเลือกให้กระบวนการใดเป็นผู้รับข่าวสารนั้น และแจ้งว่าใครเป็นผู้รับไปให้ผู้ส่งทราบ
3. อนุญาตให้ระบบเลือกว่าโพรเซสใดที่จะเข้ารับเมสเสจ (ในที่นี้อาจจะเป็นโพรเซส 2 หรือโพรเซส 3 ก็ได้ แต่ไม่ใช้ครั้งละสองโพรเซสพร้อมกัน) โดยระบบจะกำหนดผู้รับไปยังผู้ส่งก่อนการส่ง

โครงสร้างของเมลล์บ็อกซ์แบบคิว เป็นโครงสร้างที่ดึงข้อมูลออกจากเมลล์บ็อกซ์ตามลำดับก่อน-หลังของข้อมูลที่ส่งเข้ามา นั่นคือข้อมูลใดส่งเข้ามาในเมลล์บ็อกซ์ก่อนก็จะถูกดึงออกไปก่อน ส่วนข้อมูลใดส่งเข้ามาภายหลังก็จะถูกดึงออกไปภายหลัง อาจเรียกการทำงานแบบนี้ว่า FIFO (First In First Out) ลักษณะโครงสร้างเมลล์บ็อกซ์แบบคิวเป็นดังภาพที่ 2.16



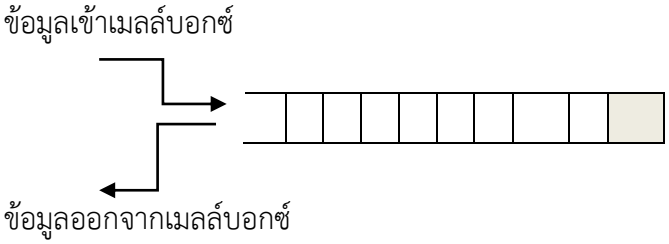
ภาพที่ 2.16 เมลล์บ็อกซ์แบบคิว

เมลล์บ็อกซ์แบบไปป์ โครงสร้างของเมลล์บ็อกซ์แบบนี้เป็นโครงสร้างที่คล้ายกับโครงสร้างแบบคิว คือ การดึงข้อมูลจะเป็นในลักษณะที่ข้อมูลใดส่งเข้ามาก่อนก็จะถูกดึงออกไปก่อน ข้อมูลใดส่งเข้ามาภายหลังจะถูกดึงออกไปใช้งานภายหลัง ข้อแตกต่างระหว่างเมลล์บ็อกซ์แบบคิวกับเมลล์บ็อกซ์แบบไปป์ คือ เมลล์บ็อกซ์แบบคิวจะมีขนาดบ็อกซ์คงที่ ถ้าใส่ข้อมูลมากเกินไปเมลล์บ็อกซ์จะเต็ม แต่ถ้าเป็นเมลล์บ็อกซ์แบบไปป์ขนาดของบ็อกซ์จะยืดหยุ่นได้ ทำให้สามารถใส่ข้อมูลในเมลล์บ็อกซ์ได้มากเท่าที่ต้องการ เมลล์บ็อกซ์จะขยายตัวโดยอัตโนมัติ ลักษณะโครงสร้างเมลล์บ็อกซ์แบบไปป์เป็นดังภาพที่ 2.17



ภาพที่ 2.17 เมลล์บ็อกซ์แบบไปป์

เมลล์บ็อกซ์แบบสแต็ก โครงสร้างของเมลล์บ็อกซ์แบบนี้เป็นโครงสร้างตรงข้ามกับเมลล์บ็อกซ์แบบคิวในการดึงข้อมูล นั่นก็คือข้อมูลใดส่งเข้าเมลล์บ็อกซ์ก่อนจะถูกดึงออกไปใช้งานภายหลัง โดยจะนำข้อมูลที่ส่งเข้ามาภายหลังออกไปใช้ก่อน อาจเรียกการทำงานแบบนี้ว่า LIFO (Last In First Out) ลักษณะโครงสร้างเมลล์บ็อกซ์แบบสแต็กเป็นภาพที่ 2.18



ภาพที่ 2.18 เมลล์บ็อกซ์แบบสแตก์

2.8 การปรับอัตรา

การจัดบัฟเฟอร์ในการสร้างลิงค์ นอกจากจะเป็นการกำหนดเส้นทางข้อมูลแล้ว ลิงค์ยังมีความจุที่เป็นตัวเลขแสดงจำนวนเมสเสจที่สามารถเก็บไว้ชั่วคราวได้ คุณสมบัตินี้อาจจะมองว่าเป็นคิวของเมสเสจที่ผูกติดลิงค์ก็ได้ การสื่อสารระหว่างโพรเซสนั้นอาจเป็นแบบทางตรงหรือทางอ้อมก็ได้ ข่าวสารจะถูกแลกเปลี่ยนโดยกระบวนการสื่อสาร ซึ่งอยู่ในกองซ้อนชั่วคราว (Temporary queue) โดยพื้นฐานแล้ว เราจะมีวิธีใช้งานกองซ้อนชั่วคราวได้ วิธีใดวิธีหนึ่งสามวิธีดังนี้

ความจุค่าศูนย์ (Zero capacity) ขนาดสูงสุดของกองซ้อนมีค่าเป็นศูนย์ หมายความว่า จะมีข่าวสารอยู่ในกองซ้อนได้เพียงชุดเดียวเท่านั้น ในกรณีนี้ผู้ส่งจะต้องบล็อกรอจนกระทั่งผู้รับได้รับข่าวสารเรียบร้อยแล้วจึงจะทำอย่างอื่นได้

ความจุค่าจำกัด (Bounded capacity) ขนาดความจุของกองซ้อนมีค่าจำกัดเท่ากับ n ดังนั้นจึงรองรับข่าวสารได้มากถึง n จำนวนเท่าความจุถ้ามีข่าวสารใหม่เข้ามาและกองซ้อนยังไม่เต็ม ก็จะเก็บไว้ในกองซ้อน ฝั่งที่ส่งก็ไม่จำเป็นต้องหยุดรอ แต่ถ้ากองซ้อนหรือสายการเชื่อมโยงเต็ม ที่ส่งจำเป็นต้องหยุดรอจนกระทั่งมีพื้นที่ว่างในกองซ้อนจึงจะสามารถวางข่าวสารได้

ความจุค่าไม่จำกัด (Unbounded capacity) ขนาดของกองซ้อนมีค่าเป็นอนันต์ ดังนั้นจึงสามารถรองรับข่าวสารได้ทั้งหมดโดยผู้ส่งไม่จำเป็นต้องหยุดรอ

ในกรณีของความจุค่าศูนย์ บางทีหมายถึง ระบบข่าวสารที่ไม่มีการปรับอัตราส่วนอีก 2 กรณีหลังเรียกว่า การปรับอัตราแบบอัตโนมัติ อาจกล่าวได้ว่าความจุศูนย์เป็นระบบที่ไม่มีบัฟเฟอร์ก็ได้ ส่วนกรณีอื่นมีการจัดบัฟเฟอร์ให้อัตโนมัติ สำหรับในกรณีที่ไม่ใช่ศูนย์ โพรเซสจะไม่รู้เลยว่าเมสเสจถึงปลายทางหรือไม่หลังจากการส่งเสร็จสิ้นไปแล้ว ถ้าข้อมูลนี้มีความสำคัญในการคำนวณ ผู้ส่งจะต้องติดต่อกับผู้รับเพื่อหาเมสเสจที่ส่งมาครั้งหลังสุด ตัวอย่างเช่น ถ้าโพรเซส P ส่งเมสเสจไปยังโพรเซส Q และสามารถทำงานได้ต่อไปเฉพาะหลังจากที่เมสเสจได้รับไปแล้ว โพรเซส P จะมีขั้นตอนดังนี้

```
send (Q, message);
receive (Q, message);

โพรเซส Q จะเอ็กซิคิวต์คำสั่ง
receive (P, message);
send (P, “acknowledgment”);
```

การรับส่งเมสเสจระหว่างโพรเซส P และโพรเซส Q ในลักษณะนี้เรียกว่า Asynchronous อย่างไรก็ตามยังมี 2-3 กรณีที่ไม่เข้ากลุ่มใดตามที่กล่าวมาแล้วนี้ โดยสามารถอธิบายได้ดังต่อไปนี้

1. การส่งเมสเสจของโพรเซสได้โดยไม่ต้องคอย กรณีนี้ถ้าผู้รับยังได้รับเมสเสจก่อนที่ผู้ส่งจะส่งเมสเสจอื่น เมสเสจแรกจะหายไป ข้อได้เปรียบของกรณีนี้ก็คือ เมสเสจขนาดใหญ่ไม่จำเป็นต้องสำเนาไว้หลายครั้งส่วนข้อเสียเปรียบ คือ การเขียนโปรแกรมของงานจะมีความยุ่งยาก โพรเซสเหล่านี้จำเป็นต้องมีการซินโครไนซ์แบบพิเศษเพื่อให้มั่นใจว่าเมสเสจไม่สูญหายไปไหน ทั้งผู้รับและผู้ส่งไม่ต้องคำนวณบัฟเฟอร์ของเมสเสจ

2. การส่งเมสเสจของโพรเซสจะล่าช้าออกไปจนกว่าโพรเซสจะได้รับคำตอบ วิธีการนี้นำมาใช้ในระบบปฏิบัติการที่ชื่อว่า Thoth โดยในระบบนี้เมสเสจจะมีขนาดแน่นอน (8 word) โพรเซส P ที่เมสเสจจะถูกบล็อกจนกว่าโพรเซสที่ทำหน้าที่รับเมสเสจแล้วส่งคำตอบกลับมา 8 word ในรูปแบบคำสั่ง reply(P,message) เมสเสจที่ตอบมานี้จะเขียนทับบัฟเฟอร์ดั้งเดิมของเมสเสจ ข้อแตกต่างระหว่าง send คือ คำสั่ง send จะทำให้โพรเซสที่ส่งเมสเสจเกิดการบล็อก ในขณะที่มีการตอบกลับจะยอมให้ทั้งโพรเซสที่ส่งเมสเสจ และโพรเซสที่รับเมสเสจสามารถทำงานต่อไปได้เลยโดยไม่มีการบล็อก

2.9 เงื่อนไขข้อยกเว้น

ระบบเมสเสจมีประโยชน์บนสภาพแวดล้อมระบบแบบกระจาย ซึ่งโพรเซสอาจจะอยู่บนเครื่องอื่น ๆ ในสภาพแวดล้อมที่กล่าวถึง ความน่าจะเป็นที่จะเกิดข้อผิดพลาดระหว่างการติดต่อสื่อสารระหว่าง โพรเซสจะมากกว่าในสภาพแวดล้อมที่เป็นเครื่องเดียว สำหรับสภาพแวดล้อมที่เป็นเครื่องเดียว เมสเสจจะอยู่ในหน่วยความจำที่ใช้แชร์ข้อมูลร่วมกัน ถ้าเกิดข้อผิดพลาดขึ้นระบบโดยรวมจะล่ม แต่สำหรับในระบบแบบกระจายเมสเสจจะถูกถ่ายโอนไปตามสาย การเกิดข้อผิดพลาดที่เครื่องใดเครื่องหนึ่งไม่จำเป็นที่ระบบโดยรวมจะล่ม ทั้งระบบที่เป็นศูนย์กลางหรือระบบแบบกระจายข้อผิดพลาดอาจจะได้รับการแก้ไข ตอนนี้มาพิจารณาเงื่อนไขยกเว้นที่ระบบต้องดูแลเมสเสจ

2.9.1 การสิ้นสุดของโพรเซส

ถ้าผู้รับหรือผู้ส่งเมสเสจสิ้นสุดก่อนเมสเสจจะเอกซิคิวต์ ในสภาวะแบบนี้ทำให้เมสเสจจะถูกกำจัด ทำให้ผู้รับไม่ได้รับเมสเสจ หรือผู้ส่งไม่ได้ส่งเมสเสจ ลองพิจารณา 2 กรณีดังนี้

1. ผู้รับโพรเซส P อาจจะรอเมสเสจจากโพรเซส Q ที่สิ้นสุดไปแล้ว ถ้าไม่มีการกระทำใด ๆ เกิดขึ้น โพรเซส P จะถูกบล็อกตลอดไป ในกรณีนี้ระบบอาจจะสั่งให้โพรเซส P สิ้นสุด หรืออาจจะแจ้งให้ โพรเซส P ทราบว่าโพรเซส Q สิ้นสุดไปแล้วก็ได้

2. โพรเซส P อาจจะส่งเมสเสจไปยังโพรเซส Q ที่สิ้นสุดไปแล้ว รูปแบบการจัดบัฟเฟอร์แบบอัตโนมัติจะไม่เกิดอันตรายใด ๆ โดย P ยังคงเอ็กซิคิวต์ต่อไป ถ้าโพรเซส P ต้องการทราบว่าเมสเสจของตนนั้นโพรเซส Q เอ็กซิคิวต์หรือไม่ จะต้องมีโปรแกรมพิเศษสำหรับการแจ้งให้ทราบ แต่ในกรณีที่ไม่มีบัฟเฟอร์ โพรเซส P จะถูกบล็อกตลอดไปเช่นเดียวกับข้อ 1 ระบบอาจจะสั่งให้ P สิ้นสุด หรืออาจจะแจ้งให้ P ทราบว่า Q สิ้นสุดไปแล้วก็ได้

2.9.2 การสูญหายของเมสเสจ

เมสเสจจากโพรเซส P ที่ส่งไปยังโพรเซส Q อาจจะสูญหายระหว่างทางของการสื่อสารก็ได้ ซึ่งอาจจะเป็นข้อผิดพลาดด้านฮาร์ดแวร์หรือสายสื่อสาร มี 3 วิธีพื้นฐานในการจัดการเหตุการณ์นี้

1. ระบบปฏิบัติการมีหน้าที่รับผิดชอบในการตรวจสอบการสูญหายนี้ เพื่อส่งเมสเสจไปใหม่

2. โพรเซสที่ทำหน้าที่ส่งเมสเสจ มีหน้าที่รับผิดชอบในการตรวจสอบการสูญหายเพื่อส่งเมสเสจใหม่ถ้าต้องการอีก

3. ระบบปฏิบัติการมีหน้าที่รับผิดชอบในการตรวจสอบการสูญหายนี้ หลังจากนั้นจะแจ้งให้โพรเซสที่ทำหน้าที่ส่งเมสเสจที่เกิดการสูญหาย เพื่อให้ส่งเมสเสจไปใหม่

จะทำการตรวจสอบได้อย่างไรว่าเมสเสจเกิดการสูญหายไปจากระบบ วิธีการตรวจสอบที่ง่ายที่สุดคือใช้การกำหนดช่วงเวลา Timeout เมื่อเมสเสจถูกส่งออกไปจะมีสัญญาณตอบกลับมา ระบบปฏิบัติการหรือโพรเซสอาจจะกำหนดช่วงเวลาที่แน่นอน โดยคาดการณ์จากสัญญาณการตอบรับที่ มาถึง ถ้าช่วงเวลาเกิดการเหลื่อมก่อนที่สัญญาณการตอบรับจะมาถึง ระบบปฏิบัติการอาจจะกำหนดว่า เมสเสจนั้นสูญหายไปและจะต้องส่งเมสเสจใหม่ อย่างไรก็ตามถ้าเมสเสจไม่ได้สูญหายไปจากระบบจริงแต่เกิดการใช้เวลาในการเดินทางในเครือข่ายมาก ในกรณีนี้ระบบอาจจะต้องทำสำเนาเมสเสจที่ส่งไปยังเครือข่ายและจะต้องมีกลไกเพื่อแบ่งแยกประเภทของเมสเสจเหล่านั้นด้วย

2.10 สรุป

ปัญหาหนึ่งในการอธิบายเกี่ยวกับระบบปฏิบัติการ คือ เราจะเรียกกิจกรรมของซีพียูว่าอย่างไรดี ในระบบการทำงานแบบกลุ่มเรียกว่า Job ในระบบแบ่งปันส่วนเรียกว่า โปรแกรมผู้ใช้หรือ Task แม้แต่ระบบผู้ใช้คนเดียวอย่าง MS-DOS และ Macintosh ผู้ใช้เครื่องอาจให้หลาย ๆ โปรแกรมทำงานพร้อมกันได้โดยโปรแกรมหนึ่งเป็นโปรแกรมโต้ตอบและที่เหลือเป็นแบบกลุ่ม

ทั้งนี้อาจเรียกโปรแกรมที่กำลังทำงานอยู่ว่าโพรเซส การทำงานของโพรเซสต้องเป็นแบบลำดับ หรืออีกนัยหนึ่งคือ ณ เวลาใดเวลาหนึ่ง จะต้องมีหนึ่งคำสั่งที่กำลังดำเนินการอยู่ในนามของโพรเซสนี้ โพรเซสไม่ได้หมายความเพียงโปรแกรมและการทำงานเท่านั้น แต่รวมถึง Program counter (พวกรีจิสเตอร์ในซีพียู), Stack (ข้อมูลชั่วคราว), Data section (ใช้เก็บตัวแปรแบบ Global) โปรแกรมที่เป็นกระบวนการเหมือนสิ่งไม่มีชีวิต (Passive entity) เช่น คำสั่งที่เก็บไว้ในจานบันทึก ส่วนกระบวนการเสมือนสิ่งมีชีวิต (Active entity) มี Program counter ทำหน้าที่ชี้บรรทัดคำสั่งที่จะทำงานต่อไป และกลุ่มของทรัพยากรที่เกี่ยวข้อง

ถึงแม้ว่าอาจมีโพรเซสสองโพรเซสทำงานอยู่บนโปรแกรมเดียวกัน ต้องนับว่าเป็นการทำงานตามลำดับแยกจากกัน เป็นเรื่องปกติที่โพรเซสหลักจะสร้างโพรเซสย่อยหลาย ๆ โพรเซสในขณะทำงาน การสื่อสารระหว่างโพรเซสจะอยู่ในรูปแบบที่เรียกว่า การส่งและการรับ ซึ่งสามารถออกแบบในการรับ-ส่งข่าวสารเป็นแบบบล็อกเรียกว่า การประสานเวลา หรือแบบ Non-Blocking บางทีเรียกว่าแบบไม่ประสานเวลาก็ได้

การส่งแบบบล็อกฝั่งผู้ส่งจะต้องหยุดรอจนกว่าข่าวสารที่ถูกส่งไปถึงที่รับแล้ว ซึ่งรับโดยโพรเซสหรือตู้ไปรษณีย์ก็ได้ แล้วมีการตอบรับจากผู้รับ การส่งแบบ Non-Blocking send ผู้ส่งไม่จำเป็นต้องการการตอบรับจากผู้รับ การรับแบบ Blocking receive ฝั่งผู้รับจะต้องหยุดรอจนกว่าจะได้รับข่าวสารอย่างครบถ้วน การรับแบบ Non-Blocking receive ฝั่งผู้รับไม่ต้องรอจนกระทั่งส่งเสร็จ สามารถเรียกใช้ข่าวสาร ซึ่งอาจได้ข่าวสารที่ถูกต้องหรือเป็นค่าว่าง (null) ก็ได้ การประสานเวลาของโพรเซสจะได้ศึกษาในบทต่อไป

แบบฝึกหัดท้ายบทที่ 2

1. จงอธิบายและเขียนแผนภาพสถานะของโพรเซสในแต่ละสถานะมาโดยสังเขป

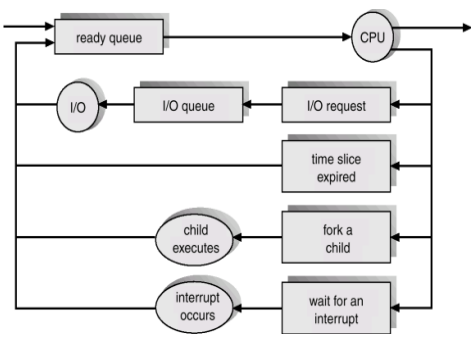
2. จงอธิบายแต่ละองค์ประกอบของ PCB (Process Control Block) มาอย่างน้อย 5 องค์ประกอบ พร้อมยกตัวอย่างการทำงาน

3. จงอธิบายการสลับการทำงานระหว่างโพรเซส

4. จงบอกความแตกต่างของ Job queue, Ready queue และ Device queue

5. จงอธิบายเหตุผลอะไรบ้างโพรเซสที่ต้องแบ่งข้อมูลให้กับโพรเซสอื่นที่เป็นโพรเซสร่วม ที่ทำให้ต้องจัดเตรียมสิ่งแวดล้อมให้กับโพรเซสที่ต้องทำงานร่วมกัน

6. จงใช้แผนภาพนี้อธิบายการทำงานของโพรเซสร่วมกับสถานะของโพรเซส



7. จงเขียนแผนภาพพร้อมอธิบายการสื่อสารกันของโพรเซสแบบส่งผ่านข้อความ และการใช้หน่วยความจำร่วมกัน

8. ในการปรับอัตราในการสื่อสารระหว่างโพรเซส จงอธิบายว่ามีประโยชน์อย่างไร

9. จงอธิบายการพิจารณาเงื่อนไขข้อยกเว้นที่ระบบต้องดูแลเมสเสจ มีเหตุผลอะไรบ้าง

บรรณานุกรมประจำบทที่ 2

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

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

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

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

Andrew S. Tanenbaum, 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/

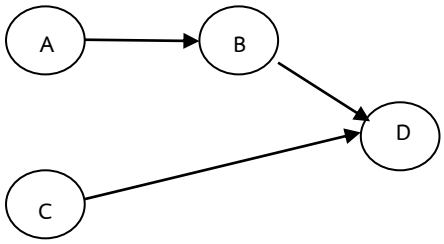
บทที่ 3
การประสานเวลาของโพรเซส

ในระบบคอมพิวเตอร์ที่มีหน่วยประมวลผลเดียวแต่มีหลายโพรเซสทำงานอยู่ในระบบ โพรเซสต่าง ๆ สลับกันทำงานด้วยความถี่หลายครั้งต่อวินาที คล้ายกับว่าโพรเซสต่าง ๆ ทำงานไปพร้อม ๆ กัน เครื่องที่มีหน่วยประมวลผลมากกว่าหนึ่งหน่วยประมวลผล โพรเซสต่าง ๆ สามารถทำงานไปพร้อมกันได้จริงบนหน่วยประมวลผลคนละตัวกัน สภาพเช่นนี้เรียกว่าภาวะพร้อมกัน (Concurrency) ระบบคอมพิวเตอร์ที่ทำงานแบบมัลติโปรแกรมมิ่ง มีโพรเซสหลายโพรเซสทำงานพร้อมกัน โดยที่แต่ละโพรเซสอาจมีการทำงานเป็นอิสระต่อกัน หรือมีการทำงานที่ขึ้นต่อกัน ดังนั้นหากมีความต้องการที่จะเข้าใช้ทรัพยากรที่มีอยู่พร้อม ๆ กัน เช่น ใช้หน่วยความจำตำแหน่งเดียวกันพร้อมกัน ทำให้เกิดปัญหาการเขียนอ่าน และสำหรับหน่วยความจำที่ใช้ร่วมกัน อาจทำให้ผลของการทำงานคลาดเคลื่อนจากที่ควรจะเป็น

3.1 การประสานเวลาของโพรเซส

โดยทั่วไปทุกโพรเซสต้องการให้มีการประมวลผลอย่างเป็นอิสระ เสมือนมีโพรเซสเดียวที่กำลังทำงานอยู่ ลักษณะความเป็นอิสระนี้เรียกว่า Asynchronous ซึ่งในความเป็นจริงมีหลายโพรเซสเกิดขึ้นในเวลาเดียวกัน และอาจมีความจำเป็นต้องการใช้ทรัพยากรที่มีอยู่อย่างจำกัดพร้อมกัน เช่น หน่วยความจำ ระบบปฏิบัติการจึงเข้ามาช่วยจัดการแต่ละโพรเซส ให้สามารถประมวลผลได้พร้อมกัน (Concurrent processing) และมีการทำงานร่วมกัน (Cooperating) เป็นผลให้โพรเซสในระบบได้มี การใช้ทรัพยากรร่วมกัน ช่วยเพิ่มความเร็วในการทำงานของระบบสูงขึ้นและมีความสะดวกในการทำงาน ในการทำงานของโพรเซสที่ต้องการให้มีการทำงานร่วมกัน จำเป็นต้องมีการสื่อสารกันระหว่างโพรเซส (Inter process communication) จำเป็นต้องให้ทุกโพรเซสทำงานประสานกันเป็นอย่างดี การเข้าจังหวะของโพรเซสหรืออาจเรียกว่าโพรเซสนั้นเกิดการประสานเวลาของโพรเซส (Process synchronization) การประสานเวลาของโพรเซสจะหมายถึง การทำงานของโพรเซสที่ต้องมีการเกี่ยวข้องกันอาจเป็นเพราะ การใช้ทรัพยากรร่วมกัน หรืออาจจะเป็นการรอให้โพรเซสทำงาน หลังจากที่โพรเซสอื่นทำงานแล้ว

เพื่อความเข้าใจยิ่งขึ้นลองพิจารณาภาพที่ 3.1 จะเห็นว่าโพรเซส B จะเริ่มทำงานได้ก็ต่อเมื่อโพรเซส A ทำงานเสร็จเรียบร้อยไปแล้ว (ซึ่งต่างกับโพรเซส C ที่ไม่ต้องรอโพรเซสใดเลยก็สามารถทำงานได้ทันที และไม่มีความเกี่ยวข้องกับโพรเซส A และ B เลย) ในทำนองเดียวกันโพรเซส D จะทำงานได้ก็ต่อเมื่อมีการทำงานโพรเซส B และโพรเซส C ทำงานเสร็จเรียบร้อยไปแล้วนั่นเอง

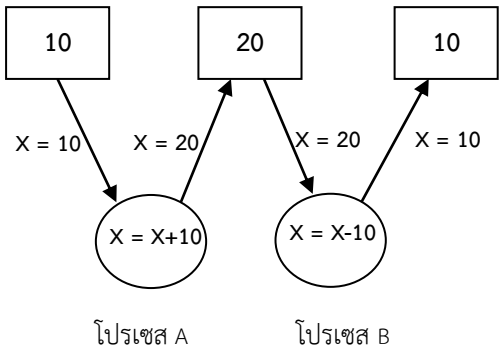


ภาพที่ 3.1 แสดงการประสานเวลาของโพรเซส

3.2 ปัญหาภาวะพร้อมกัน

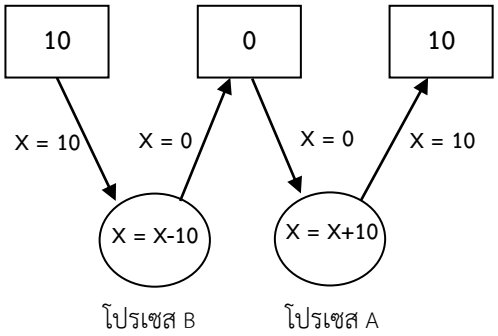
ในระบบที่มีการทำงานบนซีพียูเดียว หรือทำงานบนซีพียูหลายตัว บางครั้งมีความจำเป็นที่ต้องใช้ทรัพยากรร่วมกัน และเป็นทรัพยากรที่ต้องทำงานครั้งละ 1 งาน ไม่สามารถทำงานพร้อมกันได้ จะเรียกทรัพยากรเหล่านี้ว่า ทรัพยากรที่ไม่สามารถใช้ร่วมกันได้ (Non-dedicated resources) ดังนั้น ในขณะที่โพรเซสหนึ่งกำลังใช้ทรัพยากรอยู่ โพรเซสอื่นจะไม่สามารถเข้ามาใช้ทรัพยากรนั้น ๆ ได้ เพื่อให้ได้ผลของการทำงานที่ถูกต้อง เช่น เครื่องพิมพ์ เป็นต้น โพรเซสที่ใช้ข้อมูลร่วมกันหรือร่วมกันทำงาน การใช้ทรัพยากรต่าง ๆ เช่น หน่วยความจำ และอุปกรณ์เข้าออกร่วมกันได้ระหว่างโพรเซสทำให้เกิดปัญหาต่าง ๆ ตามมาดังตัวอย่างต่อไปนี้

ตัวอย่าง ในการใช้ทรัพยากรและข้อมูลร่วมกันโพรเซสสองโพรเซสขึ้นไป อาจก่อให้เกิดปัญหาในการทำงานที่ถูกต้องสมมุติโพรเซสทั้งสองเป็นอิสระต่อกัน ทำงานในเวลาพร้อม ๆ กัน และใช้ข้อมูลเดียวกันคือตัวแปร X ซึ่งกำหนดค่าเริ่มต้นไว้ที่ 10 โดยที่ โพรเซส A จะเป็นการเพิ่มค่าเข้าไปอีก 10 (X = X+10) ส่วนโพรเซส B จะเป็นการลดค่าจากเดิมออกไป 10 (X = X-10)



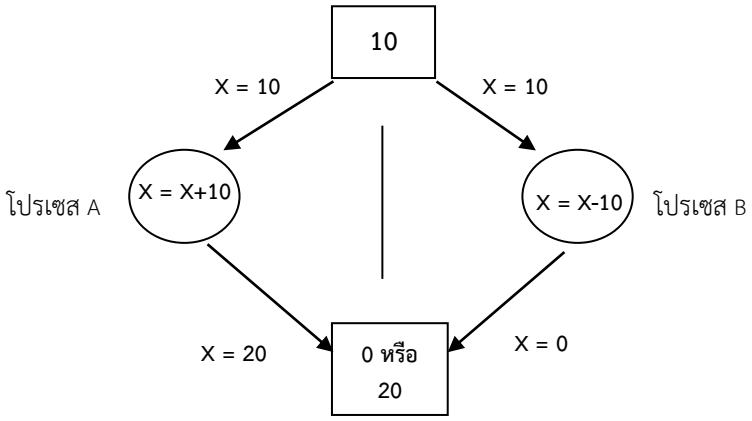
ภาพที่ 3.2 โพรเซส A ทำงานเสร็จก่อน โพรเซส B

จากภาพที่ 3.2 โพรเซส A ทำงานก่อนโพรเซส B โดยอ่านค่า X เข้ามาซึ่งมีค่าเป็น 10 แล้วเพิ่มเข้าไปอีก 10 ทำให้ X มีค่าเป็น 20 แล้วเขียนลงหน่วยความจำ หลังจากนั้นโพรเซส B จึงอ่านค่า X เข้ามาซึ่งมีค่าเป็น 20 แล้วลบค่าออกไป 10 ทำให้ X มีค่าเป็น 10 ลงหน่วยความจำ



ภาพที่ 3.3 โพรเซส B ทำงานเสร็จก่อน โพรเซส A

จากภาพที่ 3.3 การทำงานจะคล้ายกับภาพที่ 3.2 และได้ผลลัพธ์เหมือนกัน แต่โพรเซส B ทำงานก่อนโพรเซส A โดยอ่านค่า X เข้ามาซึ่งมีค่าเป็น 10 แล้วลบค่าออกไป 10 ทำให้ X มีค่าเป็น 0 จากนั้นเขียนลงหน่วยความจำ หลังจากนั้นโพรเซส A จึงอ่านค่า X เข้ามาซึ่งมีค่าเป็น 0 แล้วเพิ่มค่าเข้าไป 10 ทำให้ X มีค่าเป็น 10 แล้วเขียนค่า X ลงหน่วยความจำ



ภาพที่ 3.4 โพรเซส A และ โพรเซส B ทำงานพร้อมกัน

สำหรับภาพที่ 3.4 ทั้งโพรเซส A และ B จะทำงานพร้อม ๆ กัน โดยค่าเริ่มต้น X เป็น 10 ทั้งสองโพรเซสจะได้ผลลัพธ์ที่ต่างกัน โดยโพรเซส A จะได้ค่า X เป็น 20 (จาก 10 เพิ่มอีก 10) ส่วนโพรเซส B จะได้ค่า X เป็น 0 (จาก 10 ลบออกไป 10) ถ้าโพรเซส A ทำงานเสร็จก่อนจะเขียนค่า X ซึ่งเท่ากับ 20 ลงหน่วยความจำ แต่ถ้าโพรเซส B ทำงานเสร็จก่อนจะเขียนค่า X ซึ่งเท่ากับ 0 ลงหน่วยความจำ ซึ่งทำให้เกิดข้อผิดพลาดเนื่องจากค่า X แน่นอน ขึ้นอยู่กับว่าโพรเซสใดทำงานเสร็จก่อน

3.3 สภาวะการแย่งชิง

จากตัวอย่างโพรเซสใช้ตัวแปรตัวหนึ่งในหน่วยความจำร่วมกันทั้งสองโพรเซส การแก้ไขข้อมูลตัวแปรนั้นพบว่า ลำดับของการแก้ไขก่อนหรือหลังมีความสำคัญต่อค่าของข้อมูลตัวนั้น ทำให้โพรเซสสอง

โพรเซสที่ต้องการใช้ทรัพยากรที่แชร์ไว้พร้อมกันในเวลาเดียวกัน เช่น หน่วยความจำ สื่อจัดเก็บข้อมูล หรือแชร์ไฟล์ ซึ่งโพรเซสทั้งสองสามารถอ่านหรือเขียนทรัพยากรเหล่านี้ได้พร้อมกัน ทำให้ผลลัพธ์อาจจะเกิดการผิดพลาดขึ้นได้ ขึ้นอยู่ว่าโพรเซสใดเข้ามาใช้ก่อน ทำให้ผลลัพธ์อาจจะเกิดการผิดพลาดขึ้นได้ ขึ้นอยู่กับว่าโพรเซสใดทำเสร็จก่อนเรียกสภาวะนี้ว่า สภาวะเงื่อนไขการแย่งชิง (Race condition) ดังนั้นเพื่อป้องกันการเกิดเงื่อนไขการแย่งชิงขึ้นตามตัวอย่าง ระบบต้องแน่ใจว่าในเวลาใด ๆ จะมีเพียงโพรเซสเดียวเท่านั้นที่กำลังเข้าจัดการข้อมูลตัวแปร X โดยต้องอาศัยกลไกบางอย่างในการประสานโพรเซสอย่างได้จังหวะกันจึงสามารถรับประกันความถูกต้องได้

3.4 การแก้ปัญหาส่วนวิกฤติ

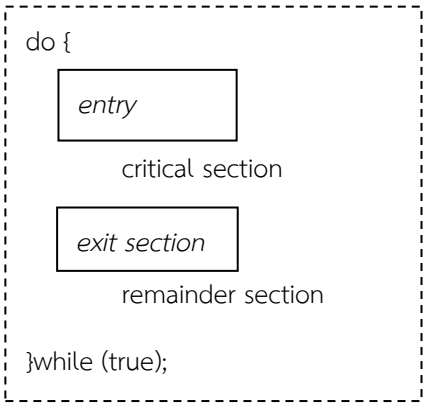
ปัญหาที่เกิดขึ้นจากสถานการณ์ที่มีเงื่อนไขการแย่งชิงนี้ เป็นผลมาจากการที่แต่ละโพรเซสมีการเข้าใช้ข้อมูลบางอย่างร่วมกัน โดยไม่มีการประสานการทำงานระหว่างกัน ต่างฝ่ายต่างทำงานตามคำสั่งไปจนกระทั่งถึงบริเวณที่มีคำสั่งใช้งานข้อมูลร่วมกับอีกฝ่ายหนึ่ง ซึ่งอาจทำให้เกิดผลลัพธ์ที่ผิดพลาดได้ ดังนั้นหากระบบสามารถประสานการเข้าทำงานในบริเวณที่แต่ละโพรเซสมีการใช้ข้อมูลร่วมกันก็จะช่วยแก้ปัญหาที่เกิดขึ้นได้ ลองพิจารณาระบบที่ประกอบด้วย n โพรเซส {P0, P1, ..., Pn-1} โดยแต่ละโพรเซสมีส่วนโปรแกรมซึ่งทำการเปลี่ยนแปลงค่าข้อมูลตัวแปร ปรับปรุงตาราง บันทึกข้อมูลลงแฟ้ม และอื่น ๆ ที่ต้องมีการทำงานร่วมกัน เรียกส่วนโปรแกรมบริเวณนี้ว่า ส่วนวิกฤติ (Critical-section) แล้วกำหนดคุณลักษณะที่สำคัญของระบบคือ ขณะที่โพรเซสหนึ่งกำลังปฏิบัติการอยู่ในส่วนวิกฤติของตนเอง จะต้องไม่มีโพรเซสอื่นได้รับอนุญาตให้เข้าไปปฏิบัติการในส่วนวิกฤติของโพรเซสเหล่านั้น

ดังนั้นการปฏิบัติการในส่วนวิกฤติของโพรเซสต้องเป็นไปในลักษณะที่มีการกีดกัน (Mutual exclusion) โดยที่แนวทางแก้ปัญหาในส่วนวิกฤติคือ การออกแบบข้อตกลงหรือโปรโตคอล (Protocol) เพื่อให้โพรเซสทั้งหลายสามารถทำงานร่วมกันได้อย่างถูกต้อง ทั้งนี้แต่ละโพรเซสต้องร้องขอคำอนุญาตเพื่อเข้าสู่ส่วนวิกฤติของตนเอง ซึ่งอยู่ในรูปแบบของการตรวจสอบตามเงื่อนไขที่กำหนด โดยส่วนโปรแกรมที่ติดตั้งการร้องขอนี้จะเป็นส่วนเริ่มต้นก่อนเข้าสู่ส่วนวิกฤติ และตามด้วยส่วนจบการทำงานซึ่งอยู่ถัดจากส่วนวิกฤติ ท้ายสุดอาจเป็นส่วนโปรแกรมที่ไม่มีความเกี่ยวข้องกับโพรเซสอื่น

การเขียนแสดงรูปแบบขั้นตอนวิธีสำหรับแก้ปัญหาส่วนวิกฤติจะมีการกำหนดเฉพาะตัวแปรที่ใช้เพื่อการประสานอย่างได้จังหวะกันเท่านั้น และบอกถึงโพรเซสเฉพาะบางโพรเซส Pi ซึ่งส่วนเริ่มต้นและส่วนจบของการทำงานในส่วนวิกฤติ จะล้อมอยู่ในกรอบรูปสี่เหลี่ยมเพื่อเน้นถึงความสำคัญของส่วนนี้ คำตอบสำหรับการแก้ปัญหาส่วนวิกฤติต้องเป็นไปตามเงื่อนไขทั้งสามข้อต่อไปนี้

3.4.1 การกีดกัน (Mutual exclusion)

ถ้าโพรเซส Pi กำลังปฏิบัติการอยู่ในส่วนวิกฤติแล้ว ต้องไม่มีโพรเซสอื่นใดกำลังอยู่ในส่วนวิกฤติของโพรเซสเหล่านั้นด้วย นั่นคือในขณะใดขณะหนึ่งมีเพียงโพรเซสเดียวเท่านั้นที่อยู่ในส่วนวิกฤติของตน



ภาพที่ 3.5 โครงสร้างภาษาโดยทั่วไปของการดำเนินการของโพรเซสปกติ

3.4.2 ความก้าวหน้า (Progress)

ถ้าไม่มีโพรเซสใดกำลังปฏิบัติการอยู่ในส่วนวิกฤติแล้วมีบาง โพรเซสเข้าสู่ส่วนวิกฤติของตน ในการเลือกว่าโพรเซสใดจะได้เข้าสู่ส่วนวิกฤติของตนต่อไปนั้น ระบบก็จะพิจารณาจากทุก ๆ โพรเซส ยกเว้นโพรเซสที่กำลังทำงานอยู่ในส่วนที่เหลือนั้น โดยการเลือกนี้ต้องไม่เลื่อนออกไปอย่างไม่มีกำหนด

3.4.3 การรออย่างมีขอบเขต (Bounded waiting)

จะต้องมีขอบเขตของเวลาในการรอ โดยเริ่มนับตั้งแต่เวลาที่โพรเซสอื่นได้รับอนุญาตให้เข้าสู่ส่วนวิกฤติไป หลังจากมีโพรเซสหนึ่งร้องขอจนมาถึงเวลาที่ระบบทำตามคำร้องขอของโพรเซสนั้น

ทั้งนี้ภายใต้ข้อสมมติว่า แต่ละโพรเซสกำลังปฏิบัติการที่ความเร็วขนาดหนึ่ง (ไม่เป็นศูนย์) แต่ไม่ให้มีการกำหนดสมมติฐานที่เกี่ยวกับความเร็วสัมพัทธ์ของโพรเซสทั้งหมด n โพรเซส จากรูปแบบขั้นตอนวิธีสำหรับการแก้ปัญหาส่วนวิกฤติและเงื่อนไขทั้งสามข้อข้างต้นที่ต้องคำนึงถึงในการแก้ปัญหาส่วนวิกฤติเป็นผลให้เกิดแนวคิดการติดตั้งใช้งานในแต่ละโพรเซส ซึ่งจะต้องติดตั้งส่วนเริ่มต้นก่อนเข้าสู่ส่วนวิกฤติเพื่อให้เกิดการกีดกันดังเงื่อนไขข้อ 1 และติดตั้งส่วนจบการทำงานเมื่อออกจากส่วนวิกฤติเพื่อปล่อยให้โพรเซสอื่นได้มีโอกาสเข้าใช้ส่วนวิกฤติของตนเองบ้าง นั่นคือ เป็นไปตามเงื่อนไขข้อที่ 2 และ 3 ซึ่งในโครงสร้างของแต่ละโพรเซสที่มีส่วนเริ่มต้นก่อนเข้าสู่ส่วนวิกฤติ มีจุดประสงค์เพื่อให้โพรเซสทั้งหลายตรวจสอบว่า ขณะนั้นมีโพรเซสใดกำลังอยู่ในส่วนวิกฤติของโพรเซสนั้นหรือไม่ ถ้าไม่มีโพรเซสที่ต้องการเข้าสู่ส่วนวิกฤติของตนเองก็สามารถเข้าทำงานได้ทันที ถ้ามีโพรเซสเหล่านั้นก็ไม่สามารถเข้าสู่ส่วนวิกฤติได้ตามต้องการ ดั้งนั้นปัญหาที่เกิดขึ้นคือ โพรเซสที่เข้าสู่ส่วนวิกฤติไม่ได้ในขณะนั้นจะดำเนินการรอต่อไปอย่างไร ทั้งนี้แนวคิดในการแก้ปัญหาโดยทั่วไปมี 2 รูปแบบคือ

1. การรอแบบไม่ว่าง (Preemptive kernels)

ในแต่ละโพรเซสมีการกำหนดเงื่อนไขที่ต้องการใช้ในการตรวจสอบก่อนเข้าสู่ส่วนวิกฤติ ซึ่งถ้าหากไม่สามารถผ่านเงื่อนไขนี้ไปได้ โพรเซสไม่สามารถเข้าสู่ส่วนวิกฤติของตนเองได้ และวนเวียนตรวจสอบเงื่อนไขนั้นอยู่ตลอดเวลาจนกว่าจะได้เข้าสู่ส่วนวิกฤตินั่นคือ ในระหว่างรอโพรเซสนั้นยังต้องทำคำสั่งทำซ้ำและทดสอบเงื่อนไขอยู่เรื่อย ๆ

2. การขัด (Non-preemptive kernels)

ในการรอแบบไม่ว่างนั้นไม่ได้ทำให้ระบบเกิดผลงานใด ๆ ขึ้นมา จึงเป็นการใช้วงรอบการทำงานของหน่วยประมวลผลกลางที่ไม่ได้ประโยชน์ และยังต้องแย่งชิงกันเข้าใช้หน่วยประมวลผลกลางระหว่างโพรเซสร่วมอื่น ๆ ที่กำลังวนลูปรอเข้าสู่ส่วนวิกฤติ รวมทั้งโพรเซสที่เป็นแบบอิสระทั้งหลายในระบบด้วย ดังนั้นจึงได้มีการพัฒนาให้ทำการบล็อกโพรเซสที่ต้องการเข้าสู่ส่วนวิกฤติแต่ยังไม่อาจเข้าได้ เนื่องจากมีโพรเซสอื่นกำลังทำงานอยู่ในส่วนวิกฤติ ซึ่งการบล็อกนี้ระบบจะนำโพรเซสเหล่านั้นไปรออยู่ในแถวของแต่ละเงื่อนไขและตัวแปรที่ใช้ในการตรวจสอบก่อนเข้าสู่ส่วนวิกฤติ ทำให้โพรเซสเหล่านี้ไม่ต้องเข้ามาแย่งชิงกันใช้ซีพียู และเมื่อใดที่ไม่มีโพรเซสใด ๆ อยู่ในส่วนวิกฤตินั่นคือ โพรเซสล่าสุดที่เข้าใช้ส่วนวิกฤติของตนเองได้ออกจากส่วนวิกฤตินั้นแล้ว จะต้องปลุกหรือใช้คำสั่งบอกให้ระบบทราบ เพื่อให้นำโพรเซสที่รออยู่ในแถวคอยตามเงื่อนไขที่รอได้กลับเข้ามาทำงานในส่วนวิกฤติของตนเองต่อไป เครื่องมือที่มีการติดตั้งใช้งานในลักษณะนี้ได้แก่ เซมาฟอร์ มอนิเตอร์ และบริเวณวิกฤติ เป็นไปได้ที่สองโพรเซสของเคอร์เนลอยู่ในโหมดรันพร้อมกันในซีพียูที่แตกต่างกัน เพราะเหตุใดโพรเซสจึงชอบเคอร์เนลที่ทำงานก่อนมากกว่าเคอร์เนลที่ทำงานทีหลัง การรอแบบไม่ว่างเหมาะสมสำหรับโปรแกรมแบบ Real time ซึ่งยอมให้โพรเซสแบบ Real time จองโพรเซสที่กำลังทำงานอยู่ในปัจจุบันบนเคอร์เนล นอกจากนี้การรอแบบไม่ว่างอาจตอบสนองเพิ่มเติมเนื่องจากมีความเสี่ยงน้อยกว่าโพรเซสของเคอร์เนลโหมดที่ทำงานโดยไม่มีการควบคุมโพรเซสที่มีเวลาการทำงานนานซึ่งได้ทำงานก่อน และทำให้ซีพียูอื่นต้องรอโพรเซสนี้จนกว่าจะทำงานเสร็จ ผลกระทบนี้สามารถลดลงได้ด้วยการออกแบบเคอร์เนลโค้ดที่ไม่ทำงานแบบนี้อย่างแน่นอน

3.5 การประมวลผลพร้อมกันโดยวิธีการทางซอฟต์แวร์

เป็นวิธีการทางซอฟต์แวร์ที่เข้ามาช่วยควบคุมการทำงานของระบบให้มีการประมวลผลพร้อมกัน โดยมีคุณสมบัติของการไม่เกิดร่วม มีอัลกอริทึมที่น่าสนใจคือ

- อัลกอริทึมของเดกเกอร์ (Dekker's algorithm)
- อัลกอริทึมของปีเตอร์สัน (Peterson's algorithm)

3.5.1 อัลกอริทึมของเดกเกอร์

ทำการพัฒนาโปรแกรมที่ประกอบด้วย 2 โพรเซส โดยสามารถทำงานพร้อมกัน และเกิดคุณสมบัติการไม่เกิดร่วม ตัวอย่างประกอบด้วยโพรเซส P0 และ P1 ที่ทำงานพร้อมกัน

```
Process 0;
...
while(turn !=0);
<CRITICAL SECTION>
turn = 1;
```

```
Process 1;
...
while(turn !=1);
<CRITICAL SECTION>
turn = 0;
```

ภาพที่ 3.6 โครงสร้างภาษาที่ใช้อัลกอริทึมของเดกเกอร์ของโพรเซส P0 และ P1

จากโค้ดตัวอย่างตามภาพที่ 3.6 เมื่อเริ่มทำงานตัวแปร turn ถูกกำหนดค่าให้เป็น 0 ทำให้โพรเซส P0 สามารถเข้าไปทำงานในส่วนวิกฤติส่วนโพรเซส P1 จะต้องวนลูปรอจนกระทั่งโพรเซส P0 ทำงานในส่วนวิกฤติเสร็จ และกำหนดให้ turn มีค่าเป็น 1 โพรเซส P1 จึงจะออกจากการวนลูป และสามารถเข้าไปทำงานในส่วนวิกฤติได้ จะเห็นว่าโพรเซส P0 และ P1 สามารถทำงานพร้อม ๆ กัน และผลัดกันเข้าไปทำงานในส่วนวิกฤติ นั่นคือเกิดคุณสมบัติการไม่เกิดร่วม แต่การทำงานของอัลกอริทึมนี้อาจเกิดปัญหา Busy Waiting ในกรณีที่โพรเซส P0 กำลังทำงานในส่วนวิกฤติ และเกิดหมดเวลาการทำงาน (Timeout) ก่อนที่ P0 จะเปลี่ยนค่าของตัวแปร turn ทำให้ P1 ไม่สามารถเข้าไปทำงานในส่วนวิกฤติได้ แม้ว่าจะมีโอกาสเข้าไปทำงานในส่วนวิกฤติ (เพราะ P0 หมดเวลาการทำงาน) ทำให้ P1 ต้องวนลูปรอตลอดช่วงเวลาการทำงาน

นอกจากนี้ยังอาจเกิดปัญหา Lock Step Synchronization ซึ่งหมายถึง การที่โพรเซสใด ๆ จะเข้าไปทำงานในส่วนวิกฤติ จะต้องทำงานเรียงตามลำดับ เช่น ต้องมีการทำงานเรียงลำดับ P0, P1, P0, P1, Pn แม้ว่าบางโพรเซสอาจไม่ต้องการเข้าไปทำงานในส่วนวิกฤติ

```
//flag[] is boolean array; and turn is an integer
    flag[0] = false
    flag[1] = false
    turn    = 0   // or 1
```

```
P0:
   flag[0] = true;
   while (flag[1] == true) {
      if (turn ≠ 0) {
         flag[0] = false;
         while (turn ≠ 0) {
           // busy wait
         }
         flag[0] = true;
      }
   }
    // critical section
   ...
   turn    = 1;
   flag[0] = false;
   // remainder section
```

```
P1:
   flag[1] = true;
   while (flag[0] == true) {
      if (turn ≠ 1) {
         flag[1] = false;
         while (turn ≠ 1) {
           // busy wait
         }
         flag[1] = true;
      }
   }
    // critical section
   ...
   turn    = 0;
   flag[1] = false;
   // remainder section
```

ภาพที่ 3.7 อัลกอริทึมของเดกเกอร์ที่ได้รับการแก้ไขปัญหา lock step ของโพรเซส P0 และ P1

จากปัญหาของอัลกอริทึมของเดกเกอร์ในเวอร์ชั่นแรกในภาพที่ 3.6 จึงมีการแก้ไขปรับปรุง ในภาพที่ 3.7 อัลกอริทึมของเดกเกอร์ที่ได้รับการแก้ไขปัญหา lock step ของโพรเซส P0 และ P1 สร้าง

mutual exclusion ได้ โพสเซส P0 และ P1 สามารถเข้าทำงานใน critical sections ซ้ำได้ ทำให้แก้ปัญหา Lock step synchronization ได้ นอกจากนี้มีโพสเซสเนื่องจากมีนำการหน่วงเวลา(delay time) ในการติดอยู่ใน Loop ของแต่ละโปรเซส โดยระยะเวลาที่ใช้หน่วงเวลาจะเท่ากับค่าตัวเลขที่สุ่มขึ้นมา(random number) ทำให้ทั้ง 2 โปรเซสได้ตัวเลขสุ่มที่มีค่าไม่เท่ากัน จะทำให้มีโปรเซสหนึ่งสามารถออกจาก Loop และเข้าไปทำงานใน critical section ได้ สามารถแก้ปัญหา Deadlock ได้

3.5.2 อัลกอริทึมปีเตอร์สันโซลูชั่น (Peterson's Solution)

พื้นฐานของการแก้ไขปัญหาซอฟต์แวร์ที่เป็นที่นิยมสำหรับปัญหาส่วนวิกฤติที่รู้จักกันในชื่อของปีเตอร์สันโซลูชั่น เนื่องจากสถาปัตยกรรมคอมพิวเตอร์ที่ทันสมัยทำงานโดยใช้คำสั่งภาษาเครื่องพื้นฐาน เช่น โหลดและจัดการ ซึ่งไม่มีการรับประกันว่าอัลกอริทึมปีเตอร์สันโซลูชั่นจะทำงานได้อย่างถูกต้องในสถาปัตยกรรมนี้ อย่างไรก็ตามการแก้ไขปัญหาแบบนี้ให้คำอธิบายอัลกอริทึมที่ดีของการแก้ปัญหาส่วนวิกฤติ และแสดงบางความสัมพันธ์ที่ซับซ้อนในการออกแบบซอฟต์แวร์ อัลกอริทึมนี้จะจำกัดโพรเซสสองโพรเซสที่สลับการดำเนินการระหว่างส่วนวิกฤติและส่วนที่เหลือ ดังภาพที่ 3.8 ที่มีโพรเซสหมายเลข P0 และ P1 และโพรเซสทั้งสองต้องการข้อมูลสองค่าเพื่อใช้ร่วมกันระหว่างกัน

```
//flag[] is boolean array; and turn is an integer
flag[0]   = false;
flag[1]   = false;
turn;
```

```
P0: flag[0] = true;
    turn = 1;
    while (flag[1] == true && turn == 1)
    {
        // busy wait
    }
    // critical section
    ...
    // end of critical section
    flag[0] = false;
```

```
P1: flag[1] = true;
    turn = 0;
    while (flag[0] == true && turn == 0)
    {
        // busy wait
    }
    // critical section
    ...
    // end of critical section
    flag[1] = false;
```

ภาพที่ 3.8 โครงสร้างภาษาที่ใช้อัลกอริทึมของปีเตอร์สันโซลูชั่นของโพรเซส P0 และ P1

จากโค้ดตัวอย่างตามภาพที่ 3.7 เมื่อเริ่มการทำงาน ตัวแปร flag[0] และ flag[1] ถูกกำหนดค่าให้เป็นเท็จ ถ้าโพรเซสใดต้องการเข้าไปทำงานในส่วนวิกฤติ ให้ทำการกำหนดค่าของตัวแปร flag ของโพรเซสเป็นจริง เช่น ถ้าโพรเซส P0 ต้องการเข้าไปทำงานในส่วนวิกฤติ จะกำหนด flag[0] มีค่าเป็นจริง (true) ทำให้โพรเซส P1 ต้องวนลูปรอ หลังจากที่ P0 ทำงานในส่วนวิกฤติเสร็จแล้ว จะกำหนดค่าของ flag[0] เป็นเท็จ ทำให้โพรเซส P1 หลุดออกจากลูป และสามารถเข้าไปทำงานในส่วนวิกฤติได้ อัลกอริทึมนี้สามารถแก้ปัญหาการไม่เกิดร่วมสำหรับ 2 โพรเซส ที่สามารถพัฒนาให้มีการทำงานเป็น n โพรเซสได้

3.6 ฮาร์ดแวร์ประสานเวลา

การประมวลผลพร้อมกัน และมีคุณสมบัติการไม่เกิดร่วม โดยวิธีการทางฮาร์ดแวร์ สามารถทำได้หลายวิธี ดังนี้

1. การปิดกั้น (lock)
2. การปิดทางขัดจังหวะ (disabling interrupt)
3. คำสั่งทดสอบและเซ็ต (test and set instruction)

3.6.1 การปิดกั้น (Lock)

เมื่อโพรเซสต้องการเข้าไปทำงานในส่วนวิกฤติ โพรเซสจะต้องตรวจสอบก่อนว่า ส่วนวิกฤติถูกล็อค หรือถูกปิดกั้นหรือไม่ หากพบว่าส่วนวิกฤติไม่ถูกล็อค แสดงว่าขณะนั้นไม่มีโพรเซสใดกำลังทำงานในส่วนวิกฤติ โพรเซสสามารถเข้าไปทำงานในส่วนวิกฤติได้ โดยก่อนที่จะเข้าไปทำงานในส่วนวิกฤติ โพรเซสต้องใส่ล็อคเพื่อเป็นการปิดกั้นไม่ให้โพรเซสอื่นเข้าไปทำงานในส่วนวิกฤติได้ จนกระทั่งโพรเซสทำงานในส่วนวิกฤติเสร็จเรียบร้อยจึงปลดล็อค เพื่อเปิดโอกาสให้โพรเซสอื่นสามารถเข้าทำงานในส่วนวิกฤติได้

```
do {
    acquire lock
        critical section
    release lock
        remainder section
} while (TRUE);
```

ภาพที่ 3.9 รูปแบบของคำสั่ง Lock

การทำงานด้วยวิธีนี้ ระบบสามารถมีมากกว่าหนึ่งโพรเซสที่ทำงานพร้อมกันได้ โดยมีคุณสมบัติการไม่เกิดร่วม แต่อาจจะเกิดปัญหาในกรณีที่มีมากกว่าหนึ่งโพรเซสต้องการเข้าไปทำงานในส่วนวิกฤติพร้อมกัน และทดสอบล็อคแล้วพบว่าล็อคว่าง ก็สามารถเข้าไปทำงานในส่วนวิกฤติได้ ทำให้มีมากกว่าหนึ่งโพรเซสเข้าไปทำงานในส่วนวิกฤติพร้อมกัน

3.6.2 การปิดทางขัดจังหวะ (Disable Interrupt)

สาเหตุหนึ่งที่ทำให้เกิดปัญหาในการทำงานพร้อมกันหลายโพรเซส คือ ขณะที่โพรเซสหนึ่งกำลังทำงานในส่วนวิกฤติ อาจมีโพรเซสอื่นขอขัดจังหวะ เพื่อให้ได้โอกาสเข้าไปทำงานในส่วนวิกฤติ เป็นผลให้โพรเซสที่กำลังทำงานในส่วนวิกฤติต้องหยุดทำงาน โดยที่การทำงานยังไม่เสร็จสมบูรณ์ และสลับให้โพรเซสอื่นเข้าไปทำงานในส่วนวิกฤติแทน โดยที่โพรเซสที่เข้าทำงานภายหลัง อาจแทรกแซงการทำงานหรือเปลี่ยนแปลงค่า หรือมีการประมวลผลใด ๆ ที่ส่งผลกระทบต่อผลลัพธ์ในการทำงานของโพรเซสแรก

บนระบบโพรเซสเซอร์เดียวสามารถระงับการใช้ Interrupts สามารถรับประกันได้ว่าลำดับของชุดคำสั่งจะถูกกระทำการโดยปราศจาการบังคับให้ออกจากเวลาของโพรเซสเซอร์ แต่ในการทำงานบน

ระบบมัลติโพรเซสเซอร์ไม่สามารถใช้วิธีนี้ได้ การห้ามขัดจังหวะบนโพรเซสเซอร์หลายตัวนั้นใช้เวลานานมากในการส่งข่าวสารไปยังโพรเซสเซอร์ทุกตัว การส่งข่าวสารจะก่อให้เกิดการหน่วงเวลาไปยังส่วนวิกฤติของทุกโพรเซสเซอร์ซึ่งส่งผลให้ประสิทธิภาพของระบบลดลง เมื่อมีการบันทึกเวลาทุกครั้งที่ถูกเปลี่ยนค่าโดยการขัดจังหวะ

แต่การทำงานด้วยวิธีนี้ไม่เป็นที่นิยม เนื่องจากเป็นการไม่ถูกต้องที่ระบบอนุญาตให้มีโพรเซสปิดกั้นการขัดจังหวะของระบบ ในกรณีที่โพรเซสปิดกั้นการขัดจังหวะ แล้วไม่ได้ปล่อยการขัดจังหวะกลับคืนให้ระบบ อาจทำให้ปัญหาอื่นตามมา การปิดทางขัดจังหวะจะมีผลให้ได้ระบบที่เป็นการประมวลผลพร้อมกันและมีคุณสมบัติการไม่เกิดร่วม เฉพาะกรณีที่เป็นการประมวลผลบนซีพียูเดียว แต่กรณีที่ประมวลผลโดยใช้ระบบมัลติโพรเซสเซอร์ แม้ว่าโพรเซสได้ปิดทางขัดจังหวะแล้ว โพรเซสยังมีโอกาสถูกขัดจังหวะการทำงานของโพรเซสจากโพรเซสเซอร์อื่นได้

3.6.3 คำสั่งทดสอบและเซต (Test and Set Instruction)

วิธีนี้พยายามแก้ปัญหาของวิธีการปิดกั้น และการปิดทางขัดจังหวะ ด้วยการใช้คำสั่งทดสอบและเซต ซึ่งเป็นคำสั่งระดับฮาร์ดแวร์ โดยการทำงานของคำสั่งไม่สามารถถูกขัดจังหวะได้ การทำงานของวิธีนี้คือ โพรเซสที่ต้องการเข้าไปทำงานในส่วนวิกฤติจะต้องเรียกใช้คำสั่งทดสอบและเซต เพื่อตรวจสอบว่ามีโพรเซสใดทำงานอยู่ในส่วนวิกฤติหรือไม่ หากพบว่าขณะนั้นไม่มีโพรเซสใดทำงานในส่วนวิกฤติ โพรเซสจะเซตค่าล็อค เพื่อเป็นการป้องกันไม่ให้โพรเซสอื่นเข้าไปทำงานในส่วนวิกฤติพร้อมกันได้ ทำให้ระบบสามารถมีโพรเซสจำนวนมากทำงานร่วมกัน โดยมีคุณสมบัติการไม่เกิดร่วม

```
boolean test_and_set (boolean *target)
   {
        boolean rv = *target;
        *target = TRUE;
        return rv:
   }
```

ภาพที่ 3.10 รูปแบบของคำสั่ง Test-and-Set

ในหลาย ๆ ระบบคอมพิวเตอร์ที่ทันสมัยให้คำสั่งในเรื่องฮาร์ดแวร์พิเศษที่ช่วยให้สามารถเลือกที่จะทดสอบและปรับปรุงแก้ไขเนื้อหาของคำสั่ง หรือเพื่อที่จะสลับเนื้อหาของคำสั่งสองคำสั่งแบบ Atomically ที่เป็นดังหนึ่ง Uninterruptible Unit เราสามารถใช้คำสั่งพิเศษเหล่านี้เพื่อแก้ปัญหาส่วนวิกฤติ ในลักษณะเชิงสัมพัทธ์ง่าย ๆ แทนที่จะอภิปรายเฉพาะหนึ่งคำสั่งสำหรับเฉพาะเครื่องหนึ่ง

คำสั่ง TestAndSet( ) สามารถกำหนดไว้ตามที่แสดงโค้ดภาษาซีในภาพที่ 3.10 ลักษณะที่สำคัญคือจะทำงานคำสั่งนี้แบบ Atomically ดังนั้นหากคำสั่ง TestAndSet( ) ที่ทำงานในเวลาเดียวกัน แต่ทำบนซีพียูที่แตกต่างกัน คำสั่งนี้จะถูกรันตามลำดับในบางลำดับแบบสุ่ม หากเครื่องสนับสนุนคำสั่ง TestAndSet( ) แล้วเราสามารถทำการเอาออกร่วมกันโดยมีการแจ้งการล็อคตัวแปร Boolean ในการเริ่มต้นจะเป็นเท็จ โครงสร้างของโพรเซส Pi จะแสดงในภาพที่ 3.11

```
// Shared boolean variable lock, initialized to FALSE
// Solution:

do {
      while (test_and_set(&lock)) ; /* do nothing */
            /* critical section */
      lock = false;
            /* remainder section */
} while (true);
```

ภาพที่ 3.11 แสดงโครงสร้างโค้ดภาษาซีของโพรเซสในรูปแบบ Test-and-Set

3.6.4 Swap Instruction

คำสั่ง Swap( ) จะตรงกันข้ามกับคำสั่ง TestAndSet( ) คำสั่ง Swap( ) จะทำงานในเนื้อหาของคำสองคำ กำหนดไว้ตามที่แสดงในภาพที่ 3.12 เช่นเดียวกับคำสั่ง TestAndSet( ) จะรัน Atomically หากเครื่องสนับสนุนคำสั่ง Swap( ) แล้วการเอาออกทั้งสองฝ่ายโดยมีเงื่อนไขดังนี้ การล็อคตัวแปร Global Boolean จะถูกประกาศและทำให้เป็นเท็จในเริ่มต้น นอกจากนี้แต่ละโพรเซสมีคีย์ของตัวแปร Local Boolean โครงสร้างของโพรเซส Pi จะปรากฏในภาพที่ 3.13

```
Void Swap (boolean *a, boolean *b)  {
            boolean temp = *a;
             *a = *b;
             *b = temp;
      }
```

ภาพที่ 3.12 แสดงฟังก์ชั่น Swap( ) ในโครงสร้างโค้ดภาษาซี

```
do {
      key = TRUE ;
          while (key = = TRUE )
                Swap (&lock, &key) ,
                // critical section
              Lock = FALSE;
                // remainder section

} while  (TRUE) ;
```

ภาพที่ 3.13 แสดงโครงสร้างโค้ดภาษาซีการกีดกันโพรเซสโดยการเรียกใช้ฟังก์ชั่น Swap( )

สำหรับนักออกแบบฮาร์ดแวร์ การใช้กลุ่มคำสั่ง TestAndSet( ) บนเครื่องประมวลผลจำนวนมากไม่เป็นงานที่สำคัญนัก การทำให้เป็นผลสำเร็จถูกอธิบายไว้ในหนังสือสถาปัตยกรรมคอมพิวเตอร์

3.7 โครงสร้างพื้นฐานสำหรับการซินโครไนซ์

โพรเซสที่ทำงานร่วมกับโพรเซสอื่นในการดำเนินการ จำเป็นต้องอาศัยการซินโครไนซ์ที่เหมาะสม เพื่อให้การทำงานถูกต้องตามที่กล่าวมาแล้วนั้น ในการประสานเวลาโพรเซสไม่ใช่เรื่องที่ทำได้ง่าย แม้ว่าจะมีคำสั่งทดสอบและเซต มาเอื้ออำนวยให้ต่อการให้ได้จังหวะกัน แต่ก็ยังไม่สะดวกในการพัฒนาทางด้านนี้ จึงมีโครงสร้างและสิ่งที่อาจจะเกิดขึ้นในการประสานเวลาที่เรียกว่า คำสั่งพื้นฐาน (Primitive) ที่น่าสนใจดังนี้

3.7.1 เซมาฟอร์ (Semaphore)

วิธีการแก้ปัญหาเขตวิกฤตที่กล่าวมาแล้วทั้งหมด ยังคงไม่สะดวกในการใช้งานกับปัญหาที่ซับซ้อนมากขึ้น เราจึงต้องหาวิธีใหม่ เรียกว่า เซมาฟอร์ ที่ไดจสตร้าได้เสนอขึ้นในปี ค.ศ. 1965 ตามนิยาม เซมาฟอร์แทนได้ด้วย S ซึ่งมีค่าเป็นเลขจำนวนเต็ม (Integer) การเรียกใช้งานจะทำได้ผ่านทาง 2 คำสั่ง คือ Signal (แทนด้วย V ย่อมาจากคำว่า Verhogen) และ Wait (แทนด้วย P ย่อมาจากคำว่า Proberen) ดังโครงสร้างตามภาพที่ 3.14 พบว่าคำสั่ง wait(S) เป็นการทดสอบค่าของ S และจะทำการลดค่าลงเมื่อตรวจสอบข้อมูลแล้วพบว่ามีค่ามากกว่า 0 ในทางตรงกันข้ามส่วนของ signal(S) เป็นการทำงานเพื่อเพิ่มค่าให้กับ S

```
// S: wait() and signal()
// Originally called P() and V()

wait (S) {
        while S <= 0
                ; // no-op
          S--;
    }
signal (S) {
```

ภาพที่ 3.14 โครงสร้างของคำสั่ง Signal และ Wait ใน Semaphore

สมมติ Operator ที่จะทำงานกับเซมาฟอร์ 2 ตัว คือ คำสั่ง wait(S) ใช้ตรวจสอบค่า รอจน S มีค่ามากกว่า 0 แล้วจึงลดค่าของ S ลง 1 แล้วทำคำสั่งถัดไป คำสั่ง signal(S) ใช้เพิ่มค่า S ขึ้น 1 แล้วทำคำสั่งถัดไปจะต้องใช้คู่กันเสมอในที่นี้ S คือ เซมาฟอร์ คือ ตัวแปรชนิดจำนวนเต็มที่มีค่าไม่น้อยกว่าศูนย์ คำสั่ง wait(S) จะกั้นโพรเซสไว้หากค่าของ S กลายเป็นศูนย์ คำสั่งทั้งสองนี้เป็นประเภทไม่อาจแบ่งแยกได้ คือไม่สามารถถูกขัดได้ด้วยโพรเซสอื่น

เซมาฟอร์ เป็นเครื่องมือประสานเวลาที่ไม่ต้องการเวลารอคอย เซมาฟอร์ S เป็นตัวแปรจำนวนเต็ม ช่วยลดความซับซ้อนสามารถเข้าถึงได้ผ่านคำสั่งปฏิบัติการมาตรฐานแบบภาวะครบหน่วย (Atomic operation) สองคำสั่ง ได้แก่คำสั่ง Wait และ Signal ความหมายคำสั่งปฏิบัติการแบบภาวะครบหน่วย คือ เมื่อโพรเซสหนึ่งกำลังแก้ไขค่าเซมาฟอร์ จะต้องไม่มีโพรเซสอื่นที่สามารถเข้ามาแก้ไขค่าเซมาฟอร์เดียวกันได้ในเวลาเดียวกัน กรณีของ wait(S) การทดสอบค่าจำนวนเต็มของ S (S <= 0) และการแก้ไขค่า (S--) จะต้องกระทำการโดยปราศจากขัดจังหวะ

เซมาฟอร์ เป็นอัลกอริทึมหนึ่งที่ง่ายต่อการจัดการทรัพยากรแก่โปรเซส โดยเซมาฟอร์นี้จะเปรียบเสมือนการหยิบและใส่ลูกหินในขวดโหล กล่าวคือ จำนวนลูกหินในขวดโหลจะแทนค่าของเซมาฟอร์ในขณะใดขณะหนึ่ง การที่โปรเซสต้องการทรัพยากรก็เปรียบได้กับการหยิบลูกหิน ถ้ามีลูกหินที่ต้องการโปรเซสนั้นก็ทำการหยิบไปลูกหนึ่ง ก็คือโปรเซสนั้นทำคำสั่ง p จากนั้นก็ทำงานของตนต่อไปแต่ถ้าในกรณีที่ไม่มีลูกหินในขวดโหลเลย โปรเซสนั้นต้องรอ (Wait) ให้มีโปรเซสอื่นมาใส่ลูกหินลงไปในขวดโหล ซึ่งก็คือการทำคำสั่ง v ซึ่งเป็นการส่งสัญญาณ (Signal) นั่นเอง ต่อจากนั้นเมื่อมีโปรเซสหนึ่งใส่ลูกหินลงไปแล้วก็ไปทำงานของตนต่อไป และถ้าโปรเซสใดๆ รอลูกหินนั้นอยู่ ก็หยิบไปใช้งานได้ แต่ถ้าไม่มีโปรเซสใดๆ รอหยิบ ก็ไม่มีอะไรเกิดขึ้น ในกรณีที่มีหลายๆ โปรเซสรอการหยิบลูกหินอยู่ ก็จะต้องมีเกณฑ์เพื่อเลือกสรรให้โปรเซสใดโปรเซสหนึ่งหยิบลูกหินอย่างยุติธรรม วิธีที่ง่ายที่สุด เช่น วิธีใครมาก่อนได้ก่อน (First-Come-First-Served)

3.7.2 การใช้งานเซมาฟอร์

สามารถนำเซมาฟอร์ ไปใช้ในการแก้ไขปัญหาส่วนวิกฤตของโพรเซส n ตัว โพรเซส n ตัวจะร่วมใช้เซมาฟอร์ ที่ชื่อ mutex (ใช้แทน Mutual exclusion) เริ่มการทำงานที่ 1 โดยที่แต่ละโพรเซส Pi ให้มีโครงสร้างดังภาพที่ 3.15

```
do {
        wait(mutex);
                critical section
        signal(mutex);
                remainder section
        } while (true);
```

ภาพที่ 3.15 การสร้าง Mutual Exclusion ด้วย Semaphore

เราสามารถใช้เซมาฟอร์ในการแก้ไขปัญหาของการทำงาน ที่ให้โพรเซสทำงานประสานกันที่หลากหลาย เช่น เมื่อพิจารณาโพรเซส 2 ตัวที่ทำงานพร้อมกันให้โพรเซส 1 มีสภาวะการทำงานที่ S1 และโพรเซส 2 มีสภาวะการทำงานที่ S2 สมมติเราต้องการให้ S2 ทำงานได้ก็ต่อเมื่อ S1 ทำงานเสร็จ เราสามารถสร้างวิธีการดังกล่าวได้โดยการใช้โพรเซส 1 และ 2 ใช้เซมาฟอร์ ที่ชื่อ synch ร่วมกัน แล้วเริ่มการ

ทำงานที่ศูนย์ แล้วทำการแทรกเข้าไปในสภาวะการทำงาน S1 ในโพรเซส 1 ในทำนองเดียวกันก็นำไปไว้ที่สภาวะการทำงาน S2 ในโพรเซส 2 นั้นจะทำให้สภาวะ S2 ในโพรเซส 2 ทำงานได้ก็ต่อเมื่อโพรเซส 1 ให้สัญญาณ Signal(synch) เมื่อทำงานสภาวะ S1 เสร็จแล้วเท่านั้น

3.7.3 การสร้างเซมาฟอร์ (Semaphore Implementation)

ข้อเสียของการแก้ไขปัญหาการกีดกันนั่นก็คือ ถ้าหากมีโพรเซสใดกำลังทำงานในส่วนวิกฤตอยู่นั้น แล้วมีโพรเซสอื่นต้องการเข้าทำงานก็จะต้องวนลูปรอเข้าทำงานไปเรื่อย ๆ เรียกว่า “การวนเวียนรอคอย” (Busy waiting) การรอเช่นนี้ทำให้เสียเวลาอยู่ในสภาวะของ Waiting แทนที่จะสามารถทำอย่างอื่นได้ ทำให้ประสิทธิภาพลดลง ย่อมเป็นเหตุให้เวลาของหน่วยประมวลผลกลางเสียไปโดยเปล่าประโยชน์ เวลาเหล่านี้อาจยกให้กระบวนการอื่นได้ใช้ประโยชน์คุ้มค่ากว่า เหตุการณ์เช่นนี้เรียกว่า Spinlock คือ โพรเซสวนเวียนทำงาน (Spin) อยู่ ขณะที่กำลังรอคอยให้เปิดล็อค ในระบบแบบ Multiprocessor จะเกิดข้อดีเนื่องจากไม่ต้องทำ Context Switch ดังนั้นถ้าการ Lock อยู่ในช่วงเวลาที่สั้น ๆ Spinlock ก็จะเป็นประโยชน์อย่างยิ่ง

เราสามารถปรับปรุงนิยามของคำสั่ง wait() และ signal() ใหม่ เพื่อให้ไม่มีปัญหาการวนเวียนรอคอยเมื่อโพรเซสทำคำสั่ง wait() และพบว่าค่าของเซมาฟอร์น้อยกว่าหรือเท่ากับศูนย์ โพรเซสจะต้องรอคอย โดยการหยุดชั่วขณะ (Block) แทนที่จะวนเวียนรอคอย การหยุดชั่วขณะนี้โพรเซสจะเข้าไปรอในแถวคอย (Waiting queue) ที่เกี่ยวข้องกับเซมาฟอร์นั้น ๆ และเปลี่ยนสถานะเป็นสถานะรอคอย (Waiting state) การควบคุมจะถูกย้ายกลับไปที่ตัวจัดตารางการทำงานของหน่วยประมวลผลกลาง เพื่อจัดให้โพรเซสอื่นทำงานต่อไป

โพรเซสที่หยุดชั่วขณะโดยรอเซมาฟอร์ S นี้ จะสามารถดำเนินต่อได้ เมื่อมีโพรเซสอื่นทำคำสั่ง signal() กับเซมาฟอร์ S ภายในคำสั่ง signal จะมีการปลุก (Wakeup) ให้กระบวนการที่หยุดชั่วขณะกลับดำเนินการต่อได้ โดยเปลี่ยนสถานะของกระบวนการนั้นเป็นสถานะพร้อม (Ready state) แล้วย้ายไปต่อแถวพร้อม

การสร้างเซมาฟอร์ตามนิยามใหม่นี้ กำหนดให้เซมาฟอร์เป็นตัวแปรประเภท Structure ในภาษาซี

```
typedef struct {
        int value;
        struct process *L;
 } semaphore;
```

ตัวแปรเซมาฟอร์จะประกอบด้วย ตัวเลขจำนวนเต็ม (ตัวแปร value) และแถวคอยของโพรเซส (ตัวแปร L) เมื่อโพรเซสหนึ่งต้องคอยเซมาฟอร์นี้ มันก็จะถูกใส่ชื่อลงในแถวคอย L การใช้คำสั่ง signal จะดึงโพรเซสออกจากหัวแถวคอยนี้ และปลุกโพรเซสที่ได้ให้กลับไปทำงานต่อ เราจึงแก้ไขโปรแกรมคำสั่งเป็นดังนี้

```
void wait(semaphore S) {
        S.value - -;
            if (S.value < 0) {
                add this process to S.L (list);
                block();
              }
    }
void signal(semaphore S) {
        S.value ++;
            if (S.value <= 0) {
              remove a process P from S.L (list);
              wakeup(P);
            }
    }
```

ภาพที่ 3.16 แสดงตัวอย่าง Semaphore เป็นโครงสร้างโค้ดในภาษาซี

คำสั่ง block จะทำให้โพรเซสที่ทำคำสั่งนี้หยุดชั่วขณะ คำสั่ง wakeup(P) จะปลุกโพรเซส P ให้กลับทำงานต่อ จะสังเกตได้ว่า นิยามดั้งเดิมของเซมาฟอร์ซึ่งมีการวนเวียนรอคอยนั้น ค่าของเซมาฟอร์จะไม่ติดลบ แต่ในนิยามใหม่นี้ค่าของเซมาฟอร์มีค่าเป็นลบได้ ค่าที่เป็นลบแสดงถึงจำนวนกระบวนการที่รอคอยเซมาฟอร์ตัวนั้น ๆ ด้วย

เราต้องไม่ให้มีโพรเซสสองตัวทำคำสั่ง wait และ signal บนเซมาฟอร์ตัวเดียวกันในเวลาเดียวกัน ซึ่งกรณีนี้ก็คือ ปัญหาเขตวิกฤตแบบหนึ่งนั่นเอง โดยสามารถเลือกวิธีจัดการได้ 2 วิธี คือ

1. ถ้าเป็นระบบที่มีหน่วยประมวลผลเพียงตัวเดียว เราก็เพียงแต่ห้ามขัดจังหวะขณะที่กำลังทำงานคำสั่ง wait และ signal ดังนั้นโพรเซสอื่นจะไม่สามารถมาขัดจังหวะการทำงานได้

2. ถ้าเป็นระบบที่มีหน่วยประมวลผลหลายตัว การห้ามขัดจังหวะจะไม่เพียงพอ เพราะโพรเซสอื่นอาจทำงานอยู่ในหน่วยประมวลผลคนละตัวกับโพรเซสที่ถูกห้ามขัดจังหวะ ดังนั้นอาจจัดการโดยใช้ขั้นตอนวิธีการแก้ไขส่วนวิกฤต แล้วเอาคำสั่ง wait และ signal ใส่ไว้ในเขตวิกฤตก็จะสามารถป้องกันได้

3.7.4 เซมาฟอร์แบบทวิภาค (Binary Semaphore)

โครงสร้างของเซมาฟอร์ ที่กล่าวมาข้างต้นเป็นลักษณะของเซมาฟอร์แบบนับ (Counting semaphore) ส่วนในเซมาฟอร์แบบทวิภาคจะกำหนดให้ใช้ตัวแปรจำนวนเต็มได้ 2 ค่าคือ 0 และ 1 เท่านั้น ทำให้สามารถสร้างได้ง่ายกว่าเซมาฟอร์แบบนับ ซึ่งสามารถสร้างเซมาฟอร์แบบนับให้อยู่ในรูปแบบทวิภาคได้ โดยการใช้โครงสร้างตามภาพที่ 3.17 เริ่มด้วยค่า S1, S3 =1 และ S2 = 0 แล้วใช้งานตามภาพที่ 3.18

```
Var     S1: binary-semaphore;
        S2: binary-semaphore;
        S3: binary-semaphore;
        C: integer;
```

ภาพที่ 3.17 โครงสร้างของการใช้ Semaphore แบบทวิภาค

- *wait* operation

```
wait(S3);
wait(S1);
C := C − 1;
if C < 0
then begin
        signal(S1);
        wait(S2);
     end
else signal(S1);
signal(S3);
```

- *signal* operation

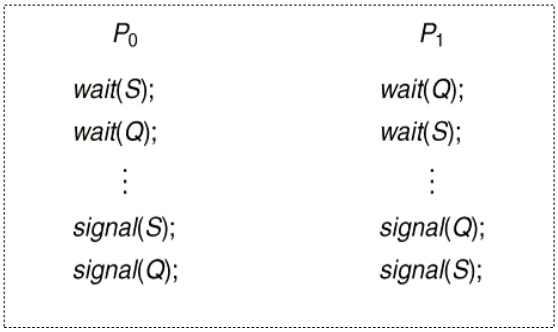
```
wait(S1);
C := C + 1;
if C ≤ 0 then signal(S2);
signal(S1);
```

ภาพที่ 3.18 รูปแบบของการใช้งาน Semaphore แบบทวิภาค

การทำงานคำสั่ง wait ก่อนที่จะมีการลดค่าของเซมาฟอร์ ต้องการทำ wait(S1) ก่อนลดค่า แล้วก็ตรวจสอบว่าค่าที่ลดน้อยว่า 0 หรือไม่ ถ้าน้อยกว่าก็จะมีการปลด lock S1 ที่รออยู่ แล้วก็ lock S2 ต่อ แต่ถ้าค่าของ C มากกว่าหรือเท่ากับ 0 ก็จะปลด lock S1 อย่างเดียว ส่วนคำสั่งของ Signal จะมีการเพิ่มค่า C เมื่อมีการ wait(S1) ไปแล้ว ซึ่งถ้าค่า C ที่เพิ่มน้อยกว่าหรือเท่ากับ 0 ก็จะส่งสัญญาณไปปลด lock S2 แล้วก็จึงปลด lock S1 การทำงานของสภาวะ S1 แทนโพรเซส 1 ส่วน S2 แทนโพรเซส 0 นั่นเอง ดังนั้นทั้งหมดจึงเป็นการเลือกการเข้าทำงานส่วนวิกฤตของสองโพรเซส

3.8 การติดตายและอดตาย

การสร้างเซมาฟอร์ด้วยการให้โพรเซสที่รอการทำงานไปอยู่ในคิว อาจส่งผลให้เกิดเหตุการณ์ที่สองโพรเซสหรือมากกว่านั้นรอแบบไม่มีที่สิ้นสุด เนื่องจากโพรเซสไปรอการทำงานของโพรเซสที่ยังรอการทำงานด้วยเช่นกัน ลักษณะเช่นนี้เรียกว่าการติดตาย (Deadlock)



ภาพที่ 3.19 สถานการณ์ในการใช้งาน 2 โพรเซสแบบ Semaphore และเกิด Deadlock

สถานการณ์ติดตายสามารถอธิบายได้ดังภาพที่ 3.19 ซึ่งมีโพรเซสทำงานอยู่ 2 โพรเซสพร้อมกัน คือ P0 และ P1 โดยในแต่ละโพรเซสก็มีการเข้าใช้เซมาฟอร์ 2 ตัวคือ S และ Q โดยให้ค่าเป็น 1 สมมติว่าโพรเซส P0 ทำงาน wait(s) พร้อมกับที่โพรเซส P1 ทำงาน wait(Q) เมื่อโพรเซส P0 ทำงาน wait(Q) มันจะต้องรอจนกว่าโพรเซส P1 ทำการ signal(Q) ในทำนองเดียวกันนั้นเมื่อโพรเซส P1 ทำงาน wait(S) มันก็ต้องรอจนกว่าโพรเซส P0 ทำการ signal(S) ให้ พบว่าไม่สามารถมีโพรเซสใดทำงานได้ ลักษณะการติดตายจึงเกิดขึ้น หากในการสร้างเซมาฟอร์โดยให้โพรเซสการรอจัดเก็บไว้ในคิวที่มีโครงสร้างแบบ LIFO หรือ เข้าหลังออกก่อน ก็อาจจะให้เกิดปัญหาของการติดตายและอดตายของโพรเซสที่จัดเก็บอยู่ด้านใน

3.9 ปัญหาพื้นฐานของการประสานเวลา

ปัญหาการทำงานของโพรเซส ปัญหาที่เกิดขึ้นกับระบบปฏิบัติการเป็นปัญหาที่น่าสนใจและถกเถียงกันอย่างกว้างขวาง ตลอดจนมีการวิเคราะห์โดยการใช้วิธีการด้านประสานเวลา ในที่นี้จะนำปัญหาพื้นฐานของการประสานเวลาที่เป็นที่กล่าวถึงหรือเป็นต้นแบบดังนี้

3.9.1 ปัญหาที่พักข้อมูลขนาดจำกัด (Bounded-Buffer Problem)

บางทีเรียกว่าปัญหาผู้ผลิต-ผู้บริโภค (Producer/Consumer problem) เป็นปัญหาที่นิยมใช้ในการทดสอบประสิทธิภาพของเทคนิคการประสานเวลา มีผู้ผลิตหนึ่งคนหรือมากกว่าทำการผลิตข้อมูลบางชนิด (ระเบียน, อักขระ) และใส่ลงในบัฟเฟอร์ และมีผู้บริโภคเพียงคนเดียวเท่านั้นที่สามารถหยิบข้อมูลนั้นออกจากบัฟเฟอร์ในขณะใดขณะหนึ่ง ระบบจะต้องกำหนดเงื่อนไขเพื่อป้องกันไม่ให้เกิดการทับซ้อนกันในการปฏิบัติการบนบัฟเฟอร์นั้น หมายความว่า ณ ขณะใดขณะหนึ่งจะมี (ผู้ผลิตหรือผู้บริโภค) เพียงคนเดียวเท่านั้นที่สามารถเข้าถึงบัฟเฟอร์ได้

```
//itemCount is an integer
int itemCount = 0;
```

```
procedure producer() {
    while (true) {
        item = produceItem();

        if (itemCount == BUFFER_SIZE) {
            sleep();
        }
         putItemIntoBuffer(item);
        itemCount = itemCount + 1;
         if (itemCount == 1) {
            wakeup(consumer);
        }
    }
}
```

```
procedure consumer() {
    while (true) {

        if (itemCount == 0) {
            sleep();
        }
         item = removeItemFromBuffer();
        itemCount = itemCount - 1;
         if (itemCount == BUFFER_SIZE - 1) {
            wakeup(producer);
        }
         consumeItem(item);
    }
}
```

ภาพที่ 3.20 ตัวอย่างการใช้อัลกอริทึม Sleep and Wakeup ในการแก้ปัญหา Bounded-Buffer Problem

เนื่องจากมีสองโพรเซสใช้บัฟเฟอร์ร่วมกันโดยที่โพรเซสของผู้ผลิต (Producer process) เป็นผู้ส่ง Information ให้กับบัฟเฟอร์ และโพรเซสของผู้บริโภค (Consumer process) เป็นผู้เรียกใช้ Information นั้น และปัญหาจะเกิดเมื่อโพรเซสต้องการเพิ่ม Information ในบัฟเฟอร์ที่เต็ม (ไม่สามารถใส่เพิ่มได้) วิธีการแก้ปัญหานี้ทางหนึ่งคือโพรเซสของผู้ผลิตต้อง Sleep จนกว่าบัฟเฟอร์จะมีที่ว่าง (wakeup เมื่อบัฟเฟอร์มีที่ว่างให้เพิ่มข้อมูลได้อีก) และในทำนองเดียวกันถ้าบัฟเฟอร์ว่างโพรเซสของผู้บริโภคต้อง Sleep ว่างจนกว่าจะมีข้อมูลเพิ่มเติม (Wakeup เมื่อมีข้อมูลเข้ามาสู่บัฟเฟอร์)

3.9.2 ปัญหาผู้อ่านและผู้เขียน (The Readers/Writers Problem)

ข้อมูลในระบบที่มีการทำงานของโพรเซสพร้อมกันหลายตัวจะถูกร่วมกันใช้งาน บางโพรเซสต้องการอ่านข้อมูล ในอีกบางโพรเซสต้องการเขียนข้อมูล โดยที่ความแตกต่างของทั้งสองส่วนคือ Readers จะเป็นโพรเซสที่ต้องการจะอ่านข้อมูลเพียงอย่างเดียว ส่วนโพรเซส Writer จะต้องการเพียงแค่การเขียนข้อมูลเท่านั้น สังเกตว่าหากมีหลาย ๆ โพรเซสทำการอ่านข้อมูลพร้อม ๆ กันจะไม่มีปัญหาใดเกิดขึ้น แต่ถ้ามีโพรเซสที่ต้องการเขียนข้อมูลพร้อม ๆ กันจะเกิดปัญหาขึ้นทันที

ปัญหานี้คือ มีพื้นที่ข้อมูลซึ่งใช้ร่วมกันได้ระหว่างกระบวนการทั้งหมด มีโพรเซสจำนวนมากที่ต้องการอ่านพื้นที่ดังกล่าวเพียงอย่างเดียว และมีโพรเซสที่ต้องการเขียนบนพื้นที่ดังกล่าวเพียงโพรเซสเดียวในเวลาเดียวกัน จึงมีการนำเอาเซมาฟอร์ช่วยในการแก้ไขปัญหาเหล่านี้ โดยให้มีโครงสร้างของการใช้ตัวแปรร่วมกันดังภาพที่ 3.21

การจัดการจะต้องเข้ากันได้กับเงื่อนไขดังนี้ คือ 1) ผู้อ่านสามารถอ่านแฟ้มได้พร้อมกันหลายคน 2) มีผู้เขียนเพียงคนเดียวเท่านั้นที่สามารถเขียนแฟ้มได้ ณ เวลาใดเวลาหนึ่ง 3) ถ้ามีผู้เขียนรอที่จะเขียนแฟ้ม จะต้องไม่มีผู้อ่านคนใดสามารถอ่านแฟ้มนั้นได้

จากข้อกำหนดข้างต้นอาจนำไปสู่ปัญหาภาวะงูกินหาง กรณีแรกผู้เขียนอาจต้องเป็นฝ่ายรออย่างไม่รู้จบ กรณีหลังผู้อ่านอาจต้องเป็นฝ่ายรออย่างไม่รู้จบ

การ Shared Data
- Data set
- Semaphore mutex initialized to 1.
- Semaphore wrt initialized to 1.
- Integer readcount initialized to 0.

ภาพที่ 3.21 โครงสร้างตัวแปรที่ใช้งานใน Semaphore

พบว่ามีการใช้เซมาฟอร์ 2 ตัวคือ mutex และ wrt โดยให้มีการเริ่มค่าที่ 1 ส่วนตัวแปร readcount จะเป็นจำนวนเต็มที่เริ่มการทำงานที่ 0 เซมาฟอร์ wrt จะถูกใช้งานทั้งโพรเซสอ่านและเขียน ส่วนเซมาฟอร์ mutex ใช้ในการแก้ไขปัญหาด้วย Mutual Exclusion เมื่อตัวแปร readcount ถูกเปลี่ยนแปลงค่าโดยที่ตัวแปรนี้ใช้เก็บข้อมูลจำนวนโพรเซสที่กำลังทำการอ่านข้อมูลอยู่ เซมาฟอร์ wrt ใช้เพื่อทำ Mutual ในส่วนของการเขียนซึ่งถูกใช้งานโดยผู้อ่านตัวแรกหรือตัวสุดท้ายที่เข้าทำงานส่วนวิกฤต แต่จะไม่ถูกใช้ในกรณีที่ผู้อ่านตัวนั้นกำลังรอจะเข้าทำงานส่วนวิกฤต แต่มีผู้อ่านตัวอื่นค้างอยู่ในส่วนวิกฤตนั้น โค้ดของโพรเซสผู้อ่านและผู้เขียนแสดงไว้ดังภาพที่ 3.22 และ 3.23

```
do {
        wait (wrt) ;

          //    writing is performed
        signal (wrt) ;
    } while (true)
```

ภาพที่ 3.22 รูปแบบของโพรเซสผู้เขียน

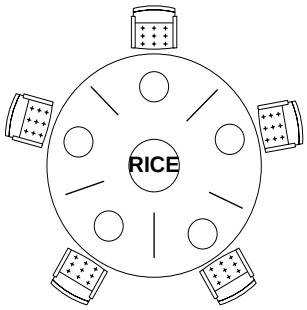
พบว่าเมื่อผู้เขียนกำลังทำงานในส่วนวิกฤต และมีโพรเซสรออยู่เป็นจำนวน n แล้ว เมื่อผู้อ่านถูกนำเข้าไปในคิวด้วย wrt ทำให้จำนวนโพรเซสที่รอลดลงเหลือ n-1 ซึ่งจะเข้าคิวด้วยเซมาฟอร์ mutex เมื่อสังเกตจะพบว่าเมื่อผู้เขียนทำ signal(wrt) อาจเกิดกรณีที่ให้โพรเซสผู้อ่านที่รออยู่เข้าทำงาน หรือจะให้สิทธิแก่โพรเซสผู้เขียนที่เหลือเพียงตัวเดียวทำงาน ทั้งนี้ผู้จัดตารางจะเป็นผู้กำหนดและจัดการ

```
do {
    wait (mutex) ;
    readcount ++ ;

    if (readercount == 1)  wait (wrt) ;
    signal (mutex)

                // reading is performed
    wait (mutex) ;
    readcount  - - ;

    if redacount  == 0)  signal (wrt) ;
    signal (mutex) ;

} while (true)
```

ภาพที่ 3.23 รูปแบบของโพรเซสผู้อ่าน

3.9.3 ปัญหาอาหารเย็นของนักปราชญ์ (The Dining Philosophers Problem)

ปัญหาอาหารเย็นของนักปราชญ์ โดย Edsger Dijkstra (ปี 1995) เป็นปัญหาที่ต้องการให้มีการจัดการเรื่องการใช้ทรัพยากรที่มีอยู่อย่างจำกัดร่วมกันระหว่างหลายโพรเซส ปัญหาการติดตายจะเกิดเมื่อนักปราชญ์ทั้งห้าหยิบตะเกียบคนละข้างพร้อมกัน ปัญหาการอดตาย การที่นักปราชญ์ทางด้านซ้ายและด้านขวาของนักปราชญ์ผู้โชคร้ายสลับกันทานอาหาร



ภาพที่ 3.24 ปัญหาอาหารเย็นของนักปราชญ์

ปัญหาอาหารเย็นของนักปราชญ์นี้นับเป็นปัญหาแม่แบบปัญหาหนึ่ง เพราะสามารถใช้แทนปัญหาต่าง ๆ ในระบบการทำงานแบบขนานได้มากมาย การแก้ปัญหานี้ทำได้โดยการกำหนดตัวแปรร่วมดังนี้

1. กำหนดเซมาฟอร์เป็นอาร์เรย์ชื่อ chopstick[5];
2. ให้ตะเกียบ (อาร์เรย์ chopstick) เป็นเซมาฟอร์โดยนักปราชญ์
3. หยิบตะเกียบด้วยคำสั่ง wait
4. วางตะเกียบด้วยคำสั่ง signal
5. ค่าเริ่มต้นของตะเกียบ (chopstick) = 1

```
do {
        wait(chopstick[i]);
        wait(chopstick[(i+1) % 5]);
                ...
                eat
                ...
        signal(chopstick[i]);
        signal(chopstick[(i+1) % 5]);
                ...
                think
                ...
} while (true);
```

ภาพที่ 3.25 ตัวอย่างโครงสร้างภาษาในการแก้ปัญหาปัญหาอาหารเย็นของนักปราชญ์

วิธีนี้อาจป้องกันไม่ให้นักปราชญ์ที่นั่งติดกันกินพร้อมกันได้ แต่อาจเกิดปัญหาวงจรอับขึ้นในระบบ สมมติว่านักปราชญ์ทั้ง 5 คนหิวพร้อมกัน แล้วหยิบตะเกียบข้างซ้ายของเขาเกือบพร้อมกันหมด จากนั้นแต่ละคนจะพยายามหยิบตะเกียบข้างขวา ซึ่งจะต้องรอกันเองตลอดไป อาจแก้ปัญหาวงจรอับนี้ได้ดังนี้

1. ให้นักปราชญ์นั่งในโต๊ะได้ ไม่เกิน 4 คน
2. ให้นักปราชญ์หยิบตะเกียบได้ ก็ต่อเมื่อตะเกียบทั้งสองข้าง (ทั้งซ้ายและขวา) ว่างทั้งคู่ (ต้องทำในเขตวิกฤตด้วย)
3. ใช้วิธีแก้ปัญหาแบบอสมมาตร โดยให้นักปราชญ์คนที่เลขคี่ ให้หยิบตะเกียบข้างซ้ายก่อน ถ้าได้แล้วจึงหยิบตะเกียบข้างขวา
4. นักปราชญ์คนที่เลขคู่ หยิบตะเกียบข้างขวาก่อนแล้วจึงหยิบตะเกียบข้างซ้าย

วิธีการนี้จะรับประกันว่าไม่มีการแย่งชิงที่เป็นสาเหตุให้นักปราชญ์คนหนึ่งเข้าสู่ภาวะติดตาย แต่วิธีนี้ไม่ได้ป้องกันสภาวะการรออย่างไม่รู้จบ

3.10 สรุป

โพรเซสมีความจำเป็นในการเข้าใช้ทรัพยากรและต้องทำงานร่วมกันกับโพรเซสอื่น ดังนั้น โพรเซสหนึ่งอาจจะได้รับผลกระทบจากโพรเซสอื่น การควบคุมให้โพรเซสสองโพรเซสทำงานร่วมกันได้ ที่เรียกว่า การประสานเวลาของโพรเซส หรืออาจเรียกว่า การซินโครไนซ์โพรเซส การซินโครไนซ์โพรเซสเป็นเรื่องที่ยุ่งยากและมีความซับซ้อน และเป็นหน้าที่ของระบบปฏิบัติการที่จะต้องจัดการ เช่น การแก้ปัญหาจากการที่โพรเซสเข้าไปใช้ทรัพยากรใดพร้อมกันที่เรียกว่า สภาวะการแย่งชิง ระบบการปฏิบัติการจะต้องแน่ใจว่า มีเพียงโพรเซสเดียวเท่านั้น ที่ได้รับการเข้าใช้ทรัพยากรใดในช่วงเวลาใดเวลาหนึ่ง ดังนั้นระบบการปฏิบัติการจะต้องป้องกันการเข้าส่วนวิกฤตให้ดี จำเป็นต้องเป็นไปตามเงื่อนไขของการแก้ไขปัญหาในการเข้าสู่ส่วนวิกฤต และการตรวจสอบเงื่อนไขที่กำหนด

การประมวลผลพร้อมกันของโพรเซส เป็นวิธีการทางซอฟต์แวร์ที่เข้ามาช่วยควบคุมการทำงานของระบบให้มีการประมวลผลพร้อมกัน โดยมีคุณสมบัติของการไม่เกิดร่วม มีอัลกอริทึมที่น่าสนใจคือ อัลกอริทึมของเดกเกอร์ และอัลกอริทึมของปีเตอร์สัน การประมวลผลพร้อมกันและมีคุณสมบัติการไม่เกิดร่วม มีวิธีการทางฮาร์ดแวร์สามารถทำได้หลายวิธี เช่นการปิดกั้นไม่ให้โพรเซสใดเข้าใช้ในส่วนวิกฤตในขณะที่มีโปสเซสอื่นใช้งานอยู่ หรือการตรวจสอบส่วนของวิกฤตว่ามีโพรเซสทำงานอยู่หรือไม่ เพื่อเป็นการป้องกันไม่ให้โพรเซสอื่นเข้าไปทำงานในส่วนวิกฤติพร้อมกันได้ ทำให้ระบบสามารถมีโพรเซสจำนวนมากทำงานร่วมกัน โดยมีคุณสมบัติการไม่เกิดร่วม

วิธีการแก้ปัญหาเขตวิกฤต ที่กล่าวมาแล้วทั้งหมด ยังคงไม่สะดวกในการใช้งานกับปัญหาที่ซับซ้อนมากขึ้น ทั้งนี้ได้นำวิธีการเซมาฟอร์เข้ามาช่วยแก้ไขกับปัญหาที่ซับซ้อนมากขึ้น เซมาฟอร์ คือ ตัวแปรชนิด จำนวนเต็ม ช่วยลดความซับซ้อน สามารถเข้าถึงได้โดยผ่านฟังก์ชั่นมาตรฐานได้แก่ฟังก์ชั่น wait() และ signal() ซึ่งเป็นคำสั่งปฏิบัติการแบบภาวะครบหน่วยกล่าวคือ เมื่อโพรเซสหนึ่งกำลังแก้ไขค่าเซมาฟอร์จะต้องไม่มีโพรเซสอื่นที่สามารถแก้ไขค่าเซมาฟอร์เดียวกันได้ในเวลาเดียวกัน

นอกจากนี้ยังสามารถใช้เซมาฟอร์แก้ปัญหาการประสานงาน (Synchronization Problem) แบบต่าง ๆ ทั้งนี้สามารถที่จะนำเซมาฟอร์นี้ นำไปใช้แก้ปัญหาพื้นฐานที่ใช้ทดสอบวิธีการแก้ปัญหาการประสานเวลาให้ตรงกันได้ เช่น ปัญหาการ Bounded-Buffer, ปัญหาการ Readers – Writers และปัญหาการทำ Dining-Philosophers ที่มีความสำคัญหลัก เนื่องจากปัญหาเหล่านี้คือตัวอย่างปัญหาหลักพื้นฐานที่ถูกใช้ในการทดสอบเกือบจะทุกโครงการเลยทีเดียว

แบบฝึกหัดท้ายบทที่ 3

1. ปัญหา Concurrent Problem คืออะไร จงอธิบายพร้อมยกตัวอย่าง รวมทั้งอธิบายถึงการแก้ปัญหาสภาวะเงื่อนไขการแย่งชิง

2. ส่วนวิกฤติคืออะไร จงอธิบายมาให้เข้าใจ

3. อัลกอริทึมของเดกเกอร์กับอัลกอริทึมของปีเตอร์สัน แตกต่างกันอย่างไร อธิบายมาให้เข้าใจ

4. จงอธิบายความหมายของคำว่า Busy waiting และมีกี่ประเภทในระบบปฏิบัติการ

5. จงอธิบายว่าทำไม Spinlocks จึงไม่เหมาะกับระบบ Single-Processor แต่กลับถูกใช้กับระบบ Multiprocessor

6. ให้แสดงว่าถ้าเซมาฟอร์ wait() และ signal() ยังไม่ได้ถูกดำเนินการ มีโอกาสที่ลักษณะที่มีการกีดกันอาจจะไม่สามารถทำตามกฎ

7. สมมติว่าโปรแกรมถูกแก้ให้ใช้ Shared binary semaphore S:

โพรเซส A	โพรเซส B
int Y;	int Z;
wait(S);	wait(S);
A1: Y = X*2;	B1: Z = X+1;
A2: X = Y;	B2: Z = X;
signal(S);	signal(S);

ค่า T ถูกตั้งค่าเป็น 1 ก่อนที่ทั้งสองโพรเซสนี้จะเริ่มทำงาน และเมื่อค่า X ถูกตั้งค่าเป็น 5 เมื่อเป็นแบบนี้ X สามารถมีค่าเป็นเท่าไรได้บ้างหลังสองโพรเซสนี้ทำงานเสร็จ?

8. จงอธิบายว่าทำไมการอินเตอร์รัพไม่เหมาะกับการนำไปใช้ใน Synchronization Primitives ในระบบ Multiprocessor

9. จงอธิบาย Circular Buffer ทำงานอย่างไรอธิบายการทำงานและยกตัวอย่าง และเป็นการแก้ปัญหาในเรื่องใด

10. จงอธิบายอัลกอริทึ่มในการแก้ปัญหาอาหารเย็นของนักปราชญ์ว่าสามารถแก้ไขได้อย่างไร

บรรณานุกรมประจำบทที่ 3

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

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

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

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

Andrew S. Tanenbaum, Herbert Bos. (2004). **Modern Operating Systems. 4th ed.** Prentice Hall, Pearson Education International.

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

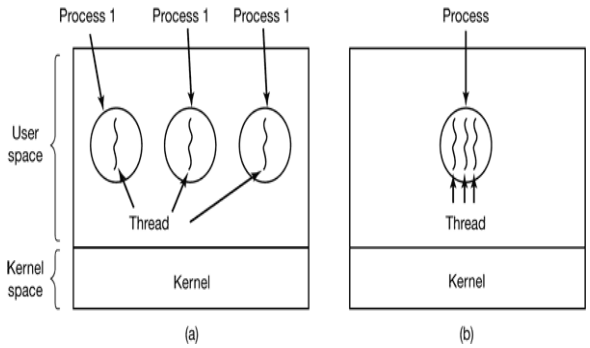
บทที่ 4
การจัดการเธรด

4.1 เธรด

เธรด คือ หน่วยการทำงานย่อยที่อยู่ในโพรเซสที่มีการแบ่งปันทรัพยากรต่าง ๆ ในโพรเซสนั้น ๆ โดยปกติโพรเซสที่มีเพียง 1 เธรดจะถูกเรียกว่า Single thread หรือเรียกอีกชื่อว่า Heavy Weight Process ซึ่งมักพบในระบบปฏิบัติการรุ่นเก่า แต่ถ้า 1 โพรเซสมีเธรดหลายเธรดจะเรียกว่า Light Weight Process (LWP) หรือ Multithread ซึ่งพบได้ในระบบปฏิบัติการรุ่นใหม่ที่ใช้กันในปัจจุบันทั่วไป และ Multithread ก็เป็นที่นิยมมากกว่า Single thread

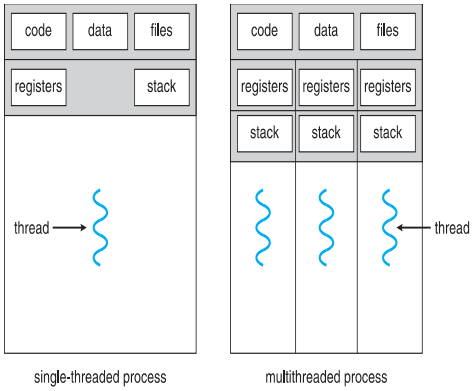
เหตุที่ต้องมีเธรดคือ การเรียกใช้หน่วยประมวลผลให้เกิดประโยชน์สูงสุด เธรดทำให้การทำงานของโปรแกรมง่ายมีประสิทธิภาพและมีประโยชน์ต่อระบบที่มีหลายหน่วยประมวลผล (Multiprocessor) หรือมีแกนประมวลผลหลายแกน (Multicore) เพราะสามารถเรียกใช้เธรดหลาย ๆ ตัวได้พร้อม ๆ กัน โดยเธรดแต่ละตัวของโพรเซสเดียวกันจะทำงานแตกต่างกัน แต่มีความเกี่ยวข้องกันบางอย่างและต้องทำงานอยู่ภายใต้สภาพแวดล้อมเดียวกัน

โพรเซสที่กล่าวผ่านมาเป็นการให้โปรแกรมทำงานในลักษณะมีการควบคุมเพียงหนึ่งเธรด (แต่ละโพรเซสจะประกอบด้วยเธรดเพียงเธรดเดียวเท่านั้น) แต่ระบบปฏิบัติการสมัยใหม่ในแต่ละโพรเซสสามารถมีได้หลายเธรด อาจจะกล่าวได้ว่าเธรดก็คือส่วนประกอบย่อยของโพรเซสนั่นเอง จนบางครั้งอาจเรียกเธรดว่า “Light Weight Process” ในภาพที่ 4.1 (a) คุณจะเห็นโพรเซส 3 โพรเซส แต่ละโพรเซสจะมีเลขที่ตำแหน่งเป็นของตนเอง และควบคุมเพียง 1 เท่านั้น ในภาพที่ 4.1 (b) จะเห็นว่าในแต่ละโพรเซสจะควบคุมสามเธรด โดยใช้เลขที่ตำแหน่งเดียวกันอยู่



ภาพที่ 4.1 (a) 3 โพรเซสแต่ละโพรเซสจะมี 1 Thread (b) 1 โพรเซสที่มี 3 Thread
ที่มา: Andrew S. Tanenbaum. Modern Operating System.
http://lovingod.host.sk/tanenbaum/Processes-and-threads.html

เธรดเป็นหน่วยพื้นฐานของการจัดสรรการใช้ประโยชน์ของซีพียู ภายในโพรเซสจะประกอบด้วยเธรดจะมีการแชร์โค้ด ข้อมูล และทรัพยากร เช่น ไฟล์ อุปกรณ์ต่าง ๆ เป็นต้น โพรเซสดั้งเดิม (ที่เรียกว่า Heavy weight) ที่มีการควบคุมเพียง 1 เธรด แสดงว่าทำงานได้ 1 งาน แต่ถ้าโพรเซสมีหลายเธรด (อาจเรียกว่า Multithread) จะทำงานได้หลายงานในเวลาเดียวกัน ภาพที่ 4.2 แสดงส่วนประกอบของโพรเซสที่เป็น Single thread และ Multithread



ภาพที่ 4.2 โพรเซสที่เป็น Single-threaded และ Multithread
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.164)

องค์ประกอบภายในเธรดประกอบด้วย

1. Threads ID หมายเลขเธรดที่อยู่ในโพรเซส
2. Counter ตัวนับเพื่อติดตามคำสั่งที่จะถูกดำเนินการเป็นลำดับถัดไป
3. Register หน่วยความจำเก็บค่าตัวแปรที่ทำงานอยู่ปัจจุบัน
4. Stack เก็บประวัติการทำงาน

4.2 ตัวอย่างการใช้เธรด

ซอฟต์แวร์ปัจจุบันที่รันกับเครื่องพีซีสมัยใหม่มีการออกแบบให้เป็น Multithread โดยแยกออกเป็นโพรเซสที่ควบคุมหลาย ๆ เธรด เช่น โปรแกรมเว็บบราวเซอร์ที่มีเธรดหนึ่งในการแสดงรูปภาพหรือเขียนข้อความในขณะที่อีกเธรดหนึ่งกำลังดึงข้อมูลจากเน็ตเวิร์ค หรือในโปรแกรมเวิร์ดโพรเซสเซอร์ที่มีหลายเธรด โดยที่เธรดหนึ่งกำลังแสดงภาพกราฟฟิก เธรดที่สองกำลังรอรับคำสั่งจากคีย์บอร์ดจากผู้ใช้ในขณะที่เธรดที่สามกำลังตรวจสอบคำสะกดและไวยากรณ์ในลักษณะทำงานอยู่เบื้องหลัง เป็นต้น

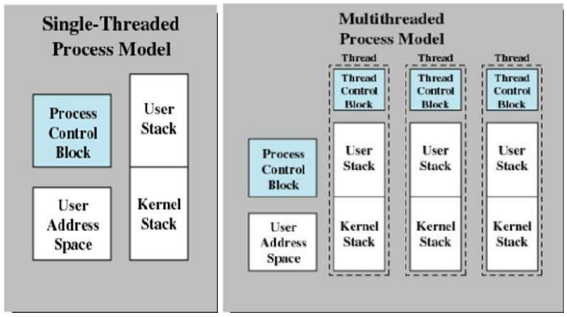
ในสถานการณ์บางอย่างการทำงานแบบ Single อาจจะต้องการทำงานพร้อมกันหลาย ๆ งาน เช่น web server ยอมรับสิ่งที่เครื่องลูกข่ายต้องการพวก Web page image sound ถ้าเครื่องให้บริการมีระบบการทำงานแบบ single เครื่องลูกข่ายนับพัน ๆ เครื่องที่ติดต่อเข้ามาก็จะได้รับการให้บริการเพียงเครื่องเดียวเท่านั้น วิธีหนึ่งคือให้เซริฟเวอร์เรียกใช้งานโพรเซสขึ้นมาหนึ่งโพรเซสและรอรับการร้องขอเมื่อได้รับแล้ว จะสร้างโพรเซสแยกออกมาเพื่อให้บริการในทุกการร้องขอที่ขอมา ซึ่งจริง ๆ แล้วการสร้างโพรเซสในลักษณะนี้เป็นวิธีแบบธรรมดาที่เคยใช้กันก่อนที่จะมีระบบเธรดขึ้นมาอีก การสร้างโพรเซสต้องใช้

เวลาในการสร้างมากและต้องละเอียดถี่ถ้วน ดังที่แสดงให้ดูในบทก่อนหน้า ถ้ามีโพรเซสใหม่ถูกสร้างขึ้นจะมีการทำงานเหมือนกับโพรเซสเดิมที่มีอยู่แล้ว และทำให้การทำงานแบบนี้ไม่เกิด Overhead

4.3 ความแตกตางระหวางโพรเซสกับเธรด

ความหมายโดยทั่วไปของเธรด คือ "Light weight process" ซึ่งหมายถึงโพรเซสที่ใชทรัพยากรอย่างพอเพียง ภาพที่ 4.3 แสดงถึงความแตกตางระหวางโพรเซสกับเธรด ภาพที่ 4.3 ทางดาน ซายแสดงถึงโครงสร้างของโพรเซส ที่มีกับเธรดอยูหนึ่งตัว (Single Threaded Process model) โดยมี PCB มีเนื้อที่ใชสอยสำหรับผู้ใช้ มี Kernel stack และ User stack สำหรับจัดการกับขอมูลตาง ๆ ของ โพรเซส เมื่อมีการประมวลผลในขณะที่โพรเซสกำลังประมวลผลอยู่ รีจีสเตอร์ของโพรเซสจะถูกควบคุมโดยโพรเซสเอง และรีจีสเตอร์เหลานี้จะถูกเก็บไว้ใช้งาน

เมื่อรีจีสเตอร์ถูกบล็อกในสวนที่เปนมัลติเธรดนั้น ยังมี PCB อยู่หนึ่งตัวรวมไปถึงเนื้อที่ใชสอยของผู้ใช้ แต่จะมีผู้ใช้และ Kernel stacks สำหรับเธรดทุกตัวรวมไปถึง Thread Control Block (TCB) ซึ่งจะเปนที่เก็บรีจีสเตอร์ ข้อมูลลำดับความสำคัญและขอมูลอื่น ๆ ที่เกี่ยวของกับสถานะภาพของเธรดนั้น ๆ เพราะฉะนั้นเธรดทั้งหมดจึงใชทรัพยากรของโพรเซสรวมกัน มีการรับรูการเปลี่ยนแปลงต่าง ๆ ที่เกิดขึ้น เชน ถาเธรดตัวหนึ่งเปดแฟ้มข้อมูลขึ้นมาอาน เธรดตัวอื่น ๆ สามารถอานแฟ้มข้อมูลเหล่านั้นไดเหมือนกัน



ภาพที่ 4.3 แสดงความแตกต่างระหว่างโพรเซสกับเธรด
ที่มา: Cardiff School of Computer Science & Informatics. Retrieved June 11, 2014 from http://www.cs.cf.ac.uk/Dave/C/node29.html

4.4 สถานะของเธรด

เชนเดียวกันกับโพรเซสสถานะของเธรด จะมีอยูสามสถานะคือ Ready, Running, และ Blocked ดังนั้นขั้นตอนที่เกี่ยวของกับการเปลี่ยนสถานะของเธรดคือ

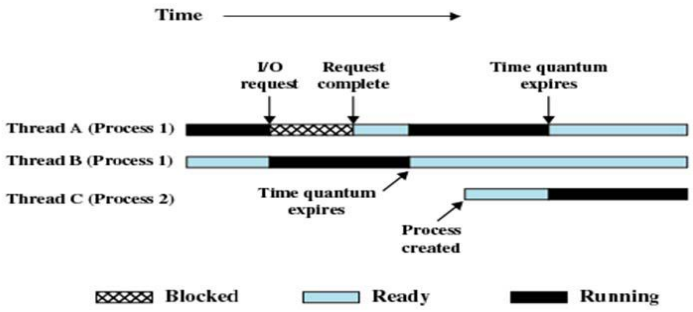
1. Spawn หมายถึงการที่โพรเซสตัวหนึ่งสรางโพรเซสอีกตัวหนึ่งขึ้นมา (ซึ่งทำใหเกิดนิยาม Parent process และ Child process) และเมื่อโพรเซสทำการ Spawn เธรดที่อยูในโพรเซสก็ทำการ Spawn ดวยและเธรดที่อยูในโพรเซสก็สามารถที่จะ Spawn เธรดใหมไดดวย

2. Block เมื่อใดก็ตามที่เธรดตองรอใหเหตุการณใด ๆ เกิดขึ้น มันก็จะทำการบล็อกซึ่งก็จะทำใหมี

การเก็บขอมูลที่เกี่ยวของ เชน User register, Program counter และ Stack pointer ไว้ และซีพียูจะไปใหบริการแกเธรดตัวอื่นที่พรอมสำหรับการบริการตอไป

3. Unblock เมื่อมีเหตุการณที่เธรดถูกบล็อก เธรดก็จะถูกนำไปเก็บไวใน Ready queue

4. Finish เมื่อเธรดสิ้นสุดการทำงานคาตาง ๆ ก็จะถูกส่งคืนให้กับระบบภาพแสดงตัวอยางของเธรดในระบบ Multithreading

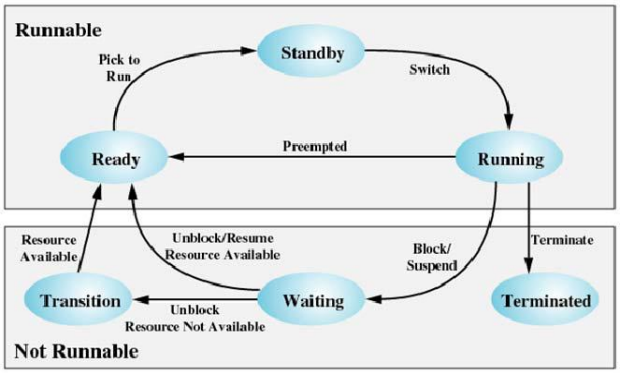


ภาพที่ 4.4 แสดงตัวอย่างของเธรดใน Multithreading system

จากภาพที่ 4.4 เรามีเธรดอยูสามตัว โดยมีสองโพรเซสที่กำลังทำงานสลับกันไปมาอยู่ในซีพียู การประมวลผลดังกล่าวมีการส่งข้อมูลไปมาระหว่างเธรด เมื่อมีการบล็อกเกิดขึ้นจากเธรดที่กำลังทำงานอยู หรือเมื่อเวลาที่กำหนดไว้หมดลง การคัดเลือกเธรดสำหรับการประมวลผลขึ้นอยู่กับ อัลกอริทึ่มของการกำหนดเวลาการใช้ซีพียู (Scheduling algorithm)

4.5 เธรดในระบบปฏิบัติการวินโดว์

เธรดใน Windows นั้นเป็นออบเจ็ต (เช่นเดียวกับโพรเซส) โดยที่โพรเซสจะเป็นส่วนประกอบที่เกี่ยวข้องกับการใช้งานของผู้ใช้ หรือ Application ที่ต้องการใช้ทรัพยากร เช่น หน่วยความจำ หรือไฟล์ ส่วนเธรดจะเป็นส่วนประกอบสำหรับการประมวลผล (Sequentially) ซึ่งสามารถถูกอินเตอรรัพได้ ซึ่งทำให้ซีพียูสามารถให้บริการแก่เธรดตัวอื่นได้ สถานะของเธรดใน Windows แสดงไดอะแกรมในภาพที่ 4.5



ภาพที่ 4.5 ไดอะแกรมสถานการณ์ทำงานของ Windows Thread

จากภาพที่ 4. 5 สามารถอธิบายสถานการทำงานของ Windows Thread ดังนี้

1. สถานะ Ready เธรดได้ถูกคัดเลือกให้เข้าทำงาน โดยมีการจัดอันดับด้วย Priority

2. สถานะ Standby เธรดได้รับการบริการจากซีพียูในอันดับถัดไปเมื่อซีพียูว่าง

3. สถานะ Running เธรดกำลังทำงานในซีพียูและจะทำไปจนกว่าจะถูกดึงออก หรือถูกบล็อกใช้เวลาที่กำหนดไว้หมด หรือเสร็จงาน ซึ่งถ้าเป็นในสองกรณีแรก เธรดก็จะถูกนำไปเก็บไว้ใน Ready queue เพื่อรอรับการบริการต่อไป

4. สถานะ Waiting เธรดจะเข้าสู่สถานะรอเมื่อ 1) ถูกบล็อก 2) รอให้งานบางอย่างเสร็จ 3) ระบบบังคับให้เธรดหยุดตัวเองลง เธรดจะกลับเข้าสู่สถานะ Ready เมื่อทรัพยากรที่เธรดต้องการมีให้ใช้

5. Transition เธรดจะเข้าสู่สถานะนี้เมื่อการรอสิ้นสุดลงและพร้อมที่จะเข้าสู่สถานะ Running แต่ทรัพยากรที่มันต้องการยังไม่มี (ระบบยังให้ไม่ได้)

6. Terminated เธรดสามารถที่จะยุติการทำงานของตัวเอง หรือจากเธรดอื่น หรือเมื่อ Parent process ยุติการทำงาน เมื่อระบบเสร็จสิ้นการทำงาน เธรดก็จะถูกดึงออกจากระบบ หรืออาจถูกเก็บไว้เพื่อการทำงานในอนาคต

4.6 ข้อได้เปรียบของ Multithreaded

การที่ระบบปฏิบัติการสนับสนุนระบบ Multithread ทำให้มีข้อได้เปรียบในด้านต่าง ๆ โดยแบ่งออกเป็น 4 ด้านหลัก ๆ ดังนี้

1. การตอบสนอง (Response) ระบบ Multithread ที่โต้ตอบแอปพลิเคชั่นจะยินยอมให้โปรแกรมยังคงดำเนินต่อไป ถึงแม้ว่าจะมีบางส่วนถูกบล็อกหรือมีการปฏิบัติที่ยาวนาน เนื่องจากการเพิ่มการโต้ตอบกับผู้ใช้นั่นเอง ยกตัวอย่างเช่น โปรแกรมเว็บบราวเซอร์ยังคงโต้ตอบกับผู้ใช้ได้ในขณะที่มีหนึ่งเธรดที่กำลังโหลดรูปภาพอยู่

2. การใช้ทรัพยากรร่วมกัน (Share resource) โดยปกติเธรดจะแชร์หน่วยความจำ และทรัพยากรของโพรเซสอยู่แล้ว ข้อได้เปรียบของการแชร์โค้ดจะทำให้แอปพลิเคชั่นสามารถมีกิจกรรมของเธรดได้หลาย ๆ กิจกรรมภายในเลขที่ตำแหน่งเดียวกัน

3. ประหยัด (Economic) การจัดสรรหน่วยความจำและทรัพยากรสำหรับการสร้างโพรเซสมีค่าใช้จ่ายมาก ในทางตรงข้ามเนื่องจากเธรดแชร์ทรัพยากรของโพรเซสที่มันอาศัยอยู่แล้ว ทำให้เกิดการประหยัดในการสร้างเธรดและทำการ Context switch ของเธรดเป็นการยากที่จะวัดความแตกต่างระหว่างการสร้างและคงสภาพโพรเซสที่มีมากกว่าเธรดได้ แต่โดยปกติแล้วจะใช้เวลาในการสร้างและคงสภาพโพรเซสสูงกว่าเวลาที่ใช้กับเธรดใน Solaris2 การสร้างโพรเซสจะช้ากว่าการสร้างเธรด 30 เท่าและ Context switch จะช้ากว่า 5 เท่า

4. รองรับการขยายระบบ (Scalability) ด้วยประโยชน์ของสถาปัตยกรรมที่รองรับการทำงานหลาย ๆ เธรด ทำให้ระบบช่วยเสริมประสิทธิภาพการทำงานของเธรดให้สูงขึ้น โดยแต่ละเธรดสามารถทำงานขนานกันไปในโพรเซสเซอร์อื่นได้ ในโพรเซสที่มีเธรดเดียวสามารถรันเพียงซีพียูเดียวเท่านั้น ไม่ว่าจะมีกี่ซีพียูก็ตามระบบมัลติเธรดในเครื่องที่มีหลายซีพียูจะเพิ่มประสิทธิภาพในการทำงานพร้อม ๆ กันได้มากขึ้น ในสถาปัตยกรรมที่มีซีพียูเดียว ซีพียูจะย้ายแต่ละเธรดให้เร็วมากขึ้น เพื่อให้ทำงานเสมือนว่าขนานกันอยู่ แต่ในความเป็นจริงจะรันเพียงเธรดเดียวเท่านั้นในเวลานั้น ๆ

4.7 เธรดสำหรับผู้ใช้และเธรดสำหรับระบบปฏิบัติการ

เธรดอาจจะแบ่งตามระดับการสนับสนุนได้ 2 แบบที่สัมพันธ์กัน คือ เธรดสำหรับผู้ใช้ง่ายที่จะถูกสร้างและอาจถูกยกเลิกก่อนเข้าเธรดสำหรับระบบปฏิบัติการได้และเธรดสำหรับระบบปฏิบัติการรองรับเธรดสำหรับผู้ใช้และการปฏิบัติงาน

4.7.1 เธรดสำหรับผู้ใช้ (User Thread)

จะได้รับการสนับสนุนจาก Kernel ด้านบน และอยู่ในไลบรารีของเธรดในระดับของผู้ใช้ ไลบรารียังสนับสนุนการสร้างเธรดการจัดเวลา และการจัดการเธรดโดยไม่ต้องได้รับการสนับสนุนจาก Kernel เนื่องจากไม่ต้องยุ่งเกี่ยวกับเธรดระดับผู้ใช้ การสร้างเธรดและการจัดเวลา เธรดทั้งหมดจะกระทำเสร็จสิ้นภายในพื้นที่ของผู้ใช้โดยไม่จำเป็นต้องใช้ Kernel ดังนั้นเธรดในระดับผู้ใช้สามารถสร้างและจัดการได้อย่างรวดเร็ว อย่างไรก็ตามถ้า Kernel เป็น Single thread แล้ว เธรดระดับผู้ใช้จะบล็อก System call จนเป็นเหตุให้ทุกโพรเซสถูกบล็อก ถึงแม้ว่าเธรดอื่นจะยังคงรันอยู่ในแอปพลิเคชั่นก็ตาม ไลบรารีของ User thread รวมถึง POSIX Pthreads, Windows, และ Java Pthreads

4.7.2 เธรดสำหรับระบบปฏิบัติการ (Kernel Thread)

ได้รับการสนับสนุนโดยตรงจากระบบปฏิบัติการ โดย Kernel จะสร้าง จัดเวลา และจัดการเธรดภายในพื้นที่ของ Kernel เอง เนื่องจากระบบปฏิบัติการเป็นผู้จัดการเกี่ยวกับการสร้างและจัดการเธรดเอง จึงทำให้เธรดสำหรับระบบปฏิบัติการจะสร้างและจัดการได้ช้ากว่าเธรดสำหรับผู้ใช้ อย่างไรก็ตามเพราะ Kernel จัดการเกี่ยวกับเธรด ดังนั้นถ้าเธรดเกิดการบล็อก System call จะทำให้ Kernel จัดการนำเอาเธรดอื่นในแอปพลิเคชั่นเข้ามาดำเนินการแทนได้ เช่นเดียวกับในสภาวะมัลติโพรเซสเซอร์ที่ Kernel สามารถจัดเธรดลงในโพรเซสเซอร์อื่นได้ ระบบปฏิบัติการที่สนับสนุนเธรดสำหรับระบบปฏิบัติการเช่น ระบบปฏิบัติการวินโดว์ ลินุกส์ แมคโอเอส และ โซลาริส เป็นต้น

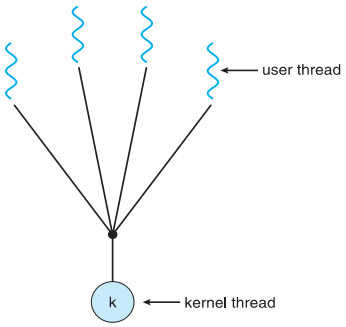
4.8 รูปแบบของเธรด

มีหลายระบบที่สนับสนุนทั้งเธรดสำหรับผู้ใช้และเธรดสำหรับระบบปฏิบัติการ ในรูปแบบที่แตกต่างกัน พิจารณาในรูปแบบทั้งสามของการดำเนินการเธรดดังนี้ เราจะอธิบายการทำงานของเธรดซึ่งมีการพัฒนามาตลอด แต่อย่างไรก็ตามการสนับสนุนการทำงานของเธรดจะขึ้นอยู่กับระดับของผู้ใช้จากเธรดของผู้ใช้หรือจาก Kernel แต่เธรดของผู้ใช้จะสนับสนุนมากกว่า Kernel และสามารถควบคุมโดยไม่ต้องใช้การสนับสนุนจาก Kernel ส่วนเธรดของ Kernel นั้นจะสนับสนุนและควบคุมโดยตรงจากระบบปฏิบัติการและเกือบทุกรุ่นของระบบปฏิบัติการไม่ว่าจะเป็น ระบบปฏิบัติการวินโดว์ ลินุกส์ แมคโอเอส โซลาริส และยูนิกส์ (เช่น Tru64 UNIX) ที่สนับสนุนการทำงานของ Kernel

ในที่สุดแล้วเธรดของผู้ใช้และเธรดของ Kernel ก็ยังเชื่อมโยงกันอยู่ดี ในส่วนนี้สามารถพิสูจน์ความสัมพันธ์ซึ่งสามารถแบ่งออกเป็น 3 รูปแบบที่เป็นที่รู้จักกันโดยทั่วไป

4.8.1 รูปแบบ Many-to-One

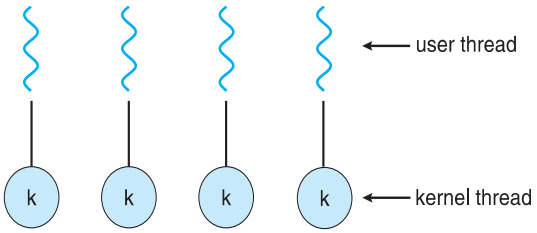
รูปแบบ Many-to-One เป็นรูปแบบที่ใช้เธรดสำหรับระบบปฏิบัติการ 1 หน่วย กับเธรดสำหรับผู้ใช้หลายหน่วย (ดังภาพที่ 4.6) การจัดการเธรดจะอยู่ในพื้นที่ของผู้ใช้ซึ่งมีประสิทธิภาพ แต่ถ้าเธรดบล็อก System call โพรเซสทั้งหมดจะถูกบล็อกไปด้วย เนื่องจากจะมีเพียงเธรดเดียวเท่านั้นที่เข้าถึง Kernel ในเวลาหนึ่ง ๆ เธรดหลาย ๆ เธรด ไม่สามารถรันขนานกันในระบบมัลติโพรเซสเซอร์ได้ ระบบที่ใช้รูปแบบนี้เช่น Green thread ซึ่งเป็นไลบรารีในโซลาริสทู (Solaris 2) นอกจากนี้ในไลบรารีของ User thread ในระบบปฏิบัติการที่ไม่สนับสนุนสำหรับระบบปฏิบัติการ จะใช้รูปแบบ Many-to-One นี้



ภาพที่ 4.6 รูปแบบ Many-to-one
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.169)

4.8.2 รูปแบบ One-to-One

รูปแบบ One-to-One เป็นรูปแบบที่แต่ละเธรดสำหรับผู้ใช้ จะจับคู่กับเธรดสำหรับระบบปฏิบัติการ ในลักษณะ 1 ต่อ 1 (ดังภาพที่ 4.7) ทำให้สามารถทำงานพร้อมกันดีกว่าแบบ Many-to-One โดยยอมให้เธรดอื่นรันได้เมื่อเธรดบล็อก System call นอกจากนี้โมเดลนี้ยังยอมให้หลาย ๆ เธรดทำงานแบบขนานกันได้ในระบบมัลติโพรเซสเซอร์ได้อีกด้วย มีข้อที่คำนึงอยู่ข้อเดียวคือ การสร้างเธรดสำหรับผู้ใช้ จำเป็นต้องสร้างเธรดสำหรับระบบปฏิบัติการที่สัมพันธ์กัน เนื่องจากการสร้างเธรดสำหรับระบบปฏิบัติการเป็นส่วนสำคัญในเพิ่มประสิทธิภาพของแอปพลิเคชัน ระบบที่โมเดลมีข้อจำกัดที่จำนวนเธรดที่สนับสนุนในระบบได้ โมเดลนี้นำมาใช้ในระบบ เช่น ในระบบปฏิบัติการวินโดว์ เป็นต้น

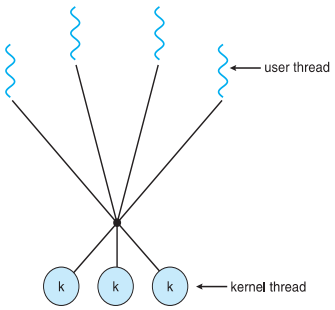


ภาพที่ 4.7 รูปแบบ one-to-one
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.170)

4.8.3 รูปแบบ Many-to-Many

รูปแบบ Many-to-Many เป็นรูปแบบที่อาจจะมีจำนวนเธรดสำหรับผู้ใช้ มากกว่าหรือเท่ากับจำนวนเธรดสำหรับระบบปฏิบัติการก็เป็นได้ (ดังภาพที่ 4.8) จำนวนเธรดสำหรับระบบปฏิบัติการ อาจจะเป็นตัวกำหนดแอปพลิเคชันเฉพาะหรือเครื่องเฉพาะ (แอปพลิเคชั่นอาจจะมีเธรดสำหรับระบบปฏิบัติการบนมัลติโพรเซสเซอร์มากกว่าเธรดสำหรับระบบปฏิบัติการบนโพรเซสเซอร์เดียว) ในขณะที่รูปแบบ Many-to-Many ยอมให้ผู้พัฒนาสร้างเธรดสำหรับผู้ใช้ ได้ตามที่เขาต้องการ แต่จะไม่สามารถทำงานได้พร้อมกัน

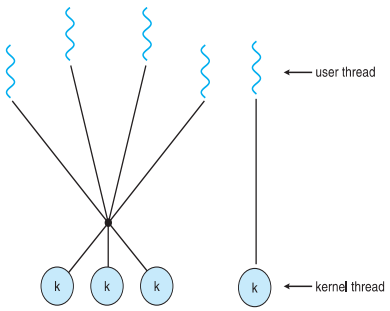
เนื่องจาก Kernel จะจัดเวลาให้ครั้งละเธรดเท่านั้น โมเดล One-to-One ยอมให้รันพร้อมกันได้ดีกว่า แต่ผู้พัฒนาต้องระวังว่าต้องไม่สร้างเธรดมากเกินไปในแอปพลิเคชั่น (ในบางครั้งอาจจะจำกัดจำนวนเธรดที่จะสร้าง) รูปแบบ Many-to-Many จะลดข้อจำกัดของรูปแบบทั้งสอง กล่าวคือ ผู้พัฒนาสามารถสร้างเธรดสำหรับผู้ใช้เท่าที่จำเป็น และสัมพันธ์กับเธรดสำหรับระบบปฏิบัติการที่สามารถรันแบบขนานในระบบมัลติโพรเซสเซอร์ นอกจากนั้น เมื่อเธรดเกิดการบล็อก System call แล้ว Kernel จะจัดเวลาเพื่อนำเธรดอื่นขึ้นมารันก่อนก็ได้ โมเดลนี้เป็นตัวอย่างของระบบปฏิบัติการ ยูนิกส์ เป็นต้น



ภาพที่ 4.8 รูปแบบ Many-to-many
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.170)

4.9 คลังข้อมูลการจัดการเธรด

คลังข้อมูลการจัดการเธรด เป็นข้อกำหนดที่สร้างโดยโปรแกรมเมอร์ สำหรับการจัดสร้างและจัดการเกี่ยวกับเธรดโดยมีสองขั้นตอนในการทำงานคือ



ภาพที่ 4.9 รูปแบบ Two-level
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.171)

1. การเข้าถึงแหล่งของข้อมูลทั้งหมดในเนื้อที่ของผู้ใช้ โดยไม่ผ่าน Kernel ทุกค่าและทุกโครงสร้างข้อมูลของแหล่งข้อมูลในพื้นที่ของผู้ใช้ หรือเป็นการเรียกจากฟังก์ชันเฉพาะในพื้นที่ของผู้ใช้และไม่ใช้ System call

2. การเข้าถึงเครื่องมือที่รับรองแหล่งของ Kernel โดยตรงด้วยระบบปฏิบัติการ ในส่วนนี้โค้ดและโครงสร้างข้อมูลสำหรับแหล่งข้อมูลในพื้นที่ของ Kernel การเรียกฟังก์ชันใน API แหล่งข้อมูลแบบธรรมดาจะส่งผลใน System call ถึง Kernel

Thread Library 3 อย่างหลัก ๆ ที่ใช้ในปัจจุบันคือ 1) POSIX Pthreads 2) Win32 3) Java

Pthreads เป็นเธรดมาตรฐานของ POSIX จะจัดการเกี่ยวกับระดับของผู้ใช้ หรือ ระดับของ Kernel Threads Library ของ Win32 เป็นแหล่งข้อมูลของ Kernel level ที่สะดวกของระบบวินโดวส์ เธรดของจาวาเป็น API ที่ยอมรับในการสร้างและจัดการเธรดโดยตรงในโปรแกรมจาวา อย่างไรก็ตามเพราะว่ากรณีส่วนใหญ่แล้ว JVM จะเป็นตัวทำงานทางด้านบนสุดของระบบปฏิบัติการ เธรดของจาวา API นั้นเป็นรูปแบบเครื่องมือที่ใช้ Threads library ได้สะดวกบนระบบโฮส ซึ่งคล้ายกับระบบปฏิบัติการวินโดวส์ เธรดของจาวาเป็นรูปแบบเครื่องมือที่ใช้ในระบบ Win32 API, UNIX และระบบปฏิบัติการลินุกส์เช่นเดียวกับการใช้ Pthreads

ในส่วนที่เหลือของเรื่องส่วนนี้เราจะอธิบายถึงการสร้างเธรดพื้นฐาน ซึ่งใช้ 3 Thread library และมีภาพประกอบตัวอย่าง เราออกแบบโปรแกรมของรูปแบบหลายเธรดเมื่อทำการหาผลรวมของค่าที่เป็นจำนวนเต็มบวกในเธรดที่ใช้ซึ่งรู้จักกันในรูปฟังก์ชันการหาผลรวม

$$sum = \sum_{i=0}^{N} \mathrm{i}$$

จากตัวอย่าง ถ้า N มีค่าเท่ากับ 5 ในฟังก์ชันนี้ค่าผลรวม จาก 0 ถึง 5 คือ 15 โปรแกรมนี้จะทำงานด้วยค่าขอบบนของการหาผลรวมของเธรดมาตรฐาน ถ้าผู้ใช้ใส่ค่า 8 ผลรวมของจำนวนค่าจาก 0 ถึง 8 จะออกมาเป็นอย่างไร

4.9.1 Pthreads

Pthreads เป็นตัวพื้นฐานของ POSIX (IEEE 103.1C) เรียกได้ว่าเป็น API สำหรับการสร้างเธรดและสิ่งที่เกิดขึ้นในเวลาเดียวกัน เป็นตัวบ่งบอกถึงพฤติกรรมของเธรดโดยไม่ใช้เครื่องมือ การออกแบบระบบปฏิบัติการมักจะใช้เครื่องมือในงานที่ต้องการ ระบบปฏิบัติการส่วนใหญ่ใช้ Pthreads เป็นเครื่องมือไม่ว่าจะเป็นระบบปฏิบัติการโซราริส ลินุกส์ แมคโอเอส และยูนิกส์ เป็นเครื่องมือที่สะดวกในการใช้งานทั่วไปสำหรับงานที่หลากหลายซึ่งดีกับระบบปฏิบัติการวินโดวส์

```
#include <pthread.h>
#include <stdio.h>
int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */
int main(int argc, char *argv[])
{
pthread t tid; /* the thread identifier */
pthread attr t attr; /* set of thread attributes */
if (argc != 2) {
fprintf(stderr,"usage: a.out <integer value>\n");
return -1;
}
if (atoi(argv[1]) < 0) {
fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
return -1;
}
/* get the default attributes */
pthread attr init(&attr);
/* create the thread */
pthread create(&tid,&attr,runner,argv[1]);
/* wait for the thread to exit */
pthread join(tid,NULL);
printf("sum = %d\n",sum);
}
/* The thread will begin control in this function */
void *runner(void *param)
{
int i, upper = atoi(param);
sum = 0;
for (i = 1; i <= upper; i++)
sum += i;
pthread exit(0);
}
```

ภาพที่ 4.10 โค้ดภาษาซี ในการเขียน Multithreaded C program โดยการใช้ Pthreads API.
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.173)

โปรแกรมภาษาซีที่แสดงในภาพที่ 4.10 เป็นตัวสาธิตแบบพื้นฐานของ Pthreads API สำหรับสร้างโปรแกรมรูปแบบหลายเธรด ที่เราทำการคำนวณหาผลรวมของจำนวนเต็มบวกในการแบ่งเธรด ในโปรแกรม Pthreads แบ่งเธรดและเริ่มประมวลผล โดยแบ่งออกเป็นฟังก์ชัน ในภาพที่ 4.10 ส่วนของฟังก์ชัน runner( ) เมื่อโปรแกรมเริ่มทำงาน เธรดเส้นหนึ่งจะควบคุมการทำงานของ main() หลัง จากนั้น main( ) จะสร้างเธรดที่สองและเริ่มควบคุมการทำงานของฟังก์ชัน runner()

เธรดสองเส้นสามารถใช้ข้อมูลร่วมกันได้ทั้งหมด เมื่อเรามองลึกลงไปในโปรแกรมจะเห็นว่า ทุกโปรแกรม Pthreads นั้นจะถูกรวบรวมไว้ในส่วนของเฮดเดอร์ไฟล์คือ Pthread.h ส่วนของ Pthreads จะรองรับเกี่ยวกับการจำแนกของการสร้างเธรด แต่ละเธรดงานสามารถที่จะกำหนดขนาดของ Attribute และตารางการใช้ข้อมูล

Pthread_attr_t จะเป็นตัวแทน Attribute ของเธรด สามารถกำหนดค่า Attribute ในส่วนของฟังก์ชันเรียก pthread_atr_int(&attr) เพราะเราไม่สามารถกำหนดค่าของ Attribute ได้แน่นอน โดยสามารถใช้ค่า Attribute พื้นฐานได้

การแบ่งเธรดถือเป็นการสร้างเธรดด้วย pthreads_create() ของฟังก์ชันเรียก ในการเพิ่มและผ่านของเธรด สามารถจำแนกเธรดได้ อีกทั้งชื่อของฟังก์ชันเมื่อมีเธรดใหม่เริ่มประมวลผล ในส่วนนี้ฟังก์ชัน runner() สุดท้ายเราจะส่งค่าจำนวนเต็มคงที่ไปยังเธรดที่ควบคุมมัน

จุดนี้โปรแกรมมี 2 เธรดคือ เธรดพ่ออยู่ใน main() และเธรดลูก ทำการรวมการทำงาน ในฟังก์ชัน runner( ) หลังจากการสร้างเธรดลูก เธรดพ่อจะรอคำสั่งสิ้นสุดการทำงานจากฟังก์ชัน pthread_join( ) เธรดลูกจะสิ้นสุดการทำงานเมื่อมันเรียกฟังก์ชัน pthread_exit( ) เธรดลูกสามารถที่จะกลับมาทำงานได้อีกครั้งเมื่อเธรดพ่อส่งข้อมูลไปยังส่วนของข้อมูลที่ใช้ร่วมกัน (Shared data)

4.9.2 เธรดของระบบ Win32

วิธีการสำหรับสร้างเธรดที่ใช้ในระบบ Win32 วีธีการจะคล้าย ๆ Pthreads ในระบบอื่น ๆ เราจะรวมไปถึงเธรดระบบ Win32 ในโปแกรมภาษาซีที่แสดงดังภาพที่ 4.11 จะเห็นได้ว่า เราจะใช้ Windows.h เป็นหัวของไฟล์ เมื่อใช้รูปแบบ Win32 เป็นรูปแบบในการนำเสนอ

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */
/* the thread runs in this separate function */
DWORD WINAPI Summation(LPVOID Param)
{
DWORD Upper = *(DWORD*)Param;
for (DWORD i = 0; i <= Upper; i++)
Sum += i;
return 0;
}
int main(int argc, char *argv[])
{
DWORD ThreadId;
HANDLE ThreadHandle;
int Param;
if (argc != 2) {
fprintf(stderr,"An integer parameter is required\n");
return -1;
}
Param = atoi(argv[1]);
if (Param < 0) {
fprintf(stderr,"An integer >= 0 is required\n");
return -1;
}
/* create the thread */
ThreadHandle = CreateThread(
NULL, /* default security attributes */
0, /* default stack size */
Summation, /* thread function */
&Param, /* parameter to thread function */
0, /* default creation flags */
&ThreadId); /* returns the thread identifier */
if (ThreadHandle != NULL) {
/* now wait for the thread to finish */
WaitForSingleObject(ThreadHandle,INFINITE);
/* close the thread handle */
CloseHandle(ThreadHandle);
printf("sum = %d\n",Sum);
}
}
```

ภาพที่ 4.11 โค้ดภาษาซี ในการสร้าง Multithreaded C program โดยการใช้ Windows API.
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.175)

Pthreads ที่แสดงในภาพที่ 4.11 ข้อมูลถูกแบ่งโดยเธรดที่ทำการแบ่ง ในส่วนนี้ ผลรวมทั้งหมดจะรองรับ (ข้อมูล DWORD) รูปแบบเป็นจำนวนเต็มบวกขนาด 32 บิต เราจะอธิบายฟังก์ชัน summation() นี้เป็นการทำงานในเธรดแบ่ง ฟังก์ชันนี้เป็นการส่ง Pointer ไปยังค่าว่าง ระบบ Win32 ถูกกำหนดโดย LPVOID การทำงานของเธรดนี้ฟังก์ชันที่กำหนดข้อมูลทั้งหมด จากผลรวมไปถึงค่าของผลรวม จาก 0 ถึงค่าคงที่ที่ผ่านไปถึงฟังก์ชัน summation( )

เธรดที่สร้างขึ้นในระบบ Win32 จะใช้ฟังก์ชัน CreateThread ( ) และใน Pthreads จะกำหนดขนาด Attribute สำหรับเธรดที่แบ่งตัวส่งค่าไปยังฟังก์ชัน attribute ที่รวบรวมข้อมูลที่ปลอดภัย ขนาดของข้อมูลนั้นจะเพิ่มหรือลดสามารถที่จะกำหนดได้ ถ้าเธรดอยู่ในสถานะรอการทำงาน ในโปรแกรมนี้จะใช้ค่าพื้นฐานสำหรับ Attribute นี้ (เป็นการกำหนดค่าของเธรด ในสถานะรอแทนที่เราจะทำมันในตอนที่ซีพียูทำงาน) ช่วงที่เธรดลูกสร้าง เธรดพ่อจะรอมันทำงานจนสิ้นสุดการทำงาน ก่อนที่จะแสดงค่าผลลัพธ์ออกมา ซึ่งค่านั้นจะถูกกำหนดโดยเธรดลูก จะถูกยกเลิกโดยโปรแกรม Pthreads (ภาพที่ 4.10) เธรดพ่อจะรอเธรดลูกใช้ pthread_join( ) ซึ่งการทำงานคล้ายกับระบบ Win32 ที่ใช้ฟังก์ชัน WaitForSingleObject( ) เพราะการสร้างเธรดจะไปปิดส่วนของการสิ้นสุดการทำงานของเธรดลูก

4.9.3 เธรดของระบบภาษาจาวา (Java Thread)

เธรดเป็นรูปแบบแรกของการประมวลผลโปรแกรมในโปรแกรมจาวา และ ภาษาจาวาเป็นข้อกำหนดลักษณะของกลุ่มสมาชิกขนาดใหญ่ในการสร้างและจัดการเกี่ยวกับเธรด ทุกโปรแกรมจาวาประกอบไปด้วยเธรดเดี่ยวขนาดเล็กทำการควบคุมอยู่ ตัวอย่างคือโปรแกรมจาวาที่มีเมธอด main( ) เพียงอย่างเดียวที่ทำงานแบบเธรดเดี่ยวของ JVM

เรามี 2 วิธีในการสร้างเธรดในโปรแกรมจาวา วิธีที่หนึ่ง คือ การสร้างคลาสใหม่ที่มาจากเธรดคลาส เธรดและผ่านเมธอด run( ) อีกทางเลือกหนึ่ง สามารถควบคุมได้หลายผู้ใช้ซึ่งเป็นวิธีการกำหนดขอบเขตของคลาสที่เป็นตัวทำงานอยู่ ซึ่งตัวที่ทำงานนั้นเราจะใช้เป็นตัวที่เราคิดตาม เมื่อคลาสทำงานคลาสจะเป็นตัวกำหนดเมธอด run( ) โค้ดที่ใช้ในเมธอด run( ) เป็นอะไรที่ทำการแบ่งเธรด

คลาสผลรวมที่ใช้แสดงส่วนที่ทำงานได้ การสร้างเธรดเป็นการทำงานโดยส่วนของการสร้างวัตถุตัวอย่าง ของคลาสเธรดและผ่านการสร้างวัตถุที่สามารถทำงานได้ การสร้างเธรดที่ไม่เจาะจงการสร้างเธรดใหม่ ในทางตรงกันข้ามมันจะเป็นเมธอด start() ที่ทำการสร้างเธรดใหม่อย่างแท้จริง การเรียกเมธอด start( ) เพื่อการทำวัตถุใหม่จะต้องใช้ 2 สิ่งคือ

1. การจัดการหน่วยความจำและการเริ่มใช้เธรด JVM
2. เรียกเมธอด run( ) ให้เธรดทำงานที่เหมาะสมด้วย JVM

เมื่อโปรแกรมทำงานเสร็จ 2 เธรดที่ถูกสร้างโดย JVM อย่างที่หนึ่ง เธรดพ่อเริ่มการทำงานในเมธอด main( ) อย่างที่สอง เธรดจะถูกสร้างเมื่อ เมธอด start( ) ในวัตถุเธรดเป็นตัวเรียก เธรดลูกนี้จะเริ่มทำการประมวลผลในเมธอด run( ) ของคลาส Summation หลังจากส่งค่าผลลัพธ์ เธรดนี้จะถูกกำจัดเมื่อมันออกจากเมธอด run( )

```
class Sum
{
private int sum;
public int getSum() {
return sum;
}
public void setSum(int sum) {
this.sum = sum;
}
}
class Summation implements Runnable
{
private int upper;
private Sum sumValue;
public Summation(int upper, Sum sumValue) {
this.upper = upper;
this.sumValue = sumValue;
}
public void run() {
int sum = 0;
for (int i = 0; i <= upper; i++)
sum += i;
sumValue.setSum(sum);
}
}
public class Driver
{
public static void main(String[] args) {
if (args.length > 0) {
if (Integer.parseInt(args[0]) < 0)
System.err.println(args[0] + " must be >= 0.");
else {
Sum sumObject = new Sum();
int upper = Integer.parseInt(args[0]);
Thread thrd = new Thread(new Summation(upper, sumObject));
thrd.start();
try {
thrd.join();
System.out.println
("The sum of "+upper+" is "+sumObject.getSum());
} catch (InterruptedException ie) { }
}
}
else
System.err.println("Usage: Summation <integer value>"); }
}
```

ภาพที่ 4.12 แสดงวิธีในการสร้างเธรดในโปรแกรมจาวา
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.178)

การแบ่งปันข้อมูลระหว่างเธรดจะเกิดขึ้นง่ายในระบบ WIn32 และ Pthreads ข้อมูลที่ถูกแบ่งปันนั้นจะถูกเปิดเผยทั่วไปอย่างแท้จริงซึ่งเป็นภาษาแนวคิดเชิงวัตถุ ภาษาจาวาไม่มีแนวคิดของข้อมูลแบบนี้ ถ้าเธรดอย่างน้อยสองเธรดทำการแบ่งปันข้อมูลในโปรแกรมจาวา การแบ่งปันข้อมูลเกิดขึ้นจากการส่งข้อมูลจากแหล่งที่แบ่งปันข้อมูลถึงเธรดที่เหมาะสม ในโปรแกรมจาวาที่แสดงในภาพที่ 4.12 เธรดหลักและเธรดรองทำการแบ่งปันข้อมูลกันในคลาส sum วัตถุที่ถูกแบ่งถูกอ้างอิงมาจากเมธอด getSum( ) และ setSum( ) (มีข้อสงสัยว่า ทำไมจึงไม่ใช้วัตถุที่เป็นจำนวนเต็มแทนที่จะใช้การออกแบบคลาสใหม่ มีสาเหตุมาจากคลาสของจำนวนเต็มเปลี่ยนแปลงไม่ได้ นั่นคือ ค่าของข้อมูลในเซตมันไม่สามารถเปลี่ยนแปลงได้

การเรียกเธรดพ่อใน Pthreads และใน Win32 ใช้ pthread_join( ) และ WaitForSingle Object( ) ตามลำดับ รอเธรดลูกเสร็จสิ้นการทำงาน เมธอด join() ในจาวาทำงานคล้ายกัน (ข้อสังเกต join( ) ในจาวาจะไม่ใช้ InterruptedException)

4.10 การยกเลิกเธรด

การยกเลิกเธรดเป็นการทำให้เธรดจบการทำงานก่อนที่จะเสร็จสมบูรณ์ เช่น ถ้ามีหลายเธรดค้นหาข้อมูลในฐานข้อมูลพร้อมกัน แล้วมีเธรดหนึ่งให้ผลลัพธ์ออกมาแล้วเธรดที่เหลือจะถูกยกเลิกในสภาวะอื่น อาจจะเกิดเมื่อผู้ใช้กดปุ่มบนโปรแกรมเว็บบราวเซอร์เพื่อหยุดการโหลดข้อมูล เนื่องจากการโหลดข้อมูลจะใช้เธรดแยกกับการกดปุ่มบนคีย์บอร์ด ดังนั้นเมื่อผู้ใช้มีการกดปุ่ม Stop บนคีย์บอร์ด จึงทำให้โปรแกรมเว็บบราวเซอร์หยุดการโหลดข้อมูล เธรดที่ถูกยกเลิกอาจเรียกว่า Target thread ซึ่งการยกเลิกอาจมี 2 รูปแบบที่ต่างกัน ดังนี้ความยุ่งยากในการยกเลิกจะเกิดในภาวะที่ทรัพยากรถูกกำหนดให้ยกเลิกเธรด หรือมีเธรดหนึ่งถูกยกเลิกในขณะอยู่ระหว่างการบันทึกข้อมูลที่ใช้ทรัพยากรร่วมกันอยู่กับเธรดอื่น สิ่งนี้เป็นกรณีพิเศษของการยกเลิกแบบ Asynchronous ระบบปฏิบัติการมักจะแก้ปัญหาทรัพยากรจากการยกเลิกเธรด แต่ไม่ใช่ทุกทรัพยากร

ดังนั้นการยกเลิกแบบ Asynchronous อาจจะใช้ยกเลิกกับทรัพยากรทั่วไปไม่ได้ ในทางกลับกัน การยกเลิกแบบ Deferred จะทำงานโดยมีเธรดหนึ่งที่กำหนดว่า Target thread ใดจะถูกยกเลิก อย่างไรก็ตาม การยกเลิกนี้จะเกิดขึ้นเฉพาะเมื่อ Target thread นั้นตรวจสอบว่าตัวเองจะถูกยกเลิกหรือไม่ สิ่งนี้ยอมให้เธรดตรวจสอบ เพื่อให้ถูกยกเลิกในจุดที่ปลอดภัย ถ้ายกเลิกจุดที่ว่านี้ใน Pthread API เรียกว่า Cancellation points ในระบบปฏิบัติการส่วนมากจะยอมให้โพรเซสหรือ thread ถูกยกเลิกแบบ Asynchronous แต่ใน Pthread API ยังสนับสนุนการยกเลิกแบบ Defferred อีกด้วย สิ่งนี้หมายความว่าระบบปฏิบัติการที่มี Pthread API จะยอมให้ยกเลิกแบบ Defferred นั่นเอง

4.11 สรุป

เธรด คือ การเรียกใช้หน่วยประมวลผลให้เกิดประโยชน์สูงสุด และทำให้การทำงานของโปรแกรมมีประสิทธิภาพมากขึ้นและมีประโยชน์ต่อระบบในปัจจุบันที่เป็น Multicore เพราะสามารถเรียกใช้เธรดหลาย ๆ เธรดได้พร้อม ๆ กัน ทั้งนี้เธรดของโพรเซสเดียวกันสามารถจะทำงานแตกต่างกัน แต่ยังมีความเกี่ยวข้องกันบางอย่างและต้องทำงานอยู่ภายใต้สภาพแวดล้อมเดียวกัน เธรดเป็นการควบคุมการทำงานภายในของโพรเซส Multithreaded ก็คือโพรเซสจะประกอบไปด้วยเธรดที่ทำหน้าที่ต่างกันแต่ทำงานอยู่บนตำแหน่งหน่วยความจำเดียวกัน ข้อดีของ Multithreaded ประกอบไปด้วย การเพิ่มการตอบสนองต่อผู้ใช้ทั้งแชร์ ทรัพยากรในโพรเซส ความประหยัดและความสามารถของงาน ในสถาปัตยกรรมแบบมัลติโพรเซสเซอร์ ในระดับของเธรดสำหรับผู้ใช้ คือสิ่งที่โปรแกรมเมอร์มองเห็นและระบบปฏิบัติการ Kernel จะสนับสนุนจัดการในระดับ Kernel โดยทั่วไปแล้วเธรดสำหรับผู้ใช้จะเร็วกว่าในการสร้างและจัดการมากกว่า เธรดสำหรับระบบปฏิบัติการและไม่ก้าวก่ายสิ่งที่ Kernel ต้องการ

รูปแบบที่แตกต่างกันระหว่างเธรดสำหรับผู้ใช้กับเธรดสำหรับระบบปฏิบัติการ โดยรูปแบบ Many-to-one หมายถึง เธรดสำหรับระบบปฏิบัติการ 1 หน่วย กับเธรดสำหรับผู้ใช้หลายหน่วย เป็นการ

ออกแบบที่จะยอมให้เธรดเพียงเธรดเดียวที่เข้าถึง Kernel ในกรณีที่เธรดไปบล็อก System call จะทำให้โพรเซสทั้งหมดถูกบล็อกไปด้วย โดยรูปแบบนี้ยอมให้สร้างเธรดสำหรับผู้ใช้ได้ตามต้องการ แต่ไม่สามารถประมวลผลได้พร้อมกัน เพราะยอมให้เข้าใช้เธรดสำหรับระบบปฏิบัติการได้ครั้งละเธรดเท่านั้น

รูปแบบ One-to-one หมายถึง เธรดสำหรับระบบปฏิบัติการ 1 หน่วย กับเธรดสำหรับผู้ใช้ 1 หน่วย ซึ่งระบบปฏิบัติการจะยอมให้เธรดอื่นประมวลผลได้เป็นระบบขนาน ที่ทำงานแบบระบบมัลติโพรเซสเซอร์ มีการใช้หลักการนี้อยู่ในระบบปฏิบัติการวินโดว์ในปัจจุบัน โดยรูปแบบนี้ต้องไม่ยอมให้สร้างเธรดสำหรับผู้ใช้มากเกินไป

รูปแบบ Many-to-many หมายถึง รูปแบบที่ลดข้อจำกัดของ 2 แบบแรก ผู้ใช้สามารถสร้างเธรดสำหรับผู้ใช้เท่าที่จำเป็น และสัมพันธ์กับเธรดสำหรับระบบปฏิบัติการที่รับการทำงานแบบขนานในแบบมัลติโพรเซสเซอร์ เมื่อมีเธรดที่บล็อก System call ทาง Kernel จะจัดเวลาให้เธรดอื่นเข้ามาประมวลผลก่อนได้

เรื่องราวเกี่ยวกับเธรดมีหลายอย่างที่ต้องพิจารณา การยกเลิกเธรดเป็นเรื่องที่ต้องทำความเข้าใจ เพราะการยกเลิกหมายถึง การทำให้เธรดเป้าหมายจบการทำงาน ก่อนที่จะทำงานจนเสร็จสมบูรณ์ การยกเลิกนี้มี 2 วิธีคือ 1) Asynchronous cancellation การยกเลิกที่เธรดอื่นสั่งให้เธรดเป้าหมายหยุดทำงาน 2) Defered cancellation การยกเลิกเธรดเป้าหมาย โดยใช้ตรวจสอบตนเองว่าตนเองต้องถูกยกเลิกด้วยหรือไม่ Pthreads อ้างถึงมาตรฐาน POSIX (IEEE 1003.1c) เพื่อกำหนด API (Application Programming Interface) สำหรับสร้าง และการซินโครไนซ์เซชัน นี่คือการกำหนดสภาพแวดล้อมของเธรด ซึ่งการกำหนดสภาพแวดล้อมของเธรดนี้ถูกจำกัดใน Solaris2 แต่ Pthread ไม่ถูกสนับสนุนใน Windows แม้จะมี Shareware เผยแพร่แล้วก็ตาม

แบบฝึกหัดท้ายบทที่ 4

1. จงอธิบายและยกตัวอย่างโปรแกรม การทำงานแบบ Multithreading ทำงานมีประสิทธิภาพดีกว่า Single threaded

2. จงอธิบายอะไรคือข้อแตกต่างระหว่าง User-Level threads และ Kernel-Level threads

3. รูปแบบ Many-to-Many กับ รูปแบบ One-to-One แตกต่างกันอย่างไรอธิบาย

4. จงอธิบายและยกตัวอย่างโปรแกรมทำงานแบบ Multithreading ที่มีการทำงานด้อยประสิทธิภาพกว่า Single threaded

5. พิจารณาระบบ Multicore และโปรแกรม Multithreaded เขียนโดยใช้รูปแบบเธรด Many-to-Many จำนวนเธรดสำหรับผู้ใช้มากกว่าจำนวนเธรดสำหรับระบบปฏิบัติการ ดังนั้นจำนวนเธรดสำหรับระบบปฏิบัติการ จัดสรรให้โปรแกรมทำงานน้อยกว่าจำนวนแกนของหน่วยประมวลผล (Processing core) ได้หรือไม่

6. จากข้อ 5 จำนวนเธรดสำหรับระบบปฏิบัติการ อธิบายจัดสรรให้โปรแกรมทำงานมากกว่าจำนวนแกนของหน่วยประมวลผล แต่น้อยกว่าจำนวนระดับเธรดสำหรับผู้ใช้

7. จงอธิบายการยกเลิกเธรดแบบ Asynchronous แตกต่างกับการยกเลิกแบบ Deferred อย่างไร

8. องค์ประกอบใดต่อไปนี้ในขณะที่โปรแกรมอยู่ในสถานะการทำงาน สามารถถูกแชร์ทรัพยากรข้ามเธรดได้ในระบบ Multithreaded process

1) Register values
2) Heap memory
3) Global variables
4) Stack memory

บรรณานุกรมประจำบทที่ 4

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. (2004). **Modern Operating Systems. 4th ed.** Prentice Hall, Pearson Education International.

Andrew S. Tanenbaum, Maarten van Steen. (2014). **Distributed Systems Principles and Paradigms.** Prentice Hall, Pearson Education International.

บทที่ 5
การติดตาย

ในสภาพการทำงานแบบหลายโปรแกรม หลายโพรเซสอาจจะมีการแข่งกันเข้าใช้งานทรัพยากรที่มีอยู่จำกัด โพรเซสจะมีการร้องขอใช้งานทรัพยากร ถ้าในขณะนั้นทรัพยากรไม่ว่างก็จะส่งผลให้โพรเซสนั้นรอจนกว่าจะได้เข้าทำงาน โพรเซสถูกให้อยู่ในสถานการณ์รอไปไม่มีที่สิ้นสุด เนื่องจากทรัพยากรนั้นก็ถูกร้องขอจากโพรเซสอื่นที่รออยู่เช่นกัน ลักษณะเช่นนี้เรียกว่า การติดตาย (Deadlock)

5.1 รูปแบบโครงสร้าง

การทำงานร่วมกันของโพรเซสหลายโพรเซส (ในระบบมัลติโปรแกรมมิ่ง) เมื่อโพรเซสต้องการใช้ทรัพยากรจะต้องร้องขอทรัพยากรจากระบบ โดยสามารถขอใช้ทรัพยากรให้ได้จำนวนมากที่สุด เพื่อให้โพรเซสสามารถทำงานได้เสร็จสิ้น แต่ไม่สามารถร้องขอทรัพยากรมากเกินกว่าที่ระบบมีอยู่จริง เมื่อโพรเซสต้องการใช้ทรัพยากรของระบบ โพรเซสต้องทำตามลำดับขั้นตอนต่าง ๆ ต่อไปนี้

1. การร้องขอ (Request) เมื่อโพรเซสต้องการใช้ทรัพยากรใด ๆ จะต้องทำการร้องขอทรัพยากรนั้นจากระบบ โดยระบบจะตรวจสอบว่าทรัพยากรที่ถูกร้องขอว่างหรือไม่ กรณีที่ทรัพยากรว่าง โพรเซสจะได้รับการจัดสรรทรัพยากรทันที แต่ถ้าทรัพยากรไม่ว่าง (ถูกใช้งานโดยโพรเซสอื่น) โพรเซสที่ร้องขอจะต้องรอจนกว่าจะได้รับทรัพยากรที่ต้องการ

2. การใช้งาน (Use) เมื่อโพรเซสได้รับทรัพยากรที่ต้องการแล้ว โพรเซสสามารถใช้งานทรัพยากรที่ได้รับ เช่น เครื่องพิมพ์ เมื่อได้รับการจัดสรรเครื่องพิมพ์แล้ว โพรเซสสามารถพิมพ์ข้อมูลออกทางเครื่องพิมพ์ได้

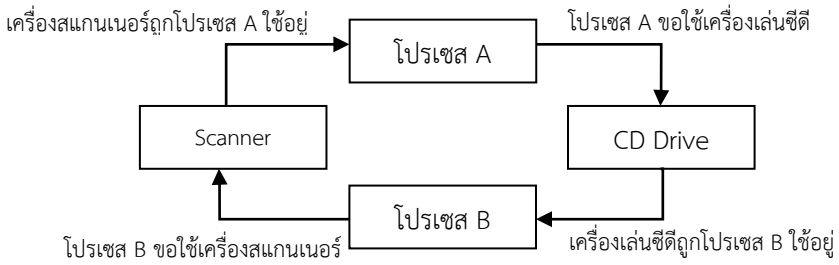
3. การคืน (Release) โพรเซสต้องคืนทรัพยากรที่ใช้เสร็จแล้วกลับสู่ระบบ เพื่อเปิดโอกาสให้โพรเซสอื่นที่ต้องการใช้ทรัพยากรนั้นได้รับการจัดสรรทรัพยากรต่อไป

5.2 การแก้ปัญหาส่วนวิกฤติ

บางครั้งเมื่อโพรเซสต้องการใช้ทรัพยากร และร้องขอทรัพยากรจากระบบ แต่ทรัพยากรที่ โพรเซสต้องการใช้ไม่ว่าง เนื่องจากโพรเซสอื่นกำลังใช้งานอยู่ ทำให้โพรเซสนั้น ๆ ต้องรอคอย และเป็นการรอคอยที่ไม่มีที่สิ้นสุด (โพรเซสไม่มีโอกาสได้รับการจัดสรรทรัพยากร) เหตุการณ์นี้คือ การติดตาย โปรแกรมหรือแอปพลิเคชั่นโดยทั่วไปเมื่อทำงานจะเกิดโพรเซสขึ้นมากมาย และโพรเซสเหล่านี้ส่วนใหญ่ต้องการใช้ทรัพยากรมากกว่า 1 สมมุติว่าการทำงานของโปรแกรมหนึ่ง ทำให้เกิดโพรเซส A และ โพรเซส B เริ่มแรกโพรเซส A ร้องขอใช้เครื่องสแกนเนอร์ (Scanner) และได้ใช้งาน

ส่วนโพรเซส B ถูกโปรแกรมขึ้นมาเพื่อเรียกใช้เครื่องเขียนแผ่นซีดี (CD writer) ก่อนและได้ใช้งาน ต่อมาโพรเซส A ต้องการใช้เครื่องเขียนแผ่นซีดีบ้าง จึงได้ร้องขอกับระบบ แต่ถูกปฏิเสธจนกว่าโพรเซส B ใช้งานเสร็จ และก็อาจจะเป็นความบังเอิญหรือความโชคร้ายของระบบปฏิบัติการที่โพรเซส B ได้ร้องขอ

สแกนเนอร์เพื่อใช้งานพร้อม ๆ กัน ณ จุดนี้เองที่โพรเซสทั้งสองจะถูกปฏิเสธให้ทำงาน และหยุดเพื่อรอทรัพยากรของอีกฝ่ายหนึ่งโดยไม่มีวันที่จะได้รับทรัพยากรของอีกฝ่ายเลย ดังในภาพที่ 5.1 ซึ่งเหตุการณ์นี้เองที่เรียกว่า วงจรอับ หรือ Deadlock

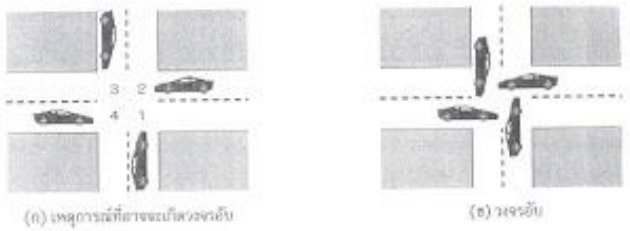


ภาพที่ 5.1 การแสดงโพรเซสสองโพรเซสเกิดวงจรอับ

5.3 ตัวอย่างของการติดตาย

ตัวอย่างง่าย ๆ ที่สามารถแสดงภาพของวงจรอับอยู่ในภาพที่ 5.2 แสดงภาพของเหตุการณ์ที่มีรถยนต์ 4 คันแล่นมาจากเส้นทาง 4 เส้นที่แตกต่างกัน เข้ามาสู่สี่แยกพร้อมกัน ณ เวลาเดียวกัน เราสามารถเปรียบเส้นทางทั้งสี่เส้นเป็นทรัพยากรที่ระบบจะต้องควบคุม และถ้ารถยนต์ทั้งสี่ต้องการแล่นตรงไปยังเส้นทางที่อยู่ตรงข้าม ระบบจะต้องทำหน้าที่จัดสรรทรัพยากรดังนี้

1. รถยนต์ที่วิ่งไปทิศเหนือจะต้องใช้เส้นทางที่ 1 และ 2
2. รถยนต์ที่วิ่งไปทิศตะวันตกจะต้องใช้เส้นทางที่ 2 และ 3
3. รถยนต์ที่วิ่งไปทิศใต้จะต้องใช้เส้นทางที่ 3 และ 4
4. รถยนต์ที่วิ่งไปทิศตะวันออกจะต้องใช้เส้นทางที่ 4 และ 1



ภาพที่ 5.2 ตัวอย่างภาพแสดงการเกิดวงจรอับ (Deadlock)
ที่มา: arthit operating system. Retrieved June 20, 2014 from http://arthit.84au.com/2013/08/15/11/

โดยปกติเมื่อขับรถเข้าไปในสี่แยก จะอนุญาตให้รถที่วิ่งมาทางด้านขวามือไปก่อน ซึ่งกฎดังกล่าวสามารถใช้งานได้ในกรณีที่มีรถเข้ามาจำนวน 2 หรือ 3 คัน เช่น ถ้ารถที่จะวิ่งไปทิศเหนือมาพร้อมกับรถที่จะวิ่งไปทิศตะวันตกพร้อมกัน รถที่วิ่งไปทิศเหนือจะต้องรอรถที่วิ่งไปทิศตะวันตกไปก่อน อย่างไรก็ตามถ้า

รถยนต์ทั้งสี่คันเข้ามาพร้อมกัน แต่ละคันจะต้องรอรถฝั่งขวามือก่อน (ตามกฎที่ตั้งไว้) ซึ่งเป็นการรอในลักษณะวนรอบโดยไม่มีที่สิ้นสุดและเกิดวงจรอับขึ้น แต่ถ้ารถทั้งสี่คันไม่ปฏิบัติตามกฎดังกล่าวและวิ่งเข้ามาในสี่แยก โดยที่รถแต่ละคันก็จองเส้นทางคนละ 1 เส้น (ดังในภาพที่ 5.2 (ข)) รถทั้งสี่จะไม่สามารถวิ่งต่อไปได้เพราะว่าเส้นทางที่ต้องการอีก 1 เส้น ได้ถูกจองไว้แล้วโดยรถคันอื่น และเข้าสู่วงจรอับอีกครั้งหนึ่ง

5.4 เงื่อนไขที่ทำให้เกิดวงจรอับ

วงจรอับ เป็นสถานการณ์ที่เราไม่ต้องการให้เกิดขึ้นในระบบ เพราะว่าเมื่อเกิดวงจรอับขึ้นมา จะทำให้ไม่มีโพรเซสใดสามารถทำงานจนเสร็จสมบูรณ์ได้ และทรัพยากรของระบบจะถูกครอบครองจนหมด ดังนั้นก่อนที่เราจะกล่าวลึกลงในรายละเอียดเกี่ยวกับการจัดการกับวงจรอับนั้น เราควรเรียนรู้เกี่ยวกับสาเหตุสำคัญที่ทำให้เกิดวงจรอับก่อน วงจรอับอาจจะเกิดขึ้นก็ต่อเมื่อเงื่อนไขทั้งสามข้อต่อไปนี้เกิดขึ้น

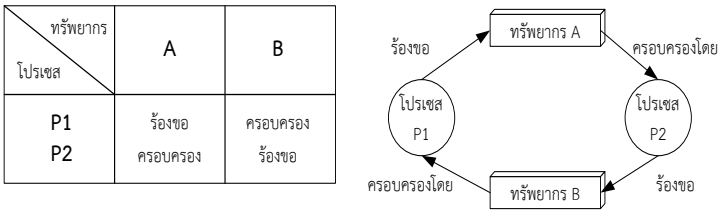
1. การไม่เกิดร่วม (Mutual exclusion) ในกรณีที่เป็นทรัพยากรที่ไม่สามารถใช้ร่วมกันได้ ทำให้ทรัพยากรถูกใช้งานได้เพียงครั้งละ 1 โพรเซส คือ ขณะเวลาหนึ่งจะมีเพียงโพรเซสเดียวที่ใช้งานทรัพยากรได้ ไม่สามารถมีโพรเซสอื่นใช้งานทรัพยากรพร้อมกันได้

2. การครอบครองและการรอใช้ทรัพยากร (Hold and Wait) โพรเซสที่ได้ครอบครอง (Hold) ทรัพยากรอยู่แล้ว ต้องการใช้ทรัพยากรอื่นเพิ่มเติม และร้องขอทรัพยากรที่มีสถานะไม่ว่าง ทำให้โพรเซสต้องรอ (Wait)

3. การไม่แย่งชิงทรัพยากร (No preemptive) โพรเซสที่รอใช้ทรัพยากรต่อจากโพรเซสอื่น (ที่กำลังใช้ทรัพยากรนั้นอยู่) จะต้องรอจนกว่าโพรเซสนั้น ๆ ทำงานเสร็จ และปลดปล่อยทรัพยากร โพรเซสไม่สามารถแย่งชิงทรัพยากรจากโพรเซสอื่นได้

4. การรอแบบวงกลม (Circulate wait) โพรเซสเกิดการรอเป็นวัฏจักร (P0, P1, P2, ..., Pn) โดย P0 รอทรัพยากรที่ถูกครอบครองโดย P1 P1 รอทรัพยากรที่ถูกครอบครองโดย P2 จนถึง Pn-1 รอทรัพยากรที่ถูกครอบครองโดย Pn และ Pn รอทรัพยากรที่ถูกครอบครองโดย P0 การเกิดติดตายจะต้องมีครบทั้ง 4 เงื่อนไข โดยเงื่อนไขทั้ง 4 ข้อนี้ ไม่ได้เป็นอิสระต่อกัน คือ ถ้าเกิดการรอแบบวงกลมแล้ว จะทำให้เกิดการครอบครองและรอเป็นต้น

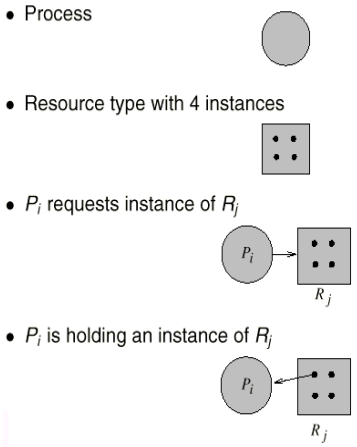
เงื่อนไขสามข้อแรกนั้นจำเป็น แต่ไม่เพียงพอต่อการทำให้เกิดวงจรอับ เงื่อนที่สี่แท้จริงแล้วเป็นผลที่เกิดจากเงื่อนไขทั้งสามข้อแรก นั่นคือเมื่อเงื่อนไขทั้งสามข้อแรกเกิดขึ้น ผลที่เกิดตามมาอาจจะทำให้เกิดวงจรรอคอย ซึ่งแท้จริงก็คือวงจรอับนั่นเอง วงจรรอคอยในภาพที่ 5.3 เป็นวงจรที่ระบบไม่สามารถแก้ได้ เพราะเงื่อนไขทั้งสามข้อแรกบังคับอยู่ ดังนั้นเงื่อนไขทั้งสี่ข้อจะต้องสมบูรณ์จึงจะทำให้เกิดวงจรอับได้



ภาพที่ 5.3 วงจรลูกโซ่ของโพรเซสที่ต่างรอคอยซึ่งกันและกัน

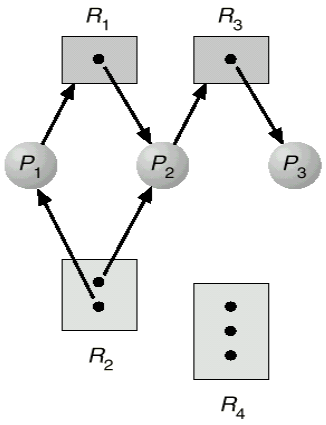
5.5 การกำหนดทรัพยากรด้วยกราฟ

การติดตายสามารถอธิบายได้ด้วยกราฟอย่างง่าย ๆ เรียกว่า System Resource-Allocation Graph กราฟนี้จะประกอบไปด้วยกลุ่มของ V (Vertices) และกลุ่มของ E (Edge) กลุ่มของ V แบ่งออกเป็น 2 ส่วนใหญ่คือ จุดของ P = {P1,P2,P3...,Pn} เซตนี้ประกอบด้วยโพรเซสที่กำลังทำงานในระบบ และจุดของ R = {R1,R2,R3...,Rm} เซตนี้ประกอบด้วยทรัพยากรทุกชนิดในระบบ การร้องขอสามารถแทนได้ด้วย Pi → Rj หมายถึงโพรเซส Pi ขอเข้าทำงานทรัพยากร Rj ส่วนการกำหนดให้ทรัพยากรบริการโพรเซสแทนได้ด้วย Rk → Pj หมายถึงทรัพยากร Rk กำลังถูกโพรเซส Pj ใช้งาน



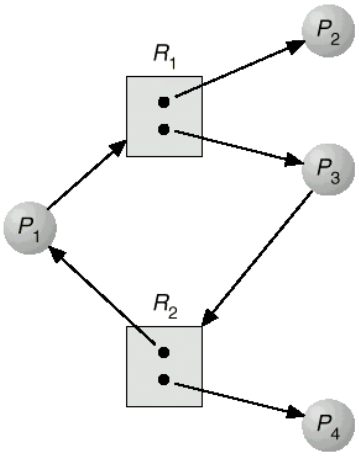
ภาพที่ 5.4 รูปแบบต่าง ๆ ของกราฟ

ลักษณะของกราฟที่ใช้อธิบายการติดตายมีรูปแบบดังภาพที่ 5.4 โดยใช้รูปวงกลมแทนโพรเซส รูปสี่เหลี่ยมแทนทรัพยากรที่มีในระบบ จุดที่อยู่บนช่องสี่เหลี่ยมเป็นตัวบ่งชี้ว่าทรัพยากรนี้สามารถถูกใช้งานได้พร้อมกันกี่โพรเซส ในรูปแบบต่อมาคือ การขอเข้าใช้งานของโพรเซส i จะใช้ลูกศรชี้เข้าหาทรัพยากร (กล่องสี่เหลี่ยม) สุดท้ายสามารถแสดงว่าทรัพยากรนี้ได้ถูกโพรเซส i ใช้งานด้วยการชี้ลูกศรออกจากกล่องสี่เหลี่ยมโดยจะพุ่งออกจากจุดที่โพรเซสนั้น ๆ กำลังใช้งาน



ภาพที่ 5.5 การกำหนดทรัพยากรด้วยกราฟ

จากภาพที่ 5.5 เป็นลักษณะของกราฟที่ไม่เกิดลูป โดยมีการทำงานคือ P1 รอเข้าใช้งานทรัพยากร R1 ในขณะที่ทรัพยากร R1 ให้บริการโพรเซส P2 แล้ว P2 ก็กำลังรอเข้าทำงานทรัพยากร R3 ซึ่งกำลังให้บริการโพรเซส P3 ทรัพยากร R2 กำลังให้บริการทั้งโพรเซส P1 และ P2 ส่วนทรัพยากร R4 ว่างไม่มีเรียกใช้งาน ซึ่งสามารถแทนได้ด้วยลักษณะการเคลื่อนดังนี้ P1 → R1 → P2 → R3 → P3



ภาพที่ 5.6 การกำหนดทรัพยากรด้วยกราฟ

จากภาพที่ 5.6 เป็นตัวอย่างของกราฟที่มีการเรียกใช้ทรัพยากรจากโพรเซสจนกระทั่งเกิดลูปขึ้นในรูปกราฟ จากลักษณะของกราฟทำให้ได้การทำงานดังนี้ P1 → R1 → P2, P1 →R1 → P3 → R2 → P1, P1 →R1 → P3 → R2 → P4

พื้นฐานของการติดตาย ถ้ากราฟไม่เป็นลูปจะไม่เกิดติดตาย แต่ถ้ากราฟมีลูปจะสามารถแยกได้ 2 กรณีคือ ถ้าทรัพยากรมีให้ใช้งานได้ทีละตัวจะเกิดการติดตาย แต่ถ้าทรัพยากรหนึ่งตัวสามารถใช้ได้พร้อมกันหลายโพรเซส จะมีโอกาสที่จะเกิดการติดตายหรือไม่เกิดก็เป็นได้

เราได้เรียนรู้การทำงานของกราฟการจัดสรรทรัพยากรในลักษณะที่ทรัพยากรแต่ละชนิดมีทรัพยากรเพียง 1 ตัว (Single resource for each type) แต่กราฟนี้สามารถใช้งานในกรณีที่ทรัพยากรแต่ละชนิดมีทรัพยากรหลาย ๆ ตัวก็ได้เช่นกัน หลังจากนี้เราจะเรียนรู้ถึงวิธีการป้องกันไม่ให้ระบบเกิดวงจรอับ ซึ่งโดยทั่วไปแล้ว การจัดการวงจรอับนั้นมีกลยุทธ์อยู่ 4 วิธี ที่สามารถนำมาใช้ได้คือ

1. ทำการป้องกันไว้ก่อน โดยไม่ให้หนึ่งในเงื่อนไขทั้งสี่ข้อของการทำให้เกิดวงจรอับเกิดขึ้น
2. ทำการหลีกเลี่ยง โดยการจัดสรรทรัพยากรให้ถูกต้อง
3. การตรวจพบและแก้ไขคืน จะอนุญาตให้วงจรอับเกิดขึ้น และค่อยทำการเช็คหาและแก้ไขมัน
4. ไม่ต้องสนใจปัญหาใด ๆ เลย ในบางครั้งถ้าเราไม่สนใจปัญหาที่จะเกิดขึ้น ปัญหาก็อาจจะไม่เกิดขึ้นกับคุณก็ได้

5.6 การป้องกันการติดตาย

วิธีการป้องกันการติดตายนั้นอาจจะกล่าวได้ง่าย ๆ โดยการออกแบบให้ระบบมีการป้องกันการ

เกิดวงจรอับไว้ก่อน จากที่เราได้กล่าวมาในหัวข้อที่ 5.1 เกี่ยวกับเงื่อนไขที่ทำให้เกิดวงจรอับ ดังนั้นการป้องกันการเกิดวงจรอับสามารถแบ่งได้เป็น 2 กลุ่ม ได้แก่

1. การป้องกันการเกิดวงจรอับทางอ้อม คือ การป้องกันการเกิดเงื่อนไขสามข้อแรกที่อาจจะทำให้เกิดวงจรอับได้

2. ที่ว่าเงื่อนไขทั้งสี่ข้อจะต้องเกิดขึ้นพร้อมกัน จึงจะทำให้เกิดวงจรอับได้ ดังนั้นถ้าป้องกันไม่ให้เงื่อนไขใดเงื่อนไขหนึ่งเกิดขึ้น เราก็จะสามารถป้องกันการเกิดวงจรอับได้ โดยจะพิจารณาในแต่ละเงื่อนไขดังต่อไปนี้

5.6.1 การใช้ทรัพยากรร่วมกันได้ (Mutual Exclusion Prevention)

ปัญหาในข้อนี้คือ การที่ทรัพยากรในระบบไม่อนุญาตให้โพรเซสหลาย ๆ โพรเซสใช้งานพร้อม ๆ กันได้ ดังนั้นการแก้ปัญหาข้อนี้ ระบบปฏิบัติการจะต้องจัดการให้โพรเซสในระบบสามารถใช้งานทรัพยากรเหล่านั้นร่วมกันได้ ตัวอย่างของทรัพยากรบางประเภท เช่น ไฟล์ข้อมูล ระบบปฏิบัติการอาจจะอนุญาตให้โพรเซสหลาย ๆ โพรเซสเข้าถึงไฟล์ข้อมูลนั้นได้ โดยมีการกำหนดให้มีการเข้าถึงเป็นแบบอ่านได้อย่างเดียว (Read only) ก็จะทำให้ไฟล์นั้นสามารถใช้งานร่วมกันได้ และอนุญาตให้โพรเซสเพียงโพรเซสเดียวเท่านั้นที่ทำการเขียนได้

อย่างไรก็ตาม การป้องกันการเกิดวงจรอับในระบบโดยใช้วิธีนี้ไม่สามารถทำได้เสมอไป เพราะว่ายังมีทรัพยากรบางตัวที่ไม่สามารถใช้งานร่วมกันกับหลาย ๆ โพรเซสพร้อมกันได้ เช่น เครื่องพิมพ์ เป็นต้น แต่เราก็อาจจะใช้วิธี Spooling เข้ามาช่วยในการจัดการได้ โดยใช้พื้นที่ดิสก์เป็นตัวรับเอาต์พุตแทนเครื่องพิมพ์ แล้วค่อยส่งข้อมูลไปให้เครื่องพิมพ์ ซึ่งวิธีนี้จะทำให้โพรเซสหลาย ๆ ตัวสามารถใช้เครื่องพิมพ์นี้ร่วมกันได้

ตัวอย่าง 5.6.1 ระบบคอมพิวเตอร์มีเครื่องพิมพ์ 1 เครื่อง ถ้าระบบมี 3 โพรเซส คือ โพรเซส A, B และ C ที่ทำงานร่วมกัน และทั้ง 3 โพรเซสร้องขอเครื่องพิมพ์พร้อมกัน โพรเซส A ได้รับจัดสรรเครื่องพิมพ์ ดังนั้นคำร้องขอใช้เครื่องพิมพ์ของโพรเซส B และ C ต้องถูกส่งเข้าที่เก็บพักบนดิสก์

เมื่อโพรเซส A ใช้เครื่องพิมพ์เสร็จเรียบร้อย จะปล่อยเครื่องพิมพ์กลับคืนให้ระบบ ระบบจะเรียกโพรเซสต่อไปที่รออยู่บนที่เก็บพักให้ได้รับจัดสรรเครื่องพิมพ์ต่อไป จะเห็นว่า ทั้ง 3 โพรเซสสามารถเรียกใช้เครื่องพิมพ์ได้พร้อม ๆ กัน

5.6.2 การป้องกันการถือครองและรอคอย (Hold and Wait Prevention)

เงื่อนไขของการถือครองและรอคอยนั้นสามารถป้องกันได้ โดยเราจะอนุญาตให้โพรเซสร้องขอทรัพยากรที่ต้องการทั้งหมดก่อน และจะไม่อนุญาตให้โพรเซสนั้นทำงานจนกว่าจะได้รับทรัพยากรที่ร้องขอไปพร้อมกันทั้งหมดก่อน อย่างไรก็ตาม วิธีการป้องกันแบบนี้จะทำให้ระบบปฏิบัติการทำงานอย่างไม่มีประสิทธิภาพ เพราะอย่างแรกโพรเซสจะต้องถือครองทรัพยากรเป็นเวลานาน ในขณะที่รอให้ตัวเองได้รับทรัพยากรทั้งหมดก่อนจึงจะทำงานได้ทั้ง ๆ ที่โพรเซสเหล่านั้นเพียงได้รับทรัพยากรบางตัวก็สามารถทำงานก่อนได้เลย อย่างที่สองบางทรัพยากรที่ถูกครอบครองโดยโพรเซสอาจจะยังไม่ได้ถูกใช้งาน และโพรเซสอื่นไม่สามารถเรียกนำไปใช้ได้ด้วย

ในทางปฏิบัติ เราสามารถพบปัญหาที่ไม่สามารถแก้ไขได้ เช่น ในกรณีที่เราใช้โปรแกรมที่มีโปรแกรมย่อย ๆ หลายโปรแกรม หรือโปรแกรมที่มีโครงสร้างที่สามารถทำงานหลาย ๆ อย่างพร้อมกันได้ (Multithreaded structure) โปรแกรมเหล่านี้จะต้องทำการตรวจสอบด้วยว่าถ้ามีการร้องขอทรัพยากรจากโปรแกรมย่อย ๆ ภายในพร้อมกัน ทรัพยากรทั้งหมดภายในระบบจะเพียงพอหรือไม่ มิฉะนั้นวงจรอับก็จะเกิดขึ้นได้เช่นกัน

ตัวอย่าง 5.6.2 พิจารณาโพรเซสที่ทำงานในการสำเนาไฟล์ข้อมูลจากเทปไปยังดิสก์ แล้วพิมพ์ออกทางเครื่องพิมพ์ ถ้าต้องให้โพรเซสมีการขอใช้ทรัพยากรก่อนที่จะเริ่มทำงานจริง ดังนั้นโพรเซสต้องเริ่มขอไปยังเทป แล้วดิสก์ สุดท้ายก็เป็นเครื่องพิมพ์ ดังนั้นเครื่องพิมพ์จึงต้องถูกจับจองเอาไว้ให้โพรเซสนี้ตลอดระยะเวลาในการทำงานทั้งหมดของโพรเซส ลักษณะเช่นนี้เกิดเป็นข้อเสียทางด้าน Resource Utilization

ส่วนวิธีที่สองยอมให้โพรเซสร้องขอเพียงแค่ช่วงของการขอเทปและดิสก์ เพื่อสำเนาข้อมูลได้ เนื่องจากโพรเซสเริ่มแรกยังไม่มีการใช้งานทรัพยากรใด แต่เมื่อกำลังใช้ในการสำเนา โพรเซสนั้นจะไม่มีสิทธิในการขอใช้เครื่องพิมพ์จนกว่าจะสำเนาเสร็จ เมื่อสำเนาเสร็จแล้วจึงต้องขอใช้ดิสก์กับเครื่องพิมพ์เพื่อสำเนาข้อมูลที่ต้องการพิมพ์มายังเครื่องพิมพ์นั้น ถ้าในระหว่างนั้นในโพรเซสอื่นอาจใช้งานเครื่องพิมพ์ หรือทรัพยากรเทปหรือดิสก์ ทำให้ต้องรอข้อมูลอาจสูญหาย หรือเกิดการรอในแต่ละช่วงของการร้องขอลักษณะเช่นนี้เป็นข้อเสียที่เกี่ยวเนื่องกับ Starvation หมายถึง สภาวะอดตาย

5.6.3 ยอมให้มีการแทรกกลางคัน (Preemptable)

เงื่อนไขนี้เราสามารถทำการป้องกันได้หลาย ๆ วิธี และเงื่อนไขนี้จะยอมให้โพรเซสสามารถแย่งชิงทรัพยากรที่ถูกครอบครองโดยโพรเซสอื่นได้ อย่างแรกคือ ถ้าโพรเซสกำลังถือครองทรัพยากรหนึ่งอยู่ เราจะให้ระบบป้องกันไม่ให้โพรเซสนั้นทำการร้องขอทรัพยากรอื่นใดอีก จนกว่าโพรเซสนั้นจะปลดปล่อยทรัพยากรที่ตัวเองครอบครองเสียก่อน และในบางกรณีถ้าโพรเซสต้องการทรัพยากรเพิ่ม ระบบอาจจะกำหนดให้โพรเซสปล่อยทรัพยากรที่ตัวเองครอบครองอยู่ก่อน และค่อยทำการร้องขอทรัพยากรเหล่านั้นอีกครั้งพร้อมกับทรัพยากรที่ต้องการเพิ่มเติม

ตัวอย่าง 5.6.3(1) เมื่อเริ่มต้นทำงาน โพรเซส A ได้รับจัดสรรทรัพยากร R1 และ R2 ในเวลาต่อมาโพรเซสร้องขอทรัพยากร R3 ระบบตรวจสอบแล้วพบว่า ทรัพยากร R3 ถูกครอบครองโดยโพรเซส B ดังนั้น โพรเซส A ต้องปล่อยทรัพยากร R1 และ R2 คืนให้ระบบ

วิธีการที่สองถ้ามีโพรเซสร้องขอทรัพยากรบางอย่างจากระบบ ให้ระบบตรวจสอบว่าทรัพยากรที่ถูกร้องขอนั้นว่างหรือไม่ กรณีที่ทรัพยากรนั้นไม่ว่าง ระบบต้องตรวจสอบต่อไปว่า ทรัพยากรนั้นถูกครอบครองโดยโพรเซสใด และโพรเซสนั้นกำลังรอทรัพยากรชนิดอื่นหรือไม่ (โพรเซสอยู่ในสถานะรอ)

ถ้าใช่ให้ระบบแย่งชิงทรัพยากรนั้นจากโพรเซสที่กำลังครอบครอง และนำทรัพยากรนั้นมอบให้โพรเซสที่ร้องขอ ถ้าไม่ใช่ก็ให้โพรเซสที่ร้องขอทรัพยากรนั้น รอจนกว่าทรัพยากรที่ต้องการว่าง (โพรเซสไม่สามารถแย่งชิงทรัพยากรได้) วิธีการที่สองจะป้องกันการเกิดวงจรอับได้ก็ต่อเมื่อ โพรเซสทั้งสองมีสิทธิและความสำคัญที่แตกต่างกัน

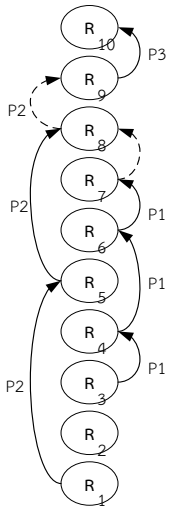
ตัวอย่าง 5.6.3(2) โพรเซส A ร้องขอทรัพยากร R1 ให้พิจารณาว่า ระบบจะจัดสรรทรัพยากรให้โพรเซส A หรือไม่ ในการพิจารณานั้นระบบต้องตรวจสอบก่อนว่า ทรัพยากร R1 ว่างหรือไม่ ถ้าทรัพยากร R1 ว่าง

ระบบจะจัดสรรทรัพยากร R1 ให้โพรเซส A ถ้าทรัพยากร R1 ไม่ว่าง ระบบจะตรวจสอบต่อไปว่าทรัพยากร R1 ถูกครอบครองโดยโพรเซสใด

ถ้า R1 ถูกครอบครองโดยโพรเซส B และโพรเซส B ร้องขอทรัพยากร R2 ซึ่งไม่ว่าง (โพรเซส B อยู่ในสถานะรอ) ให้ระบบแย่งทรัพยากร R1 จากโพรเซส B และจัดสรรทรัพยากร R1 ให้กับโพรเซส A ถ้า R1 ถูกครอบครองโดยโพรเซส B และโพรเซส B ไม่ได้ร้องขอทรัพยากรอื่น ๆ ที่ไม่ว่าง (โพรเซส B ไม่ได้อยู่ในสถานะรอ) ระบบจะไม่สามารถแย่ง R1 จากโพรเซส B มาให้โพรเซส A ดังนั้น โพรเซส A ต้องรอจนกว่าทรัพยากร R1 ว่าง

5.6.4 การป้องกันการเกิดวงจรรอคอย (Circular wait protection)

เพื่อไม่ให้เกิดการรอแบบวงกลม ระบบจะกำหนดหมายเลขประจำให้แต่ละทรัพยากร และเมื่อมีการร้องขอทรัพยากรจากโพรเซสใด ๆ จะต้องเป็นการร้องขอเรียงตามลำดับหมายเลขประจำทรัพยากรจากน้อยไปมากเสมอ จากภาพ โพรเซส P1 ได้ครอบครองทรัพยากร R3, R4, R6, R7 และได้ขอใช้ทรัพยากร R8 ซึ่งจะไม่เกิดการรอแบบวงกลม เพราะการขอใช้ทรัพยากรเป็นการขอแบบเรียงลำดับหมายเลขทรัพยากรจากน้อยไปหามาก และถ้าโพรเซสต้องการทรัพยากรที่มีหมายเลขทรัพยากรก่อนหน้าโพรเซสจะต้องคืนทรัพยากรคืนสู่ระบบทุกตัวเสียก่อน



ภาพที่ 5.7 ปฏิเสธการรอแบบวงกลม

5.7 การหลีกเลี่ยงการติดตาย

จากข้อที่ผ่านมาเป็นการป้องกันการเกิดการติดตาย โดยการจัดการกับสัญญาณที่ร้องขอเพื่อใช้ทรัพยากร แต่อาจส่งผลให้การใช้งานระบบ หรือประสิทธิภาพในการทำงานลดลง ดังนั้นจึงมีวิธีที่หลีกเลี่ยงการติดตายโดยพิจารณาจากข้อมูลของทรัพยากรที่ถูกร้องขอ เช่น ในระบบที่มีเทป และเครื่องพิมพ์อย่างละตัว ดังนั้นเราอาจจัดสรรให้โพรเซส P เข้าใช้งานเทปก่อนแล้วก็ใช้งานเครื่องพิมพ์ ในขณะที่โพรเซส Q ใช้งานเครื่องพิมพ์ก่อนแล้วค่อยใช้เทป ดังนั้นนอกจากการร้องขอแล้วตรวจว่าทรัพยากรว่างหรือไม่ จึงต้อง

มีการดูด้วยว่าขณะนั้นทรัพยากรนั้นถูกใช้โดยใครแล้วโพรเซสไหนจะใช้อะไรก่อนหลัง นอกจากนี้อาจมีข้อมูลที่มากที่สุดที่ขอใช้ทรัพยากรชนิดนั้น รูปแบบอัลกอริทึมของการหลีกเลี่ยงการติดตายจะมีการตรวจสอบสถานะของการใช้ทรัพยากร เพื่อให้แน่ใจว่าจะไม่เกิดการรอแบบลูปอย่างสม่ำเสมอตลอดเวลา ความหมายคือต้องการให้ระบบอยู่ในสถานะที่ปลอดภัย (Safe state) ดังนั้นสถานะของการใช้ทรัพยากรถูกกำหนดด้วยจำนวนของทรัพยากรที่ถูกใช้งานและจำนวนทรัพยากรที่มีอยู่ในระบบ

5.7.1 สถานะที่ปลอดภัย (Safe State)

เมื่อโพรเซสร้องขอทรัพยากรที่มีอยู่ ระบบต้องตัดสินว่าถ้าให้ใช้ทรัพยากรแล้วระบบจะยังคงอยู่ในสถานะที่ปลอดภัยหรือไม่ (Safe state หมายถึง สถานะที่ทรัพยากรสามารถให้บริการได้ทุกโพรเซส) ลักษณะของ Sequence <P1, P2, ...,Pn> จะยังคงดำเนินได้ก็ต่อเมื่อทรัพยากรที่ Pi ขอทำงาน ยังคงทำงานได้อย่างน่าพอใจ ซึ่งจะกำหนดโดยทรัพยากรที่มีในขณะนั้นบวกกับทรัพยากรที่ถูกใช้งานอยู่ทั้งหมดของ Pj โดยที่ j < i นั้นหมายถึง ถ้าทรัพยากรที่ Pi ต้องการยังไม่ว่างให้ใช้งาน Pi ต้องสามารถรอได้จนกว่า Pj ทำงานเสร็จ แล้วเมื่อ Pj ทำงานเสร็จ Pi สามารถเข้าทำงานทรัพยากรที่ต้องการนั้นแล้วก็ออกไปเมื่อใช้งานเสร็จสิ้น เมื่อโพรเซส Pi ทำงานแล้วออกจากทรัพยากรนั้น Pi+1 สามารถที่จะใช้งานทรัพยากรนั้นได้

ดังนั้นความจริงของสถานะที่ปลอดภัยจะไม่เกิดการติดตาย แต่ถ้าเป็นสถานะที่ไม่ปลอดภัยอาจเกิดการติดตายได้ ดังนั้นจึงต้องทำให้ระบบอยู่ในสถานะที่ปลอดภัย และไม่ยอมให้ระบบเข้าสู่ภาวะที่ไม่ปลอดภัยเพื่อหลีกเลี่ยงการติดตาย ตัวอย่างเช่น ถ้าระบบมีเทป 12 ตัวและมีโพรเซส P0, P1 และ P2 เมื่อ P0 ต้องการใช้งานเทป 10 ตัว P2 ต้องการใช้ 4 ตัว P3 ต้องการใช้อีก 9 ตัว สมมติว่าในเวลาที่ t0 โพรเซส 0 กำลังใช้งานเทป 5 ตัวอยู่ โพรเซส 1 ใช้งานอยู่ 2 และโพรเซส 2 ใช้งานอยู่ 2 ดังนั้นจะมีทรัพยากรเทปว่างอยู่อีก 3 ดังนั้นในเวลาที่ t0 ระบบยังอยู่ในภาวะปลอดภัย แต่ก็มีโอกาสที่จะเข้าสู่ภาวะไม่ปลอดภัยได้ สมมติที่เวลา t1 โพรเซส 2 มีการขอใช้งานเทปเพิ่มอีก 4 ตัว หรือมากกว่านั้น ดังนั้นระบบจะอยู่ในภาวะที่ไม่ปลอดภัย มีเพียงโพรเซส 1 ที่จะช่วยได้คือต้องปล่อยทรัพยากรเทปให้ว่าง เพื่อให้มีทรัพยากรเหลือว่างพอให้โพรเซส 2 ใช้งานได้

5.7.2 อัลกอริทึมการกำหนดทรัพยากรด้วยกราฟ

Claim Edge Pi → Rj เป็นตัวกำหนดว่าโพรเซส Pi อาจทำการขอใช้ทรัพยากร Rj ซึ่งแทนได้ด้วยจุดปะ ดังนั้นเส้นจุดปะจะถูกเปลี่ยนให้กลายเป็นการขอใช้งานจริง (แทนด้วยเส้นทึบ) เมื่อโพรเซสนั้นทำการขอใช้ทรัพยากรจริง ดังนั้นเมื่อทรัพยากรบริการโพรเซสเรียบร้อยแล้วก็จะเปลี่ยนกลับเป็นเส้นปะ Claim Edge เหมือนเดิม สมมติถ้ามีโพรเซส 2 ขอใช้งานทรัพยากร 2 และ 1 ในขณะนั้นมีโพรเซส 1 กำลังร้องขอใช้งานด้วย แล้วทรัพยากร 1 ให้บริการโพรเซส 1 อยู่ ดังนั้นระบบมีสิทธิที่จะยอมให้โพรเซส 2 เข้าใช้งานทรัพยากร 2 ได้ แต่พบว่าถ้ายอมให้มีการใช้ทรัพยากร 2 จะทำให้ระบบอยู่ในภาวะที่ไม่ปลอดภัย เนื่องจากจะเกิดการลูปรอหรือการติดตายทันที

5.7.3 อัลกอริทึมของนายธนาคาร (Banker's Algorithm)

ในระบบที่ทรัพยากร 1 ตัวสามารถให้บริการได้พร้อมกันหลายโพรเซส การใช้กราฟแทนการใช้งานทรัพยากรจะไม่เหมาะสม จึงมีการเสนออัลกอริทึมของนายธนาคาร ซึ่งเป็นอัลกอริทึมที่ใช้งานได้จริงใน

ระบบธนาคารที่ว่าธนาคารจะไม่จ่ายเงินที่มีอยู่ให้ตามความต้องการของลูกค้าทั้งหมดได้เป็นเวลานาน (หมายถึง มีคนถอนออกอย่างเดียว ธนาคารจะไม่สามารถอยู่ได้) ดังนั้นจึงต้องมีระบบเพื่อกำหนดจำนวนสูงสุดที่จะสามารถให้บริการได้ จำนวนนี้อาจไม่จำเป็นต้องเป็นจำนวนทรัพยากรทั้งหมดที่มีอยู่ในระบบ เมื่อผู้ใช้ขอใช้กลุ่มของทรัพยากร ระบบต้องทำการตรวจสอบว่าถ้าให้ใช้แล้วระบบจะยังคงอยู่ในภาวะปลอดภัยหรือไม่ ถ้าไม่ปลอดภัยการให้ใช้ทรัพยากรต้องรอจนกว่าจะมีผู้ใช้ทรัพยากรรายอื่นที่ใช้แล้วกลับเข้าสู่ระบบ โครงสร้างของระบบนายธนาคารมีดังนี้ กำหนดให้มี n โพรเซส มีทรัพยากรในระบบทั้งหมด m ตัวมีข้อมูลดังนี้

1. Available : เป็นเวกเตอร์ของขนาดที่ใช้ชี้จำนวนของทรัพยากรที่สามารถใช้งานได้ Available [j]=k ดังนั้น k คือจำนวนที่ทรัพยากรชนิด j จะสามารถให้บริการได้

2. Max : เป็นค่าเมตริกซ์ขนาด n x m ที่กำหนดความต้องการสูงสุดของแต่ละโพรเซส ถ้า Max[i,j] = k แล้วโพรเซส i อาจต้องใช้ทรัพยากร j สูงถึง k บริการ

3. Allocation : เป็นเมตริกซ์ขนาด n x m ที่กำหนดจำนวนของทรัพยากรในแต่ละชนิดที่ให้บริการแต่ละโพรเซสอยู่ได้ ถ้า Allocation[i,j] = k หมายถึงโพรเซส i กำลังใช้งานทรัพยากรชนิด j อยู่เป็นจำนวน k บริการ

4. Need : เมตริกซ์ขนาด n x m เพื่อบ่งบอกจำนวนทรัพยากรที่เหลือที่ยังต้องการใช้ของแต่ละโพรเซส เช่น Need[i,j] = k หมายถึงโพรเซส i ยังคงต้องการใช้งานทรัพยากร j อยู่อีก k บริการ พบว่า Need[i,j] = Max[i,j] – Allocation[i,j]

5.7.4 อัลกอริทึมที่ปลอดภัย

อัลกอริทึมที่หาได้ว่าระบบปลอดภัยหรือไม่ สามารถทำได้โดยกำหนดให้ work และ finish เป็นเวกเตอร์ที่มีขนาด n x m โดยเริ่มทำงานตามลำดับดังนี้

ขั้นตอนที่ 1 Work = Available และ Finish[i] = false โดยที่ i มีค่าตั้งแต่ 0, 1, 2...,n-1

ขั้นตอนที่ 2 หาค่า P_i ทั้ง 2 ฟังก์ชัน คือ Finish[i] == false , $Need_i \leq$ Work ถ้าไม่ตามเงื่อนไข ข้ามไปยังขั้นตอนที่ 4

ขั้นตอนที่ 3 Work = Work + Allocation, Finish[i] = true กลับไปยังขั้นที่ 2 ถ้า

ขั้นตอนที่ 4 ถ้า Finish [i] == true สาหรับทุกๆ i, แสดงว่าระบบอยู่ ใน Safe state เวลาในการใช้ทำงานอัลกอริทึมนี้เท่ากับ $m \times n^2$

5.7.5 อัลกอริทึมของการขอใช้ทรัพยากร

กำหนดให้ $Request_i$ เป็นเวกเตอร์ของการขอใช้งานสำหรับโพรเซส i ถ้า $request_i[j] = k$ หมายถึงโพรเซส i ต้องการทำงาน k บริการจากทรัพยากร j เมื่อมีการขอใช้งานทรัพยากรโดยโพรเซส i ก็จะเกิดการทำงานดังต่อไปนี้

1. ถ้า $Request_i \leq Need_i$ ไปยังขั้นตอนที่ 2 นอกจากนั้น ให้แสดงข้อความเตือน เนื่องจาก โพรเซสมีการใช้ทรัพยากรมากกว่าที่คาดการณ์ไว้

2. ถ้า $Request_i \leq Available$ ไปยังขั้นตอนที่ 3 นอกจากนี้โพรเซส i ต้องรอเนื่องจากทรัพยากรไม่มีให้ใช้งาน

3. มีการตั้งระบบลวงเพื่อใช้ทรัพยากรที่โพรเซส i ขอใช้ โดยการใส่ค่าในตัวแปรต่อไปนี้

$Available = Available - Request;$

$Allocation_i = Allocation_i + Request_i;$

$Need_i = Need_i - Request_i;$

ถ้าผลของการกำหนดค่าการให้ใช้ทรัพยากรออกมา พบว่าระบบอยู่ในภาวะปลอดภัย ก็จะจบสิ้นการทำงานแล้วยอมให้โพรเซส i ใช้งานทรัพยากรนั้นจริง ๆ อย่างไรก็ตามถ้าผลออกมาว่าไม่ปลอดภัย โพรเซส i ต้องรอ $Request_i$ แล้วก็จะกลับไปสู่ค่าสถานะเก่า

ตัวอย่าง 5.7.5 ใช้อัลกอริทึมของนายธนาคาร พิจารณาระบบที่มี 5 โพรเซสในการทำงาน P0 – P4 และมีทรัพยากรให้ใช้งานได้ 3 ตัว คือ A, B และ C ทรัพยากร A มีให้ใช้ได้ถึง 10 ตัว ทรัพยากร B ใช้ได้ถึง 5 ตัว และทรัพยากร C ใช้ได้ถึง 7 ตัว สมมติว่าที่เวลา t0 เกิดเหตุการณ์ดังภาพที่ 5.8

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

ภาพที่ 5.8 ข้อมูลของระบบที่ใช้งาน

ดังนั้นจะสามารถหา Need หาได้จาก Max – Allocation จะได้ดังภาพที่ 5.9 และเมื่อใช้อัลกอริทึมหาภาวะปลอดภัย พบว่าระบบอยู่ในสถานะปลอดภัย ปลอดภัย เนื่องจากกระบวนการอาจทำงานได้ตามลำดับ (P1, P3 , P4 , P0 , P2) ซึ่งเป็นไปตามเงื่อนไขของสถานะปลอดภัย

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Work = Available ; Work = (3,3,2)
Work = Work + Allocation ; Finish[i] = ture ;
P_1 : Finish[1] = ture ; Work = (3,3,2) + (2,0,0) = (5,3,2)
P_3 : Finish[3] = ture ; Work = (5,3,2) + (2,1,1) = (7,4,3)
P_4 : Finish[4] = ture ; Work = (7,4,3) + (0,0,2) = (7,4,5)
P_0 : Finish[0] = ture ; Work = (7,4,5) + (0,1,0) = (7,5,5)
P_2 : Finish[2] = ture ; Work = (7,5,5) + (3,0,2) = (10,5,7)

ภาพที่ 5.9 ค่าของ Need = Max – Allocation ของแต่ละโพรเซส และการหาลำดับความปลอดภัย

สมมติให้โพรเซส 1 มีการขอใช้งานทรัพยากรเพิ่ม 1 ตัว ของ A และจากทรัพยากร C อีก 2 ตัว ดังนั้น $Request_1$ = (1,0,2) ซึ่งสามารถตรวจสอบได้จาก

1. $Request_1$ =< $Need_1$: (1,0,2) <= (1,2,2) เป็นจริง
2. $Request_1$ =< Available : (1,0,2) <= (3,3,2) เป็นจริง

ดังนั้นจึงต้องทำการสร้างระบบจำลองขึ้นมาเพื่อตรวจสอบภาวะของระบบดังภาพที่ 5.10 หลังจากนั้นไปเข้าไปทำอัลกอริทึมที่ปลอดภัยเพื่อหาลำดับความปลอดภัย และได้ลำดับออกมาดังนี้ <P1, P3, P4, P0, P2> แต่เราพบว่าถ้าโพรเซส P4 มีการขอใช้ทรัพยากร (3,3,0) จะไม่มีให้ใช้งานได้ และถ้า P0 ขอใช้งาน (0,2,0) ก็จะใช้ไม่ได้เช่นกันเนื่องจากจะทำให้อยู่ในภาวะไม่ปลอดภัย

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

ภาพที่ 5.10 ระบบจำลองที่สร้างเพื่อตรวจสอบภาวะของระบบ

5.8 การตรวจจับการติดตาย

ระบบต้องมีอัลกอริทึมที่สามารถตรวจสอบภาวะของระบบว่าจะเกิดการติดตายหรือไม่ (การตรวจจับ) อัลกอริทึมที่จะกู้ระบบจากการติดตาย

5.8.1 กรณีที่มี 1 บริการในทรัพยากร 1 ตัว

ถ้าทุกทรัพยากรสามารถใช้บริการได้เพียง 1 โพรเซสแล้ว ดังนั้นสามารถกำหนดอัลกอริทึมของการตรวจจับการติดตายด้วยการใช้กราฟแสดงทรัพยากรที่เรียกว่ากราฟ Wait-for การติดตั้งดูแลการใช้กราฟสามารถทำได้โดยกำหนดให้โหนดแทนโพรเซส สัญลักษณ์ Pi → Pj หมายถึงโพรเซส i กำลังรอโพรเซส j โดยจะทำการแปลงจากกราฟแทนทรัพยากรเป็น wait-for กราฟที่มีแต่โพรเซส จากนั้นจึงอ้างใช้อัลกอริทึม เพื่อทำการค้นหาตรวจดูว่ากราฟที่ได้เกิดเป็นลูปขึ้นหรือไม่ ซึ่งใช้ order n2 โดยที่ n คือจำนวนของเส้นตรงที่อยู่ในกราฟ

5.8.2 กรณีที่สามารถให้บริการมากกว่า 1 ในทรัพยากร 1 ตัว

ลักษณะจะคล้ายกับอัลกอริทึมของนายธนาคาร โดยมีโครงสร้างดังนี้

1. Available : เป็นเวกเตอร์ของขนาดที่ใช้ชี้จำนวนของทรัพยากรที่สามารถใช้งานได้

2. Allocation : เป็นเมตริกซ์ขนาด n x m ที่กำหนดจำนวนของทรัพยากรในแต่ละชนิดที่ให้บริการแต่ละโพรเซสอยู่ได้

3. Request : เป็นเวกเตอร์ขนาด n x m เพื่อบ่งบอกการร้องขอทรัพยากรของแต่ละโพรเซสในขณะนั้น เช่น Request[i,j] = k หมายถึงโพรเซส i กำลังขอใช้ทรัพยากรจาก j เป็นจำนวน k บริการ

อัลกอริทึมของการตรวจจับมีดังนี้

1. ให้ Work และ Finish เป็นเวกเตอร์ที่มีขนาด n และ m เริ่มค่าที่ work = available สำหรับทุกค่าของ i ถ้า Allocation ≠ 0 แล้ว Finish[i] = false นอกนั้นค่า Finish[i] = true
2. ทำการหา i ดังต่อไปนี้ Finish[i] == false และ Requesti ≤ Work
3. ถ้าไม่เป็นตามนี้ไปข้อ 4 Work = Work + Allocation; Finish[i] = true ; กลับไปยังข้อ 2
4. ถ้า Finish[i] == false สำหรับบางค่าของ i ที่ 0 ≤ i ≤ n แล้วระบบจะอยู่ในสภาวะการติดตาย ยิ่งกว่านั้นถ้า Finish[i] == false แล้วโพรเซส i จะถูกติดตาย

อัลกอริทึมนี้ใช้เวลาในการทำงาน order = m x n2

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

ภาพที่ 5.11 ข้อมูลตัวอย่างที่ระบบทำงาน

ในระบบดังภาพที่ 5.11 ไม่เกิดการติดตาย เมื่อเราใช้อัลกอริทึมการตรวจจับสามารถลำดับออกเป็น <P0, P2, P3, P1, P4> ที่ทำให้ค่า i เป็นจริงทุกค่า สมมติว่าถ้าให้โพรเซส 2 มีการขอใช้ทรัพยากรเพิ่มอีก 1 ในทรัพยากร C ดังนั้นค่า Request จะเป็นไปตามภาพที่ 5.12 ระบบสามารถเกิดการติดตายแน่นอนเนื่องจากโพรเซส 0 ก็ไม่สามารถปล่อยทรัพยากร C ให้โพรเซส 2 ใช้เนื่องจากโพรเซส 0 ไม่ได้มีการใช้งานทรัพยากร C เลย

	Request
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

ภาพที่ 5.12 ค่าของ Request หลังจากที่โพรเซส C มีการขอใช้ทรัพยากร

การใช้งานอัลกอริทึมตรวจจับ จะมีการใช้งานอัลกอริทึมก็ต่อเมื่อ 1) การติดตายเกิดขึ้นบ่อยอย่างไร 2) จะมีโพรเซสจำนวนเท่าไหร่ที่ถูกผลกระทบเมื่อเกิดการติดตาย ถ้ามีการใช้อัลกอริทึมตรวจจับในลักษณะวนลูป อาจพบหลายลูปในทรัพยากรกราฟ ดังนั้นจะไม่สามารถบอกได้ว่าโพรเซสใดที่ทำให้เกิดการติดตาย

5.9 การกู้คืนจากการติดตาย

ระบบที่เกิดการติดตายจะไม่สามารถทำงานใด ๆ ต่อไปได้ จำเป็นต้องมีวิธีการกำจัดการติดตายที่เกิดขึ้น เพื่อกู้ระบบกลับคืนสู่สภาพที่ใกล้เคียงกับสภาพก่อนที่จะเกิดการติดตาย เพื่อให้ระบบสามารถทำงานที่ยังค้างอยู่ต่อไปได้ วิธีการที่ใช้กำจัดการติดตายสามารถทำได้ 2 วิธี คือ

1. การยกเลิกโพรเซส (Process termination)
2. การแย่งชิงทรัพยากรจากโพรเซส (Resource preemption)

5.9.1 การยกเลิกโพรเซส (Process Termination)

การกำจัดการติดตายด้วยการยกเลิกโพรเซส สามารถทำได้ 2 วิธี ซึ่งแต่ละวิธีนั้นระบบจะเรียกทรัพยากร (ที่ได้จัดสรรไปแล้ว) คืนจากโพรเซส จากนั้นจึงยกเลิกโพรเซส

1. การยกเลิกทุกโพรเซสที่เกิดการติดตาย วิธีนี้เป็นการกำจัดการติดตายด้วยการยกเลิก โพรเซสที่เกิดการติดตายทั้งหมด แต่มีข้อเสียอาจมีบางโพรเซสที่ทำงานมาเป็นเวลานานแล้ว และงานใกล้จะเสร็จสิ้นแล้ว แต่โพรเซสต้องถูกยกเลิกไป ทำให้โพรเซสต้องเสียเวลาและเสียค่าใช้จ่าย เนื่องจากโพรเซสที่ถูกยกเลิกไปนั้นต้องเริ่มต้นทำงานใหม่

2. การยกเลิกทีละหนึ่งโพรเซส วิธีนี้เป็นการยกเลิกโพรเซสทีละหนึ่งโพรเซส จนกว่าการติดตายจะถูกกำจัดออกไป โดยที่หลังจากแต่ละโพรเซสถูกยกเลิกหรือกำจัดไปนั้น ระบบต้องเรียกอัลกอริทึมตรวจสอบการติดตายเพื่อตรวจสอบว่า ระบบยังคงมีการติดตายอยู่หรือไม่ ทำให้เปลืองค่าใช้จ่ายอื่น (Overhead) เป็นอย่างมาก

การยกเลิกโพรเซสออกจากระบบไม่ได้เป็นสิ่งที่ทำได้ง่าย ถ้าโพรเซสทำงานไปได้ระยะหนึ่ง แล้วสถานะของทรัพยากรต่าง ๆ ถูกเปลี่ยนแปลงไป เช่น โพรเซสกำลังปรับปรุงแฟ้มข้อมูลให้เป็นปัจจุบัน และถูกยกเลิก ทำให้แฟ้มข้อมูลถูกทิ้งค้างอยู่ในสถานะไม่สมบูรณ์ หรือโพรเซสกำลังพิมพ์งานและถูกยกเลิก ระบบต้องกำหนดสถานะของเครื่องพิมพ์ใหม่ ก่อนที่จะพิมพ์งานอื่นต่อไป

นอกจากนี้การยกเลิกแต่ละโพรเซสนั้น ระบบต้องสามารถตัดสินใจได้ว่า จะเลือกยกเลิกโพรเซสใดจึงจะทำให้ระบบต้องเสียค่าใช้จ่ายน้อยที่สุด และกำจัดการติดตายออกไปได้ ดังนั้นปัจจัยที่ระบบใช้ในการพิจารณาเลือกว่าระบบจะยกเลิกโพรเซสใด โดยสามารถใช้หลักเกณฑ์ดังนี้

1. เลือกโพรเซสที่ได้ใช้เวลาของตัวประมวลผลไปแล้วน้อยที่สุด การที่ระบบเลือกยกเลิกโพรเซสลักษณะนี้ เนื่องจากจะเกิดความเสียหายน้อยกว่าการเลือกยกเลิกโพรเซสที่ถูกประมวลผลไปเป็นระยะเวลานาน และงานของโพรเซสนั้นใกล้เสร็จสมบูรณ์แล้ว
2. เลือกโพรเซสที่ได้ให้ผลลัพธ์ หรือเอาต์พุตออกมาแล้วน้อยที่สุด

3. เลือกโพรเซสที่ได้ครอบครองทรัพยากรไปแล้วน้อยที่สุด เนื่องจากโพรเซสนั้นมีโอกาสทำงานเสร็จสมบูรณ์น้อยกว่าโพรเซสที่ได้ครอบครองทรัพยากรเป็นจำนวนมาก และยังคงเหลือทรัพยากรที่ต้องการอีกเป็นจำนวนน้อยกว่า

4. เลือกโพรเซสที่มีลำดับความสำคัญ หรือความสำคัญน้อยที่สุด จะถูกยกเลิกก่อนโพรเซสที่มีลำดับความสำคัญมากกว่า

5. เลือกโพรเซสที่ยังต้องการเวลาในการทำงานมากที่สุด

5.9.2 การแย่งชิงทรัพยากรจากโพรเซส (Resource preemptive)

วิธีนี้ระบบพยายามกำจัดการติดตายด้วยการแย่งชิงทรัพยากรจากโพรเซสใดโพรเซสหนึ่ง เพื่อนำทรัพยากรนั้นไปมอบให้อีกโพรเซสหนึ่ง ทำเช่นนี้เรื่อยไปจนกว่าการติดตายจะถูกกำจัดออกไปได้ โดยที่การแย่งชิงทรัพยากรนี้ระบบจำเป็นต้องพิจารณาเรื่องต่อไปนี้

1. การเลือกเหยื่อ (Selecting a victim)

ระบบจำเป็นต้องพิจารณาว่าจะแย่งชิงทรัพยากรชนิดใดจากโพรเซสใด โดยปัจจัยที่นำมาพิจารณาคือ ค่าใช้จ่าย ซึ่งเป็นค่าใช้จ่ายที่เกิดขึ้นจาก การยกเลิกโพรเซสใดโพรเซสหนึ่ง ดังนั้นถ้าระบบจำเป็นต้องพิจารณาเพื่อยกเลิกโพรเซสจำนวนหนึ่งโพรเซส ระบบอาจเลือกยกเลิกโพรเซสที่ครอบครองทรัพยากรเป็นจำนวนน้อยที่สุด และเป็นโพรเซสที่ทำงานไปแล้วเป็นเวลาไม่นานนัก

2. การถอยกลับ (Rollback)

ถ้าระบบแย่งชิงทรัพยากรจากโพรเซสใดก็ตาม จะทำให้โพรเซสที่ถูกแย่งชิงทรัพยากรไปนั้นไม่สามารถประมวลผลต่อไปได้ ระบบจำเป็นต้องให้โพรเซสถอยหลังกลับไปอยู่ในสถานะปลอดภัย และเริ่มต้นทำงานจากสถานะนั้น วิธีที่ง่ายที่สุดในการพิจารณาให้โพรเซสถอยกลับไปอยู่ในสถานะที่ปลอดภัย คือ ให้โพรเซสถอยหลังกลับไปยังสถานะที่อยู่ห่างจากสถานะที่ทำให้เกิดการติดตายมากที่สุด การย้อนกลับต้องพิจารณาด้วยว่าการกลับเริ่มทำงานใหม่ของโพรเซส ดังนั้นต้องมีการเก็บค่าสถานะของ โพรเซสที่กำลังทำงานไว้ก่อนเริ่มงานใหม่

3. การอดตาย (Starvation)

ในระบบที่ใช้วิธีการเลือกเหยื่อโดยพิจารณาจากค่าใช้จ่ายเป็นสำคัญ ในการแย่งชิงทรัพยากรของโพรเซสนั้น เป็นไปได้ว่าอาจเกิดการแย่งชิงทรัพยากรจากโพรเซสเดิมอยู่ตลอดเวลา ทำให้ โพรเซสที่ถูกแย่งชิงทรัพยากรนั้นไม่สามารถทำงานได้เสร็จสิ้น คือ โพรเซสอยู่ในสถานะอดตาย การรอดตายต้องให้มั่นใจว่าเมื่อยึดไปแล้วจะไม่เกิดการอดตาย เนื่องจากโพรเซสไม่มีโอกาสได้ใช้ทรัพยากรอีกเลย

5.10 สรุป

วงจรอับหรือเรียกกันว่า การติดตาย เป็นปัญหาที่สำคัญสำหรับทุก ๆ ระบบปฏิบัติการ ปัญหานี้จะเกิดขึ้นเมื่อกลุ่มของโพรเซสหลาย ๆ โพรเซสต่างได้รับและกำลังถือครองทรัพยากร และโพรเซสแต่ละ โพรเซสเหล่านี้ต่างต้องการใช้ทรัพยากรที่โพรเซสอื่นในกลุ่มนั้นครองอยู่ ดังนั้นโพรเซสภายในกลุ่มนั้นจะถูกปฏิเสธให้ทำงาน (Blocked) และไม่มีโพรเซสใดทำงานได้อีกต่อไป ระบบปฏิบัติการสามารถหลีกเลี่ยง

วงจรอับได้โดยตรวจสอบว่าระบบอยู่ในสถานะที่ปลอดภัยหรือไม่อย่างสม่ำเสมอ สถานะที่ปลอดภัยหรือ Safe state คือสถานะที่ระบบสามารถทำการยืนยันได้ว่าโพรเซสต่าง ๆ ในระบบสามารถทำงานจนเสร็จสมบูรณ์ได้ แต่จะไม่สามารถทำการยืนยันได้ในสถานะที่ไม่ปลอดภัย และวิธีการคิดแบบนายธนาคาร (Banker's Algorithm) สามารถนำมาใช้ให้ระบบหลีกเลี่ยงการเกิดวงจรอับได้ โดยไม่อนุญาตให้ทรัพยากรแก่โพรเซสที่ร้องขอ

ถ้าการให้นั้นจะทำให้ระบบเข้าไปอยู่ในสถานะที่ไม่ปลอดภัย เราสามารถออกแบบระบบที่มีการป้องกันการเกิดวงจรอับตั้งแต่เริ่มต้นได้ ตัวอย่างเช่น การที่เราจะอนุญาตให้โพรเซสสามารถครอบครองทรัพยากรได้เพียง 1 ตัวในขณะหนึ่ง เพื่อเป็นการป้องกันการเกิดวงจรรอคอย (Circular wait) ซึ่งทำให้เกิดปัญหาวงจรอับอย่างแน่นอน หรือ วงจรอับสามารถป้องกันได้โดยใส่ลำดับเลขให้แก่ทรัพยากรทั้งหมด และให้โพรเซสร้องขอได้เฉพาะทรัพยากรที่มีลำดับของเลขสูงกว่าตัวเองถือครองอยู่

แบบฝึกหัดท้ายบทที่ 5

1. จงอธิบายลักษณะของการเกิด Deadlock มีอะไรบ้าง มาให้เข้าใจ

2. จงอธิบายการป้องกันการเกิด Deadlock พร้อมยกตัวอย่างการป้องกันปัญหานี้มา 1 ตัวอย่าง

3. พิจารณาระบบจำลองที่สร้างเพื่อตรวจสอบภาวะของระบบต่อไปนี้

	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0
P1	1 0 0 0	1 7 5 0	
P2	1 3 5 4	2 3 5 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

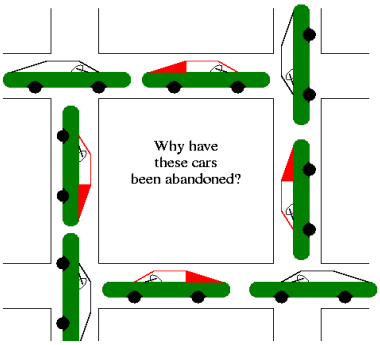
จงตอบคำถามต่อไปนี้โดยใช้ Banker's Algorithm

3.1 อะไรคือประเด็นที่ต้องการของเมตริกซ์นี้

3.2 ระบบอยู่ในสภาวะที่ไม่เกิด Deadlock ใช่หรือไม่

3.3 ถ้าการร้องขอจาก P1 มาถึง (0,4,2,0) การร้องของ P1 จะสามารถให้สิทธิได้ทันทีหรือไม่ เพราะเหตุใด

4. พิจารณาจากรูปด้านล่างการจราจร เข้าข่ายลักษณะของเงื่อนไขการเกิด Deadlock อะไรบ้างจงอธิบายมาให้เข้าใจ



5. จงอธิบายวิธีการป้องกัน Deadlock ว่าทำอย่างไร

6. ในกรณีที่เราใช้โปรแกรมที่มีโปรแกรมย่อย ๆ หลายโปรแกรม หรือโปรแกรมที่มีโครงสร้างที่สามารถทำงานหลาย ๆ อย่างพร้อมกันได้ (Multithreaded structure) จงอธิบายถึงสาเหตุที่โปรแกรมเหล่านี้มีโอกาสเกิด Deadlock ได้จากสาเหตุใดบ้าง

7. จงอธิบายหลักเกณฑ์ใดบ้างที่ถูกเลือกให้ยกเลิกการทำงานของแต่ละโพรเซสที่เกิดวงจรอับทีละตัวจนกว่าไม่มีวงจรอับเกิดขึ้นในระบบ

8. ในการแก้ปัญหาการติดตายด้วยการแย่งชิงทรัพยากรจากโพรเซสอื่น เพื่อให้อีกโพรเซสทำงานได้แล้วส่งผลให้ระบบไม่เกิดการติดตาย จงอธิบายถึงสาเหตุจำเป็นที่ต้องพิจารณาในด้านใดบ้าง

9. สมมุติเหตุการณ์มีรถยนต์ 4 คันแล่นมาในทิศทาง 4 ทิศทางต่างกัน เข้าสู่แยกพร้อมกัน ณ เวลาเดียวกัน นักศึกษาสามารถเปรียบเส้นทางทั้งสี่เส้นทางเป็นทรัพยากรระบบที่ต้องควบคุม และทั้งรถยนต์ทั้ง 4 คัน ต้องแล่นไปยังเส้นทางที่ตรงข้าม ระบบจะต้องทำหน้าที่จัดสรรทรัพยากรอย่างไร

บรรณานุกรมประจำบทที่ 5

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

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

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

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

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

บทที่ 6
กำหนดการใช้ซีพียู

ในการทำงานของโพรเซส จำเป็นต้องมีการเรียกใช้ทรัพยากรของระบบซึ่งมีอยู่อย่างจำกัด ระบบจำเป็นต้องมีการแบ่งสรรและจัดลำดับการเข้าใช้งานของทรัพยากร ซีพียูเป็นทรัพยากรที่จำเป็นต้องมีการจัดลำดับให้หลายโพรเซสเข้าไปใช้งาน ดังนั้นเมื่อซีพียูว่างและมีโพรเซสเข้าคิวรออยู่ที่คิวพร้อม ต้องการเข้าไปใช้งานที่ซีพียู ในการกำหนดการใช้ซีพียู (CPU scheduler) จะทำหน้าที่เลือกโพรเซสที่อยู่ในคิวพร้อมออกมาทีละโพรเซส และมอบซีพียูให้โพรเซสที่ได้รับเลือก ทำให้โพรเซสนั้นเข้าไปทำงานที่ซีพียูได้ และเปลี่ยนสถานะจากสถานะพร้อมเป็นสถานะทำงาน

การจัดเวลาซีพียู (CPU scheduling) เป็นหลักการทำงานหนึ่งของระบบปฏิบัติการ ที่ทำให้คอมพิวเตอร์มีความสามารถในการรันโปรแกรมหลาย ๆ โปรแกรมในเวลาเดียวกัน ซึ่งการแบ่งเวลาการเข้าใช้ซีพียูให้กับโพรเซสที่อาจถูกส่งมาหลาย ๆ โพรเซสพร้อม ๆ กัน ในขณะที่ซีพียูอาจมีจำนวนน้อยกว่าโพรเซส หรืออาจมีซีพียูเพียงตัวเดียว จะทำให้คอมพิวเตอร์สามารถทำงานได้ปริมาณงานที่มากขึ้นกว่าการให้ซีพียูทำงานให้เสร็จทีละโพรเซส ในบทนี้เราจะมาพูดถึงอัลกอริทึมพื้นฐานของการจัดเวลาซีพียูนี้ โดยจะพูดถึงวิธีการหลัก ๆ ที่แต่ละอัลกอริทึมมีแตกต่างกัน ข้อดีข้อเสียและความเหมาะสมต่อระบบงานแบบต่าง ๆ เพื่อการเลือกใช้อย่างถูกต้อง

6.1 หลักความต้องการพื้นฐาน

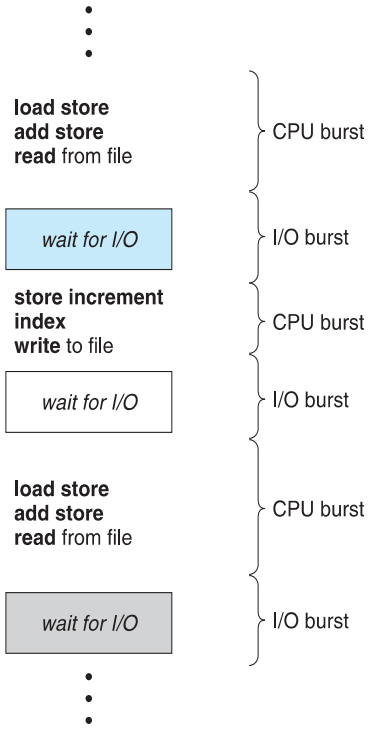
จุดประสงค์ของการรันโปรแกรมหลายโปรแกรมคือ ความต้องการที่จะให้ซีพียูมีการทำงานตลอดเวลา เพื่อให้มีการใช้ซีพียูอย่างเต็มที่และเต็มประสิทธิภาพ ซึ่งระบบคอมพิวเตอร์มีซีพียูตัวเดียว ในเวลาใดเวลาหนึ่งซีพียูจะทำงานได้เพียงงานเดียวเท่านั้น ถ้ามีหลายโปรแกรมหรือหลายงาน งานที่เหลือก็ต้องคอยจนกว่าจะมีการจัดการให้เข้าไปใช้ซีพียู ความคิดและหลักการของการทำงานกับหลายโปรแกรมในขั้นพื้นฐานนั้นค่อนข้างที่จะไม่ซับซ้อน แต่ละโปรแกรมจะถูกรันจนกระทั่งถึงจุดที่มันจะต้องคอยอะไรซักอย่างเพื่อใช้สำหรับการทำงานช่วงต่อไป ส่วนมากการคอยเหล่านี้ก็คือการทำงานที่เกี่ยวข้องกับอินพุต/เอาต์พุตนั่นเอง ในระบบคอมพิวเตอร์ที่มีความสามารถรันโปรแกรมได้ทีละโปรแกรม การทำงานของระบบก็จะไม่ซับซ้อน ซีพียูจะหยุดการทำงานในระหว่างที่คอยอินพุต/เอาต์พุตนี้ ซึ่งการคอยเหล่านี้เป็นการเสียเวลาโดยเปล่าประโยชน์ เพราะซีพียูไม่ได้ทำงานเลย ส่วนหลักในการรันหลายโปรแกรม เราพยายามใช้เวลาที่ซีพียูต้องคอยนี้ทำงานอย่างอื่นต่อไป

ดังนั้นเมื่อใดก็ตามที่ซีพียูต้องคอย และยังมีโปรแกรมในหน่วยความจำหลายโปรแกรมที่คอยการใช้ซีพียูอยู่ เราจะจัดให้ซีพียูทำงานในโปรแกรมเหล่านั้นทันที ซึ่งระบบจะจัดการนำเอาโปรแกรมที่คอยอินพุต/เอาต์พุตออกไปก่อน เพื่อที่จะให้โปรแกรมอื่นที่คอยใช้ซีพียูนี้สามารถเข้ามาได้ ถ้าทำงานในแบบนี้ไปเรื่อย ๆ ซีพียูก็จะได้มีงานเกือบตลอดเวลากับโปรแกรมหลาย ๆ โปรแกรมที่อยู่ในระบบ การจัดเวลาให้กับซีพียูแบบนี้ เป็นหลักความต้องการพื้นฐานของระบบปฏิบัติการในคอมพิวเตอร์ ทรัพยากรที่คอมพิวเตอร์มีอยู่ในเครื่อง ๆ หนึ่ง จะถูกจัดสรรการใช้ก่อนการนำไปให้โปรแกรมใด ๆ ซีพียูเองก็ถือได้ว่า

เป็นทรัพยากรของระบบคอมพิวเตอร์ชนิดหนึ่งที่มีความสำคัญมากที่สุด โดยซีพียูนี่เองที่จะเป็นศูนย์กลางของการสร้างระบบปฏิบัติการที่มีความสามารถในการรันหลายโปรแกรม

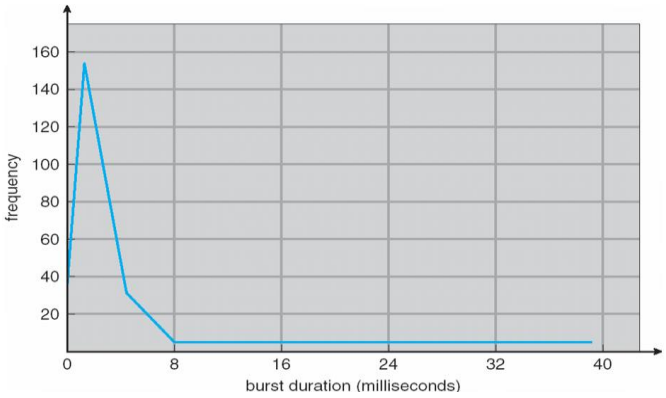
6.1.1 ช่วงเวลาอินพุต/เอาต์พุต และช่วงเวลาใช้ซีพียู (I/O and CPU Burst Cycle)

ความสำคัญของการจัดเวลาของซีพียูนั้น ขึ้นอยู่กับคุณลักษณะการทำงานของโพรเซส โดยทั่ว ไป การทำงานของโพรเซสจะประกอบด้วยเวลาที่ใช้ซีพียู (CPU burst cycle) และเวลาที่คอยอุปกรณ์อินพุต/เอาต์พุต (Input/Output and CPU burst cycle) ในขณะที่มีการทำงานโพรเซส จะมี การสลับการทำงานระหว่าง 2 ช่วงเวลานี้เท่านั้น และจะเกิดไม่พร้อมกัน และการทำงานมักจะเริ่มจาก การใช้ซีพียู แล้วก็ตามด้วยรออินพุต/เอาต์พุต เมื่อจบการรอคอยก็จะตามมาด้วยเวลาของซีพียู สลับกันไปเรื่อย ๆ จนกว่าจะจบการทำงาน ซึ่งการทำงานนี้มักเป็นการใช้เวลาซีพียูเพื่อทำการจบหรือสิ้นสุดโพรเซสมากกว่าการรอคอยอินพุต/เอาต์พุต (ดูภาพที่ 6.1)



ภาพที่ 6.1 การทำงานที่หลากหลายที่ใช้เวลาซีพียู และเวลาในการคอยอินพุต/เอาต์พุต
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.262)

จากการค้นคว้าที่ผ่านมาได้มีการศึกษาถึงลักษณะช่วงเวลาของการใช้ซีพียูไว้ สามารถมองเห็นคร่าว ๆ ว่า ลักษณะของช่วงเวลาที่ใช้ซีพียูในแต่ละโพรเซสจะมีรูปแบบอย่างไร ถึงแม้ว่ารูปแบบของเวลาอาจจะมีความแตกต่างกันอยู่บ้าง แต่ก็มีแนวโน้มที่แต่ละโพรเซสมักจะมีคาบเวลาการเข้าใช้ซีพียูคล้าย ๆ กันในภาพที่ 6.2



ภาพที่ 6.2 ฮีสโตแกรมของเวลาซีพียู
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.263)

จากกราฟอธิบายได้ว่า การใช้ซีพียูของโพรเซสใด ๆ นั้น มักจะมีความถี่มากสำหรับเวลาซีพียูช่วงเวลาสั้น ๆ ส่วนเวลาซีพียูระยะเวลานาน ๆ นั้นจะมีความถี่น้อยกว่า นอกจากนี้ยังมีแนวโน้มว่า โพรเซสที่มีการติดต่อแลกเปลี่ยนข้อมูลกับอุปกรณ์ภายนอกมาก ๆ ก็มักจะมีช่วงเวลาของการใช้ซีพียูค่อนข้างสั้นแต่บ่อยครั้ง และมีช่วงเวลาการใช้ซีพียูนาน ๆ เพียงเล็กน้อย และการศึกษาคุณสมบัติของโพรเซสต่าง ๆ เหล่านี้อย่างละเอียด ทำให้เราสามารถนำมาใช้เป็นข้อมูลในการเลือกลักษณะของอัลกอริทึมที่เหมาะสมสำหรับการจัดเวลาให้กับซีพียูให้ดีที่สุดในระบบคอมพิวเตอร์แต่ละระบบได้ต่อไป

6.2 ตัวจัดการเวลาซีพียู

เมื่อใดก็ตามที่ซีพียูว่าง ระบบปฏิบัติการจะต้องเข้ามาเลือกโพรเซสตัวใดตัวหนึ่งที่คอยอยู่ในคิวเข้ามาใช้งานซีพียู การเลือกโพรเซสเพื่อเข้ามาใช้ซีพียูนี้ จะถูกจัดการด้วยส่วนที่เรียกว่า ตัวจัดการช่วงสั้น (Short – term scheduler) หรือตัวจัดการเวลาซีพียู (CPU scheduler) ตัวจัดการเวลานี้จะเลือกโพรเซสที่อยู่ในหน่วยความจำที่พร้อมในการทำงานที่สุด เพื่อให้ครอบครองเวลาซีพียูและทรัพยากรที่เกี่ยวข้องกับโพรเซสนั้น ตัวของโพรเซสในหน่วยความจำนั้นไม่จำเป็นต้องเป็นแบบมาก่อนได้ก่อน (FIFO : First in First out) เสมอไป อาจจะเป็นไปตามลำดับความสำคัญ โครงร่างต้นไม้ (Tree) หรืออาจจะเป็น ลิ้งลิสต์ก็ได้ อย่างไรก็ตามโพรเซสทุกโพรเซสที่พร้อมใช้ซีพียู จะต้องมีโอกาสได้เข้าครอบครองเวลาซีพียูไม่เวลาใดก็เวลาหนึ่ง ซึ่งการเข้าและออกจากการครอบครองเวลาซีพียูแต่ละครั้ง จำเป็นต้องมีการเก็บข้อมูลไว้เสมอว่า เข้ามาแล้วได้ทำอะไรไปบ้าง ช่วงต่อไปจะทำอะไร เป็นต้น โดยใช้พื้นที่หน่วยความจำส่วนหนึ่งเก็บข้อมูลของแต่ละโพรเซสหลังเสร็จสิ้นการใช้ซีพียู พื้นที่หน่วยความจำที่มีชื่อว่าบล็อกควบคุมโพรเซส (Process Control Block : PCB)

6.2.1 การให้สิทธิการจัดเวลา (Preemptive Scheduling)

การตัดสินใจของซีพียูในการเลือกให้โพรเซสใด ๆ ทำงาน ขึ้นอยู่กับสถานการณ์ดังนี้

1. เมื่อมีการเปลี่ยนสถานะของโพรเซสจากสถานะรัน ไปเป็นสถานะคอย เช่น ในสภาวะที่คอยอินพุต/เอาต์พุต หรือการคอยให้โพรเซสลูกเสร็จสิ้นไปก่อน เป็นต้น

2. เมื่อมีการเปลี่ยนสถานะของโพรเซสจากรัน เป็นสถานะพร้อม เช่น เมื่อมีอินเทอร์รัพเกิดขึ้น

3. เมื่อมีการเปลี่ยนสถานะของโพรเซสจากสถานะคอย เป็นสถานะพร้อม เช่น เมื่อ อินพุต/เอาต์พุต เสร็จสิ้นไปแล้ว เป็นต้น

4. เมื่อโพรเซสเสร็จสิ้นหรือสิ้นสุดการดำเนินการ

ทั้ง 4 สถานการณ์ดังกล่าวข้างต้น ในสถานการณ์ที่ 1 และ 4 นั้นเป็นสถานการณ์ที่จะต้องมีการตัดสินใจทำอะไรอย่างใดอย่างหนึ่ง โดยไม่สามารถหลีกเลี่ยงได้ เช่น ต้องไปเลือกโพรเซสใหม่เข้ามาทำงานต่อไป เนื่องจากโพรเซสเดิมไม่ใช้ซีพียูอีกแล้ว แต่สำหรับสถานการณ์ที่ 2 และ 3 นั้น การตัดสินใจต้องอยู่บนพื้นฐานหรือกฎเกณฑ์ของแต่ละอัลกอริทึมที่ใช้ ซึ่งอาจทำให้มีการทำงานที่แตกต่างกันไป การตัดสินใจที่เกิดขึ้นเนื่องจากสถานการณ์ 1 และ 4 การจัดเวลาซีพียูจะเป็นแบบไม่ให้สิทธิ์ก่อน (Non preemptive) นอกนั้นจะเรียกว่าให้สิทธิ์ก่อน (Preemptive) ภายใต้การทำงานแบบไม่ให้สิทธิ์ก่อนนี้ โพรเซสจะครอบครองเวลาซีพียูไปจนกว่าจะเสร็จ หรือเปลี่ยนสถานะตัวเองเป็นสถานะคอย ซึ่งเป็นเพียงวิธีการที่ใช้ได้เฉพาะกับระบบของฮาร์ดแวร์เฉพาะแบบเท่านั้น ถ้าระบบนี้ต้องการทำเป็นระบบที่ให้ผู้ใช้หลายคนเข้ามาใช้งานพร้อมกัน (Multi-user) การจัดเวลาแบบให้สิทธิ์ก่อนจะเหมาะสมกว่า

6.2.2 ตัวส่งต่อ (Dispatcher)

องค์ประกอบที่สำคัญอีกตัวหนึ่งที่เกี่ยวข้องกับฟังก์ชันในการจัดเวลาซีพียูก็คือ สิ่งที่เรียกว่า Dispatcher ซึ่งเป็นโมดูลที่ทำหน้าที่ควบคุมการครอบครองซีพียูของโพรเซส โมดูลนี้ประกอบด้วยฟังก์ชัน

1. การย้าย Context
2. การย้ายไป User mode
3. กระโดดไปยังตำแหน่งที่เหมาะสมของโปรแกรม เพื่อที่จะเริ่มรันโปรแกรมนั้นใหม่อีกครั้ง

ลักษณะของ Dispatcher คือการทำ Context switching ดังนั้นควรมีการทำงานที่เร็วที่สุดเท่าที่จะทำได้ เพราะว่ามันจะต้องทำงานทุกครั้งที่มีการย้ายโพรเซส ซึ่งเวลาที่ถูกใช้ไปกับการย้ายโพรเซสที่กำลังใช้ซีพียูให้ออกจากซีพียู และนำโพรเซสอื่นเข้าใช้ซีพียูแทนเช่นนี้เรียกว่า Dispatch latency

6.3 เกณฑ์การวิเคราะห์ประสิทธิภาพ

อัลกอริทึมในการเลือกโพรเซสเข้าไปใช้ซีพียูมีหลายอัลกอริทึม และแต่ละอัลกอริทึมของการจัดเวลาซีพียูมีคุณสมบัติแตกต่างกันไป ดังนั้นการเลือกอัลกอริทึมสำหรับใช้ในสถานการณ์ต่าง ๆ นั้น จำเป็นที่จะต้องพิจารณาถึงคุณสมบัติและเป้าหมายการทำงานเหล่านี้ให้ดีว่ามีข้อดีข้อเสียอย่างไร ซึ่งลักษณะข้อพิจารณาแต่ละชนิดนั้น สามารถสร้างความแตกต่างให้เห็นได้อย่างชัดเจนในการตัดสินว่าอัลกอริทึมใดดีที่สุด ข้อพิจารณาดังกล่าวมีดังนี้

1. มีการใช้งานหน่วยประมวลผลกลาง (CPU utilization) โดยจำเป็นให้หน่วยประมวลผลกลางถูกใช้งานให้มากที่สุด โดยปกติควรจะมีการใช้งานร้อยละ 40 ถึงร้อยละ 90 ทั้งนี้ขึ้นอยู่กับปริมาณงานในระบบ

2. มีปริมาณงานมากที่สุด (Throughput) คือปริมาณงานต่อหน่วยเวลา เกณฑ์นี้เกี่ยวพันกับการใช้งานหน่วยประมวลผลกลาง ทั้งนี้ปริมาณงานที่ผ่านเข้ามาในระบบมาก หน่วยประมวลผลกลางจะถูกใช้งานมากตามปริมาณงาน

3. มีเวลาครบวงงานน้อยที่สุด (Turnaround time) เป็นเวลาต่อรอบงาน คือเวลาที่ใช้ในระบบทั้งหมดในการทำงานของโพรเซสใด ๆ หรือกล่าวได้ว่าเป็นเวลาที่ผู้ใช้ต้องรอ โดยเริ่มตั้งแต่นำโพรเซสเข้าสู่ระบบจนได้รับผลลัพธ์หรือเอาท์พุตที่ต้องการกลับมา สิ่งที่ควรให้ความสำคัญคือแม้ว่าการใช้งานหน่วยประมวลผลกลางจะสูง แต่ถ้าผู้ใช้ระบบต้องรอนาน อาจทำให้เกิดความล่าช้าต่อความต้องการของผู้ใช้งาน

4. มีเวลารอน้อยที่สุด (Waiting time) คือเวลาของโพรเซสที่ถูกรอใน Ready queue

5. มีเวลาตอบสนองน้อยที่สุด (Response time) หมายถึงเวลาที่มีการร้องขอข้อมูลหรือการส่งงานเข้าไปในระบบและได้การตอบสนองกลับมาในครั้งแรก (ไม่ใช้ผลลัพธ์) เช่น ระบบที่มี การโต้ตอบข้อมูล (interactive system) ซึ่งต้องมีการตอบข้อมูลกลับมาหลังป้อนคำสั่งเข้าไป

6.4 อัลกอริทึมของการจัดเวลา

อัลกอริทึมสำหรับการจัดการเวลาในโพรเซส นั้นมีความสำคัญอยู่ที่การตัดสินว่าจะให้โพรเซสใดครอบครองเวลาของซีพียูก่อน ซึ่งต่อไปนี้จะมาศึกษากันว่ามีวิธีการใดที่ใช้ในการตัดสินใจคัดเลือกโพรเซส

6.4.1 First-Come, First-Served (FCFS) Scheduling

เป็นอัลกอริทึมที่ง่ายที่สุด โดยจะกำหนดให้โพรเซสที่ร้องขอซีพียูก่อน เป็นโพรเซสที่ได้รับซีพียูก่อน เมื่อมีโพรเซสที่อยู่ในสถานะพร้อมที่จะทำงาน โพรเซสนั้นจะถูกนำเข้าไปต่อท้ายคิวพร้อม เมื่อซีพียูว่าง ระบบปฏิบัติการจะเรียกกำหนดการซีพียู เพื่อให้พิจารณามอบซีพียูให้แก่โพรเซสที่อยู่ต้นคิวของคิวพร้อม สูตรในการคำนวณหาเวลาครบวงงาน สามารถคำนวณได้ดังนี้

$T = \sum_{i=1}^{n} Ti \times 1/n$ เมื่อ $Ti = Fi - Ai$

Ti หมายถึง เวลาครบวงงานของแต่ละโพรเซส (Turnaround time)
Fi หมายถึง เวลาที่แต่ละโพรเซสทำงานเสร็จสิ้น (Finish time)
Ai หมายถึง เวลาที่แต่ละโพรเซสเข้ามาในระบบ (Arrival time)
n หมายถึง จำนวนของโพรเซสที่เข้ามาในระบบ
T หมายถึง เวลาครบวงงานเฉลี่ย (Average Turnaround time)

ตัวอย่างที่ 1 ระบบคอมพิวเตอร์มี 3 โพรเซสที่ต้องการเข้าใช้งานซีพียู คือ P1, P2 และ P3 เมื่อ

1. โพรเซส P1 เข้าระบบเมื่อเวลา 8.00 และต้องการใช้ซีพียู 2 หน่วยเวลา
2. โพรเซส P2 เข้าระบบเมื่อเวลา 8.10 และต้องการใช้ซีพียู 1 หน่วยเวลา
3. โพรเซส P3 เข้าระบบเมื่อเวลา 8.25 และต้องการใช้ซีพียู 0.25 หน่วยเวลา

ให้แสดงวิธีทำเพื่อหาเวลาครบวงงานเฉลี่ย กรณีใช้อัลกอริทึมจัดลำดับใช้ซีพียูแบบมาก่อนบริการก่อน

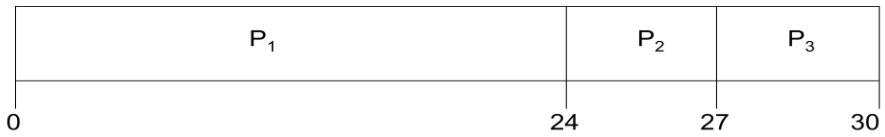
Process	Arrival Time	Run Time	Start Time	Finish Time	Turnaround Time
P1	8.00	2.00	8.00	10.00	2.00
P2	8.10	1.00	10.00	11.00	2.90
P3	8.25	0.25	11.00	11.25	3.00
				รวม	7.90
					เวลาครบวงงานเฉลี่ย = 7.90/3 = 2.63 หน่วยเวลา

จากการทำงานด้วยอัลกอริทึมนี้ สามารถคำนวณค่าเฉลี่ยของเวลาครบวงงานได้ เท่ากับ 2.63 หน่วยเวลา

ตัวอย่างที่ 2 ให้พิจารณาระบบที่ประกอบไปด้วย 3 โพรเซสที่ถูกรับเข้ามาในระบบ เรียงตามลำดับ คือ P1, P2 และ P3 โดยที่แต่ละโพรเซสต้องการใช้ซีพียูเป็นเวลาตามที่กำหนด ให้หาค่าเฉลี่ยของเวลารอ เมื่อกำหนดให้ใช้อัลกอริทึมแบบมาก่อนบริการก่อน

Process	Burst Time
P_1	24
P_2	3
P_3	3

การรอของแต่ละโพรเซส สามารถแสดงได้ด้วย Gantt Chart ดังนี้:



- P1 เข้ามาในระบบและได้รับการจัดสรรให้ใช้ซีพียูทันที จึงไม่ต้องเสียเวลาในการรอซีพียู ดังนั้นเวลาในการรอซีพียู = 0 หน่วยเวลา
- P2 ต้องรอจนกว่า P1 ทำงานเสร็จเรียบร้อยและคืนซีพียูให้กับระบบ ดังนั้นเวลาในการรอซีพียู = 24 หน่วยเวลา
- P3 ต้องรอจนกว่า P2 ทำงานเสร็จเรียบร้อยและคืนซีพียูให้กับระบบ ดังนั้นเวลาในการรอซีพียู = 27 หน่วยเวลา

ดังนั้นค่าเฉลี่ยของเวลาที่ใช้ในการรอซีพียู คือ (0+24+27)/3 = 17 หน่วยเวลา การทำงานของอัลกอริทึมนี้ดูเหมือนเป็นการยุติธรรมที่ให้สิทธิการเข้าใช้ซีพียูแก่โพรเซสที่เข้ามาอยู่ในคิวพร้อมก่อน แต่ใน

กรณีที่ในคิวพร้อมของระบบมีทั้งโพรเซสที่เน้นซีพียู และโพรเซสที่เน้น I/O จะพบว่า โพรเซสที่เน้น I/O จะต้องเสียเวลารอนานมาก เพื่อเข้าใช้งานซีพียูในระยะเวลาที่ไม่นานมากนัก
ซึ่งจะทำให้เกิดปัญหา Convoy effect คือ เหตุการณ์ที่โพรเซสขนาดเล็กในระบบ จะต้องเสียเวลารอโพรเซสขนาดใหญ่ที่ครอบครองซีพียูเป็นเวลานาน การทำงานของอัลกอริทึมนี้ เป็นการทำงานที่ไม่สามารถขัดจังหวะ หรือแทรกกลางได้ (Non-preemptive process) ซึ่งจะไม่เหมาะกับระบบที่ต้องมีการแบ่งส่วนการทำงานให้งานแต่ละงานได้ใช้ซีพียูอย่างทั่วถึง

6.4.2 Shortest-Job-First (SJF) Scheduling

จากอัลกอริทึมมาก่อนบริการก่อนนั้น พบว่าค่าเฉลี่ยของเวลาครบวงงาน และค่าเฉลี่ยของเวลารอมีค่าสูง โดยเฉพาะกรณีที่ในคิวพร้อมมีโพรเซสที่ต้องการใช้ซีพียูเป็นเวลาที่แตกต่างกัน อัลกอริทึมของงานสั้นทำก่อน จะพยายามลดค่าเฉลี่ยของเวลาครบวงงาน และค่าเฉลี่ยของเวลารอ โดยกำหนดให้โพรเซสที่ต้องการใช้ซีพียูเป็นระยะเวลาน้อยได้เข้าใช้ซีพียูก่อนโพรเซสที่ต้องการใช้ซีพียูเป็นระยะเวลานาน

ตัวอย่างที่ 3 พิจารณาระบบที่ประกอบด้วยโพรเซส P1, P2, P3 และ P4 โดยที่ทุกโพรเซสถูกรับเข้ามาในระบบพร้อมกัน

Process	เวลาที่ต้องการใช้ซีพียู (burst time)
P1	6
P2	8
P3	7
P4	3

การรอของแต่ละโพรเซสจากอัลกอริทึม Shortest-Job-First (SJF) Scheduling สามารถแสดงได้ด้วย Gantt Chart ดังนี้

จาก Gantt Chart จะเห็นว่า

- โพรเซส P1 ต้องรอเป็นเวลา 3 หน่วยเวลา
- โพรเซส P2 ต้องรอเป็นเวลา 16 หน่วยเวลา
- โพรเซส P3 ต้องรอเป็นเวลา 9 หน่วยเวลา
- โพรเซส P4 ต้องรอเป็นเวลา 0 หน่วยเวลา
- ค่าเฉลี่ยของเวลาที่ใช้ในการรอ คือ (3+16+9+0)/4 = 7 หน่วยเวลา

ตัวอย่างที่ 4 จากโจทย์ในตัวอย่างที่ 1 ให้แสดงวิธีทำเพื่อหาเวลาครบวงงานเฉลี่ย เมื่อกำหนดให้ใช้อัลกอริทึมจัดลำดับใช้ซีพียูแบบงานสั้นทำก่อน

Process	Arrival Time	Run Time	Start Time	Finish Time	Turnaround Time
P1	8.00	2.00	8.00	10.00	2.00
P2	8.10	1.00	10.25	11.25	3.15
P3	8.25	0.25	10.00	10.25	2.00
				รวม	7.15
				เวลาครบวงงานเฉลี่ย = 7.15/3 = 2.38 หน่วยเวลา	

อัลกอริทึมนี้สามารถทำงานได้ทั้งแบบ Preemptive process และ Non-Preemptive process กรณีที่เป็นการทำงานแบบ Preemptive นั้น ขณะที่โพรเซสหนึ่งกำลังทำงาน และมีโพรเซสใหม่เข้ามาในคิวพร้อม และโพรเซสใหม่ต้องการใช้ซีพียูเป็นเวลาน้อยกว่าโพรเซสที่กำลังทำงาน โพรเซสเดิมที่กำลังทำงานจะถูกขัดจังหวะให้หยุดการทำงาน และคืนซีพียูให้แก่ระบบ เพื่อที่ระบบจะได้มอบซีพียูให้ โพรเซสใหม่ที่ต้องการใช้ซีพียูเป็นเวลาน้อยกว่าได้เข้าไปทำงานที่ซีพียูก่อน

กรณีที่เป็นการทำงานแบบ Non-Preemptive ระบบจะยังคงให้โพรเซสเดิมทำงานต่อไป จนกว่าจะเสร็จสิ้นการทำงานของโพรเซสนั้น

ตัวอย่างที่ 5 ระบบคอมพิวเตอร์มี 3 โพรเซสที่ต้องการเข้าไปใช้งานซีพียู คือ

- โพรเซส P1 เข้าระบบเมื่อเวลา 0.0 และต้องการใช้ซีพียู 8 หน่วยเวลา
- โพรเซส P2 เข้าระบบเมื่อเวลา 0.4 และต้องการใช้ซีพียู 4 หน่วยเวลา
- โพรเซส P3 เข้าระบบเมื่อเวลา 1.0 และต้องการใช้ซีพียู 1 หน่วยเวลา

ให้แสดงวิธีทำเพื่อหาเวลาครบรอบวงงานเฉลี่ย กรณีที่ใช้อัลกอริทึมจัดลำดับใช้ซีพียูแบบงานสั้นทำก่อน ในแบบ Preemptive และแบบ Non-Preemptive

1. จัดลำดับใช้ซีพียูแบบงานสั้นได้ก่อน ในแบบ Non-Preemptive

Process	Arrival Time	Run Time	Start Time	Finish Time	Turnaround Time
P1	0.0	8	0.0	8.0	8.0
P2	0.4	4	9.0	13.0	12.6
P3	1.0	1	8.0	9.0	8.0
				รวม	28.6
				เวลาครบวงงานเฉลี่ย = 28.6/3 = 9.53 หน่วยเวลา	

2. จัดลำดับใช้ซีพียูแบบงานสั้นได้ก่อน ในแบบ Preemptive

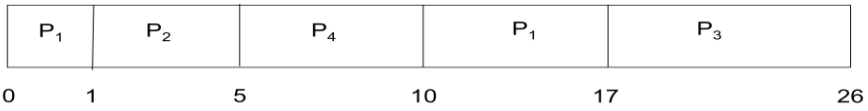
Process	Arrival Time	Run Time	Start Time	Finish Time	Left Time	Turnaround Time
P1	0.0	8	0.0	0.4	7.6	0.4
P2	0.4	4	0.4	1.0	3.4	0.6
P3	1.0	1	1.0	2.0	0	1.0
P2	1.0	3.4	2.0	5.4	0	4.4
P1	0.4	7.6	5.4	13.0	0	12.6
					รวม	19.0
เวลาครบวงงานเฉลี่ย = 19/3 = 6.33 หน่วยเวลา						

6.4.3 Shortest-remaining-time-first

เราเพิ่มแนวคิดของเวลาที่มาถึงของโพรเซสที่แตกต่างกัน และวิเคราะห์การจัดลำดับใช้ซีพียูแบบงานสั้นได้ก่อน ในแบบ Preemptive

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

การจัดลำดับใช้ซีพียูแบบงานสั้นได้ก่อน ในแบบ Preemptive สามารถแสดงได้ด้วย Gantt Chart ดังนี้:



ค่าเฉลี่ยของเวลาที่ใช้ในการรอ คือ = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

6.4.4 ลำดับความสำคัญ (Priority Scheduling)

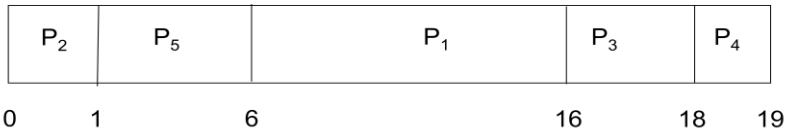
เป็นวิธีจัดลำดับการใช้ซีพียูโดยกำหนดลำดับความสำคัญให้แต่ละโพรเซส โดยระบบจะต้องกำหนดว่า ให้ตัวเลขที่มีค่าน้อยที่สุดแสดงถึงลำดับความสำคัญน้อยที่สุด ให้ตัวเลขที่มีค่ามากที่สุดแสดงถึง

ลำดับความสำคัญมากที่สุด หรือให้ตัวเลขที่มีค่าน้อยที่สุดแสดงถึงลำดับความสำคัญมากที่สุด ให้ตัวเลขที่มีค่ามากที่สุดแสดงถึงลำดับความสำคัญน้อยที่สุด

ตัวอย่างที่ 6 กำหนดให้โพรเซส P1 P2 P3 P4 P5 และ P6 มีระยะเวลาทำงานและค่าลำดับความสำคัญดังต่อไปนี้

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

สามารถแสดงได้ด้วย Gantt Chart ดังนี้:



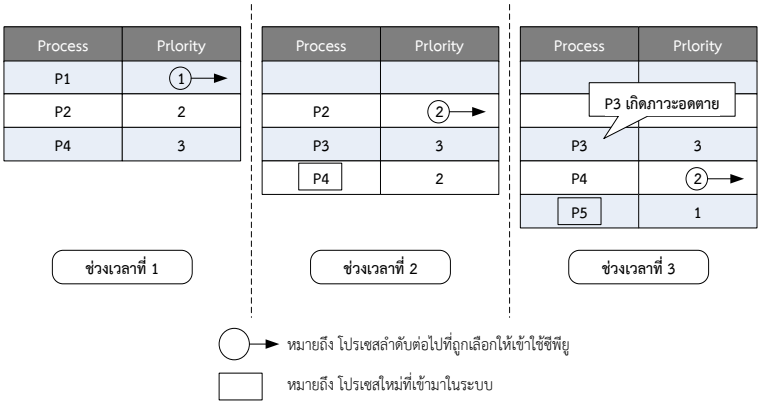
ค่าเฉลี่ยของเวลาที่ใช้ในการรอ คือ = 6+0+16+18+1/5 = 8.2 msec

การทำงานของอัลกอริทึมนี้สามารถทำงานได้ทั้งในกรณีแบบ Preemptive และแบบ Non-Preemptive โดยในกรณีที่อัลกอริทึมทำงานในแบบ Preemptive จะทำให้เกิดปัญหาสำคัญ คือ การอดตาย (Starvation) หมายความว่า โพรเซสที่มีลำดับความสำคัญต่ำกว่าถูกโพรเซสที่มีลำดับความสำคัญสูงกว่าแย่งชิงซีพียูไปใช้งาน ทำให้โพรเซสที่มีลำดับความสำคัญต่ำกว่าไม่มีโอกาสเข้าไปใช้ซีพียู วิธีการแก้ปัญหาการอดตาย สามารถทำได้โดยการทำ Aging คือ การกำหนดให้มีการเพิ่มค่าของลำดับความสำคัญของทุกโพรเซสในระบบเป็นระยะ

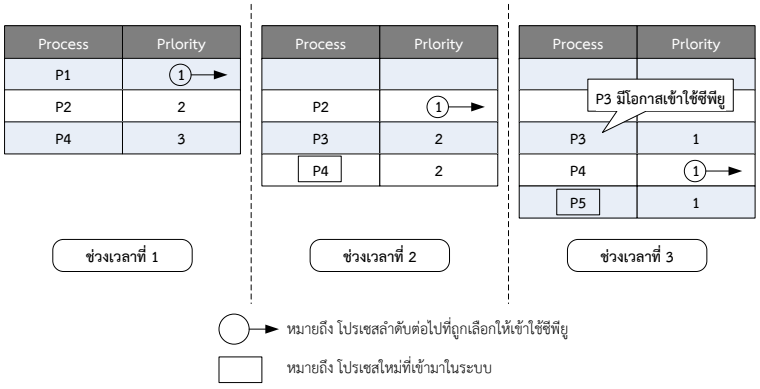
ตัวอย่างที่ 7 กรณีของการจัดลำดับใช้งานของซีพียูแบบจัดลำดับความสำคัญในแบบ Preemptive ที่มีการใช้ Aging และไม่มีการใช้ Aging

Process	Priority
P1	1
P2	2
P3	3

เมื่อกำหนดให้ ตัวเลขที่มีค่าน้อยที่สุดมีลำดับความสำคัญสูงที่สุด



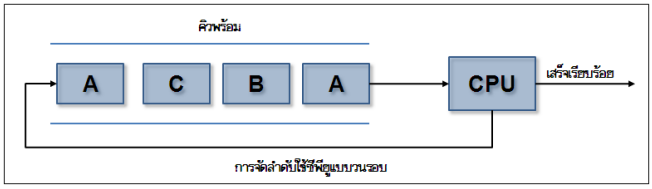
ภาพที่ 6.3 การจัดลำดับใช้งานซีพียูแบบจัดลำดับความสำคัญในแบบ Preemptive ที่ไม่มีการใช้ Aging



ภาพที่ 6.4 การจัดลำดับใช้งานซีพียูแบบจัดลำดับความสำคัญในแบบ Preemptive ที่มีการใช้ Aging

6.4.5 วิธีวนรอบ (Round-Robin Scheduling: RR)

อัลกอริทึมนี้ถูกออกแบบมาเพื่อใช้สำหรับระบบแบ่งเวลา โดยมีการทำงานเหมือนอัลกอริทึมแบบมาก่อนบริการก่อน แต่กำหนดให้โพรเซสใช้ซีพียูในเวลาที่จำกัด เรียกว่า เวลาควอนตัม (Quantum time) หรือ การแบ่งเวลา (time slice)



ภาพที่ 6.5 การแสดงการจัดลำดับซีพียูแบบวนรอบ

ในการทำงาน ตัวจัดลำดับการใช้ซีพียูจะเลือกโพรเซสจากต้นคิวพร้อมเข้าไปทำงานเป็นเวลา 1 เวลาควอนตัม ภายในระยะเวลาที่กำหนดถ้าโพรเซสสามารถทำงานเสร็จ โพรเซสจะคืนซีพียูให้ระบบ แต่

ถ้าโพรเซสไม่สามารถทำงานเสร็จภายในเวลา 1 เวลาควอนตัม โพรเซสจะถูกขัดจังหวะและถูกนำไปต่อท้ายคิวพร้อม เพื่อเปลี่ยนให้โพรเซสอื่นเข้าไปทำงานในซีพียูต่อไป

ตัวอย่างที่ 8 ระบบคอมพิวเตอร์มีโพรเซสทั้งหมด 3 โพรเซส แต่ละโพรเซสมีเวลาเข้าระบบ และเวลาที่ต้องการใช้ซีพียู ดังนี้

- โพรเซส P1 เข้าระบบเมื่อเวลา 0.0 และต้องการใช้ซีพียู 8 หน่วยเวลา
- โพรเซส P2 เข้าระบบเมื่อเวลา 0.4 และต้องการใช้ซีพียู 4 หน่วยเวลา
- โพรเซส P3 เข้าระบบเมื่อเวลา 1.0 และต้องการใช้ซีพียู 1 หน่วยเวลา

เมื่อกำหนดให้ใช้การจัดลำดับใช้งานซีพียูแบบวนรอบ ที่มีเวลาควอนตัมเท่ากับ 2.0 หน่วยเวลา ให้แสดงวิธีทำเพื่อคำนวณหาเวลาครบวงงานเฉลี่ย

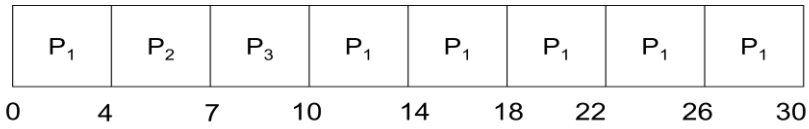
เวลา	โพรเซสที่ทำงาน
0.0-2.0	P1
2.0-4.0	P2
4.0-5.0	P3 ... งานเสร็จเรียบร้อย
5.0-7.0	P1
7.0-9.0	P2 ... งานเสร็จเรียบร้อย
9.0-11.0	P1
11.0-13.0	P1 ... งานเสร็จเรียบร้อย

Process	Arrival Time	Run Time	Start Time	Finish Time	Turnaround Time
P1	0.0	8	0.0	13.0	13.0
P2	0.4	4	2.0	9.0	8.6
P3	1.0	1	4.0	5.0	4.0
				รวม	25.6
เวลาครบวงงานเฉลี่ย = 25.6/3 = 8.53 หน่วยเวลา					

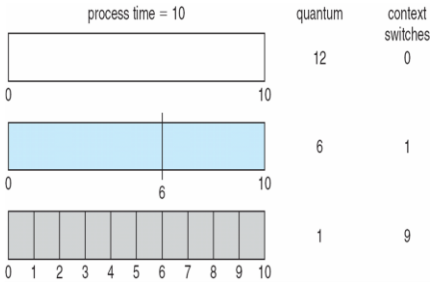
ตัวอย่างที่ 9 เมื่อกำหนดให้ใช้การจัดลำดับใช้งานซีพียูแบบวนรอบ ที่มีเวลาควอนตัมเท่ากับ 4.0 หน่วยเวลา

Process	Burst Time
P_1	24
P_2	3
P_3	3

สามารถแสดงได้ด้วย Gantt Chart ดังนี้:



เมื่อพิจารณาจะพบว่า โดยปกติแล้วเวลาครบวงงานเฉลี่ย (Turnaround time) จะมีค่าสูงกว่าอัลกอริทึมแบบ Shortest-Job-First (SJF) Scheduling แต่มีการตอบสนองที่ดีกว่าประสิทธิภาพของอัลกอริทึมแบบวนรอบขึ้นอยู่กับขนาดของเวลาควอนตัมที่กำหนด ถ้าเวลาควอนตัมมีขนาดใหญ่มาก พบว่าการทำงานของอัลกอริทึมแบบวนรอบจะเหมือนกับการทำงานของอัลกอริทึมแบบมาก่อนบริการก่อน ถ้าเวลาควอนตัมมีขนาดเล็ก เช่น 1 หน่วยเวลา จะทำให้แต่ละโพรเซสรู้สึกเหมือนว่ามีซีพียูเป็นของตัวเอง เนื่องจากโพรเซสมีโอกาสเข้าไปทำงานในซีพียูตลอดเวลา แต่สิ่งที่ตามมาคือ เวลาที่เสียไปในการสลับการทำงานจากโพรเซสหนึ่งไปยังอีกโพรเซสหนึ่ง ที่เรียกว่าการทำ Context switching ที่ต้องมีการเก็บข้อมูลต่าง ๆ ของโพรเซสที่กำลังทำงานในซีพียูและที่สลับออกไป

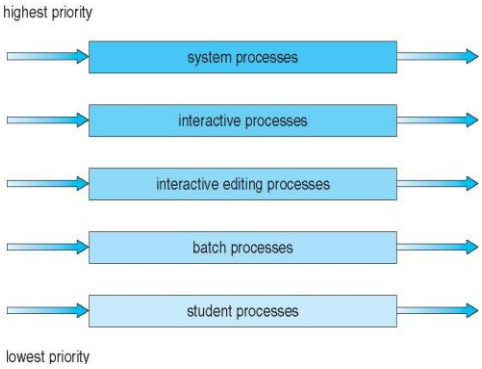


ภาพที่ 6.6 แสดงเวลาควอนตัมและเวลาในการสลับการทำงานโพรเซส
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.273)

6.5 คิวหลายระดับ

อัลกอริทึมของการจัดลำดับวิธีนี้ ถูกสร้างขึ้นจากแนวความคิดที่ว่า โพรเซสสามารถถูกแบ่งออกเป็นกลุ่มต่าง ๆ ได้หลายกลุ่ม เช่น โพรเซสของระบบ (System process) โพรเซสแบบกลุ่ม (Batch process) และโพรเซสแบบโต้ตอบ (Interactive process)

โพรเซสแต่ละกลุ่มจะมีเวลาการตอบสนอง (Response time) ที่แตกต่างกัน จึงต้องการการจัดลำดับที่แตกต่างกันด้วย เช่น โพรเซสแบบโต้ตอบต้องการได้รับการตอบสนองที่รวดเร็ว ควรได้ลำดับการทำงานก่อนโพรเซสแบบกลุ่ม ขั้นตอนวิธีในการจัดตารางการทำงานแบบแถวคอยหลายชั้นนี้ เริ่มจากการจัดแถวพร้อมของระบบออกเป็นหลาย ๆ แถวแยกจากกัน ดังแสดงในภาพที่ 6.7



ภาพที่ 6.7 แสดงการแบ่งระดับความสำคัญในการเข้าคิวหลายระดับขิงโพรเซส
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.273)

จากภาพที่ 6.7 แสดงการจัดลำดับแบบคิวหลายระดับ ที่แบ่งโพรเซสในคิวพร้อมออกเป็นคิวย่อย (Sub queue) 4 คิวย่อย ที่มีระดับความสำคัญแตกต่างกัน เมื่อมีโพรเซสใหม่เข้ามาในระบบ จะถูกนำไปเข้าคิวรอที่คิวใดคิวหนึ่ง โดยที่แต่ละคิวนี้สามารถมีการจัดลำดับที่แตกต่างกัน ในการทำงานนั้น โพรเซสที่อยู่ในคิวที่มีค่าลำดับความสำคัญสูงสุดจะถูกทำงานก่อน ส่วนโพรเซสที่อยู่ในกลุ่มที่มีค่าลำดับความสำคัญน้อยกว่าจะถูกทำงานได้ก็ต่อเมื่อ โพรเซสในคิวย่อยที่มีค่าลำดับความสำคัญสูงกว่าถูกทำงานเสร็จเรียบร้อยแล้วเท่านั้น

นอกจากนั้นก็จะต้องมีการจัดตารางการทำงานระหว่างแถวพร้อมเหล่านั้นด้วย โดยทั่วไปจะใช้การจัดตารางการทำงานแบบศักดิ์สูงได้ก่อน ชนิดที่ให้แทรกกลางได้ เช่น แถวสำหรับโพรเซสที่ทำงานแบบโต้ตอบ จะมีศักดิ์สูงกว่าแถวสำหรับโพรเซสที่ทำงานแบบกลุ่ม สมมติว่าในระบบมีแถวพร้อมอยู่ 5 แถว ดังนี้

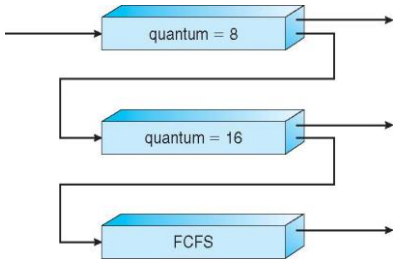
- งานของระบบ (System processes)
- งานแบบโต้ตอบ (Interactive processes)
- งานแก้ไขข้อมูล (Interactive edition processes)
- งานแบบกลุ่ม (Batch processes)
- งานของนักศึกษา (Student processes)

โดยแถวบนจะมีศักดิ์สูงกว่าแถวล่าง ดังนั้นโพรเซสที่ทำงานแบบกลุ่มจะสามารถทำงานได้ต่อเมื่อโพรเซสที่อยู่ในแถวพร้อมของงานระบบ งานโต้ตอบ และงานแก้ไขข้อมูล ได้ทำงานจนเสร็จสมบูรณ์แล้ว และในกรณีที่มีโพรเซสใหม่เข้ามาสู่แถวพร้อมของงานแก้ไขข้อมูล ในขณะที่โพรเซสแบบกลุ่มกำลังทำงานอยู่ โพรเซสที่ทำงานแบบกลุ่มจะถูกแทรกกลางทันที หรืออาจจัดแบ่งเวลาของหน่วยประมวลผลให้แต่ละแถว โดยที่แต่ละแถวก็จะไปจัดแบ่งปันเวลาที่ได้รับกันเอง เช่น แถวพร้อมของการทำงานแบบโต้ตอบอาจจะได้รับช่วงเวลาในการใช้งานซีพียูถึง 80 เปอร์เซ็นต์ ในขณะที่แถวพร้อมของการทำงานแบบกลุ่มได้รับช่วงเวลาดังกล่าวเพียง 20 เปอร์เซ็นต์

6.5.1 การจัดตารางการทำงานแบบจัดลำดับหลายชั้นแบบเลื่อนชั้นได้ (Multilevel Feedback Queue Scheduling)

โดยปกติแล้วในวิธีการจัดตารางการทำงานแถวคอยหลายชั้น (Multilevel queue scheduling) โพรเซสที่เข้าสู่ระบบจะถูกกำหนดแถวที่แน่นอนตลอดการทำงาน โดยไม่อาจเปลี่ยนแถวได้อีกเลย ซึ่งวิธีดังกล่าวไม่ยืดหยุ่นนัก การจัดตารางการทำงานแบบจัดลำดับหลายชั้นแบบเลื่อนชั้นได้นี้ จะมีวิธีการในการทำงานที่แก้ข้อเสียดังกล่าว

โดยเมื่อโพรเซสใดใช้เวลาในการทำงานกับหน่วยประมวลผลกลางนานเกินไป โพรเซสนั้นจะถูกย้ายลงไปในแถวที่มีค่าศักดิ์ต่ำกว่าแถวเดิม วิธีนี้จะทำให้โพรเซสที่เน้นการรับส่งข้อมูล และโพรเซสโต้ตอบถูกจัดอยู่ในแถวพร้อมที่มีศักดิ์สูงขึ้น ในทำนองเดียวกันเมื่อโพรเซสใดโพรเซสหนึ่งรอคอยอยู่เป็นเวลานานมากแล้วในแถวที่มีศักดิ์ต่ำ โพรเซสนั้นก็สามารถที่จะย้ายไปต่อในแถวที่มีศักดิ์สูงขึ้นได้ ซึ่งวิธีการดังกล่าวเป็นรูปแบบหนึ่งของการเพิ่มศักดิ์ ตัวอย่างเช่น ระบบที่มีแถวพร้อม 3 แถว คือ แถวที่ 0, 1 และ 2 ตามลำดับ (จากภาพที่ 6.8)



ภาพที่ 6.8 แถวคอยแบบป้อนกลับหลายระดับ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.276)

โดยตัวจัดตารางการทำงานจะเริ่มจัดให้โพรเซสที่อยู่ในแถวที่ 0 ทำงานจนเสร็จหมดทั้งแถวเสียก่อน จากนั้นจึงจะเริ่มจัดให้โพรเซสที่อยู่ในแถวที่ 1 ได้เข้ารับการทำงานต่อไป ในทำนองเดียวกันการทำงานของโพรเซสในแถวที่ 2 เริ่มได้ก็ต่อเมื่อโพรเซสในแถวที่ 0 และที่ 1 ทำงานจนเสร็จหมดแล้ว และในขณะที่โพรเซสที่อยู่ในแถวที่ 2 กำลังทำงานอยู่ ถ้ามีโพรเซสใหม่เข้ามาในแถวที่ 1 โพรเซสที่กำลังทำงานอยู่ในแถวที่ 2 จะถูกแทรกกลางคัน หรือในขณะที่โพรเซสในแถวที่ 1 กำลังทำงานอยู่นั้น ได้มีโพรเซสใหม่เข้ามาในแถวที่ 0 โพรเซสที่กำลังทำงานอยู่ในแถวที่ 1 ก็จะถูกแทรกกลางคันเช่นเดียวกัน

ถ้ามีโพรเซสใหม่เข้ามาในระบบ โพรเซสนั้นจะถูกจัดให้เข้าทำงานในแถวที่ 0 แต่ถ้าไม่สามารถทำงานให้เสร็จสมบูรณ์ได้ภายใน 1 ส่วนแบ่งเวลาของแถวที่ 0 (ซึ่งมีค่าเท่ากับ 8 มิลลิวินาที) โพรเซสนั้นจะถูกเลื่อนลงไปต่อข้างท้ายของแถวที่ 1 และเมื่อโพรเซสในแถวที่ 0 ทำงานจนเสร็จหมดแล้ว ตัวจัดตารางการทำงานก็จะเริ่มมาจัดให้โพรเซสที่อยู่ในแถวที่ 1 ได้ทำงานต่อไป โดยแถวที่ 1 นี้ (ส่วนแบ่งเวลามีค่าเท่ากับ 16 มิลลิวินาที) ซึ่งถ้ามีโพรเซสใดไม่อาจทำงานให้เสร็จสมบูรณ์ได้ภายใน 1 ส่วนแบ่งเวลาที่กำหนด โพรเซสนั้นก็จะถูกนำไปต่อท้ายของแถวที่ 2 ซึ่งเป็นแถวสุดท้ายและมีการทำงานแบบมาก่อน- ได้ก่อน โดยโพรเซสที่อยู่ในแถวสุดท้ายนี้จะได้ทำงานก็ต่อเมื่อ แถวที่ 0 และ 1 ว่างแล้ว

จากขั้นตอนวิธีในการทำงานข้างต้น จะพบว่า โพรเซสที่ใช้ช่วงประมวลผลไม่เกิน 8 มิลลิวินาที เสมือนมีศักดิ์สูงที่สุด (เช่น โพรเซสที่ใช้เวลาส่วนใหญ่ทำงานด้านรับส่งข้อมูล) ส่วนโพรเซสที่ใช้ช่วงประมวลผลระหว่าง 8 – 24 มิลลิวินาที ก็เสมือนมีศักดิ์ต่ำลงมาอีกชั้นหนึ่ง และสำหรับโพรเซสที่ต้องใช้ช่วงเวลาประมวลผลนาน (มากกว่า 24 มิลลิวินาที) ก็จะเคลื่อนลงสู่แถวที่ 2 ซึ่งมีการจัดตารางแบบมาก่อน-ได้ก่อน เสมือนมีศักดิ์ต่ำที่สุด โดยทั่ว ๆ ไปแล้ว การทำงานโดยวิธีการจัดตารางการทำงานแบบจัดลำดับหลายชั้นแบบเลื่อนชั้นได้นี้ ถูกกำหนดโดยพารามิเตอร์ดังนี้

- จำนวนแถวพร้อม
- ขั้นตอนวิธีในการจัดตารางการทำงานของแต่ละแถว
- วิธีที่จะใช้ในการพิจารณาเพื่อที่จะยกระดับให้โพรเซสที่มีค่าศักดิ์สูงขึ้น
- วิธีที่จะใช้ในการพิจารณาเพื่อที่จะลดระดับให้โพรเซสที่มีค่าศักดิ์น้อยลง
- วิธีที่จะใช้ในการพิจารณาว่า เมื่อโพรเซสเข้ามาในระบบ ควรจะให้อยู่ในแถวใด

จากนิยามของตัวจัดตารางการทำงานดังกล่าวข้างต้น จะพบว่า เป็นการรวบรวมเอาวิธีการในการจัดตารางหลายวิธีเข้าไว้ด้วยกัน ทำให้การทำงานวิธีจัดลำดับหลายชั้นแบบเลื่อนชั้นได้นี้ สามารถที่จะเลือกใช้ขั้นตอนวิธีที่เหมาะสมกับการทำงานในช่วงของการออกแบบ แต่อย่างไรก็ตามถึงแม้ว่าวิธีการจัดตารางการทำงานแบบนี้จะเป็นวิธีที่ใช้ได้ทั่วไปที่สุด แต่ต้องมีการพิจารณาเลือกค่าพารามิเตอร์ต่าง ๆ ให้เหมาะสมกับสภาวะของระบบต่าง ๆ ทั้งยังเป็นระบบที่ซับซ้อนที่สุดอีกด้วย

6.6 การจัดตารางการทำงานสำหรับหลายหน่วยประมวลผล

ถ้าหน่วยประมวลผลมีหลายแบบ (เรียกว่า Heterogeneous system) วิธีการจัดตารางก็จะถูกจำกัดมาก แต่ละหน่วยประมวลผลก็จะมีแถวคอยเป็นของตนเอง เพราะโพรเซสต่าง ๆ ย่อมมีลักษณะเฉพาะที่จะทำงานได้ในหน่วยประมวลผลแบบหนึ่งแบบเดียว เช่น โปรแกรมที่เขียนขึ้นด้วยภาษา Assembly ของ VAX จะไม่สามารถทำงานบนเครื่อง IBM ได้ เป็นต้น โพรเซสจึงต้องเข้าไปทำงานตามหน่วยประมวลผลที่เหมาะสม โดยแยกแถวคอยกันเอง

ถ้าหน่วยประมวลผลเป็นแบบเดียวกันหมด (เรียกว่า Homogenous system) สามารถเฉลี่ยแบ่งงานกันทำได้ดีขึ้น โดยอาจให้มีแถวคอยแยกแต่ละหน่วยประมวลผล แต่วิธีนี้อาจทำให้หน่วยประมวลผลบางตัวว่างงาน ในขณะที่บางตัวทำงานหนัก ดังนั้นจึงควรใช้แถวคอยร่วมแถวเดียวกันให้ทุก ๆ โพรเซสเข้าแถวคอยแถวเดียวกันหมด เมื่อมีหน่วยประมวลผลใดว่างก็จะให้รับงานไปจากแถวคอยนี้ การจัดแถวคอยร่วมนี้ อาจแบ่งได้เป็น 2 วิธี

1. ให้หน่วยประมวลผลแต่ละตัวจัดตารางการทำงานเอง โดยเลือกงานจากแถวคอยเดียวกัน ปัญหาหลักของวิธีนี้คือ การที่หน่วยประมวลผลใช้ข้อมูลร่วมกัน (แถวคอยร่วมกัน) ย่อมต้องการการประสานงานที่ดี เพื่อป้องกันปัญหาเขตวิกฤต จำเป็นต้องทำให้แน่ใจว่าจะไม่มีหน่วยประมวลผลใดเลือกโพรเซสซ้ำกัน

2. กำหนดให้หน่วยประมวลผลหนึ่งมีหน้าที่จัดตารางการทำงานโดยเฉพาะ คอยจัดตารางการทำงานให้ทุก ๆ หน่วยที่เหลือ วิธีนี้เรียกว่า วิธีเจ้านายและทาส (Master-slave) หรือเรียกว่าการทำงานแบบหลายหน่วยประมวลผลชนิดไม่สมมาตร (Asymmetric multiprocessing)

6.7 การจัดตารางการทำงานแบบตอบสนองฉับพลัน

การจัดตารางการทำงานแบบตอบสนองฉับพลัน แบ่งเป็น 2 ประเภท คือ

1. Hard Real-Time System ต้องการเวลาที่คงที่แน่นอนตายตัว โดยทั่วไปโพรเซสจะได้รับเวลาจำนวนหนึ่งเพื่อนำไปทำงานให้เสร็จ ตัวจัดตารางการทำงานจะให้สิทธิโพรเซส เพื่อรับประกันว่าโพรเซสจะทำงานเสร็จตามเวลา หรือไม่ก็ปฏิเสธการร้องขอของโพรเซสนั้น ถ้ามั่นใจว่าจะทำงานไม่เสร็จได้ตามเวลา การทำงานแบบนี้ถูกเรียกว่า การจองทรัพยากร (Resource reservation)

2. Soft Real-Time System เป็นระบบที่ทำงานโดยไม่ต้องมีเวลามาจำกัด นั่นหมายถึงจะมีโพรเซสอยู่โพรเซสหนึ่งมีลำดับความสำคัญสูงกว่าโพรเซสอื่น การทำงานแบบ Soft Real-Time system อาจจะไม่เหมาะกับ Time sharing เพราะเกิดการล่าช้าหรือปัญหาการอดตาย แต่เหมาะกับระบบทั่ว ๆ ไป ที่สนับสนุนด้านมัลติมีเดีย กราฟฟิก เป็นต้น

เพื่อที่จะลดเวลาในการหยุดโพรเซสหนึ่งเพื่อให้อีกโพรเซสหนึ่งทำงาน (Dispatch latency) เราต้องอนุญาตให้โปรแกรมเรียกระบบที่สามารถแทรกกลางคันได้ วิธีหนึ่งที่ใช้ได้คือ การแทรกโปรแกรม โดยทำการเรียกระบบที่ทำงานในระยะยาวและเรียกว่าจุดแทรกกลางคัน (Preemption point) เพื่อตรวจสอบว่าโพรเซสที่มีลำดับความสำคัญสูงต้องได้ทำงานก่อน เมื่อโพรเซสดังกล่าวเสร็จงาน โพรเซสที่ถูกขัดจังหวะจึงจะได้ทำงานต่อ

อะไรจะเกิดขึ้นถ้าโพรเซสที่มีลำดับความสำคัญสูงกว่าต้องการอ่านหรือปรับปรุงข้อมูลใน Kernel ในขณะที่โพรเซสที่มีลำดับความสำคัญต่ำกว่ากำลังทำงาน โพรเซสที่มีลำดับความสำคัญสูงกว่าควรจะรอให้โพรเซสที่มีลำดับความสำคัญต่ำกว่าทำงานเสร็จก่อน สถานการณ์นี้ถูกเรียกว่า การกลับสิทธิ (Priority inversion) ปัญหานี้แก้ได้โดยสนธิสัญญาการสืบทอดศักดิ์ (Priority-inheritance protocol) ซึ่งโพรเซสเหล่านี้ (โพรเซสซึ่งกำลังเข้าถึงทรัพยากรที่โพรเซสที่มีลำดับความสำคัญสูงต้องการ) สืบทอดลำดับความสำคัญที่สูงจนกระทั่งทำงานเสร็จลำดับความสำคัญจะถูกกลับ (Revert) ไปเป็นดังเดิม โดยระยะที่เกิดการขัดแย้งกัน (Conflict phase) ของ Dispatch latency มี 2 องค์ประกอบดังนี้

- เกิดการแทรกกลางคันของโพรเซสที่กำลังทำงานใน Kernel
- โพรเซสที่มีลำดับความสำคัญต่ำปลดปล่อยทรัพยากรให้โพรเซสที่มีลำดับความสำคัญสูง

6.8 การประเมินอัลกอริทึม

จะเห็นว่ามีวิธีการจัดตารางได้หลายวิธี แต่ละวิธีมีตัวแปรและลักษณะเฉพาะ ทำให้การเลือกวิธีที่เหมาะสมหรือดีที่สุดทำได้ยาก ขั้นแรกจำเป็นต้องกำหนดคุณสมบัติก่อน ว่าต้องการคุณสมบัติใดมีค่าเท่าใด เช่น ให้ได้ประสิทธิผลการใช้ซีพียูสูงสุดที่เวลาตอบสนองนานที่สุดไม่เกิน 1 วินาที และให้มีอัตรางาน (Throughput) เสร็จสูงที่สุด โดยที่วงรอบการทำงาน (Turnaround time) โดยเฉลี่ยเป็นสัดส่วนโดยตรงกับการประมวลผลจริง

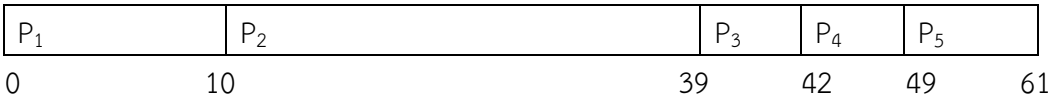
6.8.1 การกำหนดโมเดล (Deterministic Modeling)

วิธีนี้ทำโดยการกำหนดกลุ่มงานทดสอบขึ้นมาหนึ่งกลุ่ม แล้วคำนวณค่าคุณสมบัติของวิธีการจัดตารางแต่ละแบบ ตัวอย่างเช่น กำหนดกลุ่มงานดังนี้

โพรเซส (Process)	เวลาทำงาน (Burst Time)
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

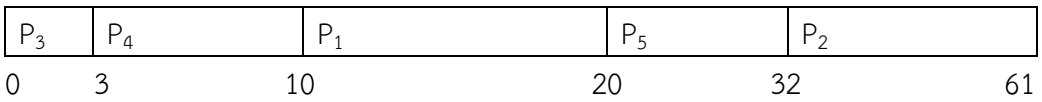
ให้ทั้ง 5 โพรเซสมาถึงระบบในเวลาเดียวกัน ที่เวลา 0 ตามลำดับ พิจารณาใช้วิธีการจัดตารางการทำงาน 3 แบบ คือ FCFS, SJF, RR (โดยมีส่วนแบ่งเวลา 10 มิลลิวินาที) แล้วคำนวณหาเวลารอคอยเฉลี่ยต่ำที่สุด

วิธีมาก่อน-ได้ก่อน (FCFS)



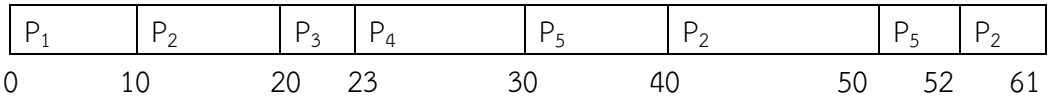
เวลารอคอยเฉลี่ย = (0 + 10 + 39 + 42 + 49) / 5 = 28 มิลลิวินาที

วิธีสั้นที่สุดได้ก่อน (SJF)



เวลารอคอยเฉลี่ย = (10 + 32 + 0 + 3 + 20) / 5 = 13 มิลลิวินาที

วิธีเวียนเทียน (RR) (ส่วนแบ่งเวลา = 10 มิลลิวินาที)



เวลารอคอยเฉลี่ย = (0 + 32 + 20 + 23 + 40) / 5 = 23 มิลลิวินาที

จากตัวอย่างนี้จะเห็นว่า วิธีงานที่สั้นที่สุดได้ก่อน มีค่าต่ำที่สุด (ดีที่สุด) คือ เพียงครึ่งเดียวของวิธีมาก่อน-ได้ก่อนและวิธีเวียนเทียนมีค่าปานกลาง

วิธีการกำหนดโมเดลนี้ เป็นวิธีที่ง่ายและสะดวก ผลลัพธ์ที่ได้ก็สามารถเปรียบเทียบเป็นตัวเลขได้โดยตรง แต่มีข้อเสียคือ ผลลัพธ์ที่ได้อาจเป็นเฉพาะกรณีตามกลุ่มงานที่กำหนดขึ้นเท่านั้น วิธีนี้เหมาะเพียงการอธิบายการจัดตารางแบบต่าง ๆ เพื่อเป็นโมเดลการใช้งานที่เหมาะสม เช่น จากโมเดลต่าง ๆ ที่กล่าวมาพอจะสรุปได้จากตัวอย่างว่า วิธีสั้นที่สุดได้ทำงานก่อน ให้เวลารอคอยเฉลี่ยสั้นที่สุด

6.8.2 การวิเคราะห์แถว (Queuing Models)

ระบบคอมพิวเตอร์อาจถือได้ว่าเป็นเครือข่ายของผู้ให้บริการ (Network of servers) ผู้ให้บริการแต่ละตัวมีแถวคอยของตนเอง หน่วยประมวลผลกลาง (ผู้ให้บริการประมวลผล) มีแถวพร้อมของตนเอง อุปกรณ์รับส่งข้อมูลแต่ละตัว (บริการรับส่งข้อมูล) ก็มีแถวคอยอุปกรณ์ของตน โดยใช้ค่าอัตราการมาถึงระบบ (Arrival rate) กับเวลาในการประมวลผล (Service rate) เราสามารถคำนวณหาค่าเฉลี่ยของประสิทธิผล ความยาวแถวคอย เวลารอคอย และอื่น ๆ ได้ เราเรียกกรรมวิธีนี้ว่า การวิเคราะห์เครือข่ายของแถวคอย (Queuing-network analysis)

<u>ตัวอย่าง</u> ให้ n เป็น ขนาดของแถวคอยเฉลี่ย (ไม่รวมโพรเซสที่กำลังทำงานอยู่) W เป็น เวลารอคอยเฉลี่ยในแถวคอย λ เป็น อัตราเฉลี่ยของจำนวนโพรเซสที่มาถึงแถวคอย (เช่น 3 โพรเซสต่อวินาที) จะเห็นได้ว่าในช่วงเวลา W ขณะที่โพรเซสหนึ่งต้องรอคอยในแถวคอย มีโพรเซสใหม่บางระบบเท่ากับ λ x W ถ้าระบบอยู่ในภาวะคงที่ จำนวนโพรเซสที่ออกจากแถวคอยจะต้องเท่ากับ จำนวนโพรเซสที่มาถึง ดังนั้น

$$n = \lambda \times W$$

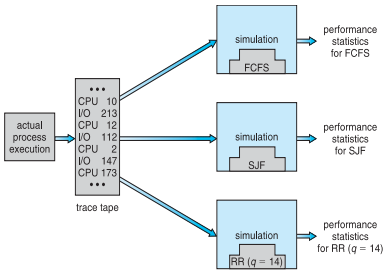
สมการนี้มีชื่อว่า สูตรของ Little (Little's Formula) สูตรนี้มีประโยชน์มากในการคำนวณเพราะเป็นจริงเสมอ ไม่ว่าจะเป็นวิธีการจัดตารางแบบใด และการกระจายตัวของข้อมูลแบบใดก็ตามสามารถใช้สูตรของ Little หาค่าตัวแปรหนึ่งในสามตัว เมื่อทราบคู่ตัวแปรอีกสองตัว เช่น เรารู้ว่าโพรเซสมาถึงระบบโดยเฉลี่ย 7 โพรเซสต่อวินาทีและแถวคอยมีความยาวเฉลี่ย 14 โพรเซส ดังนั้นเวลารอคอยเฉลี่ยของแต่ละโพรเซสจะมีค่าเท่ากับ 2 วินาที

6.8.3 การจำลองสถานการณ์ (Simulations)

เพื่อให้ได้ผลลัพธ์ถูกต้องใกล้เคียงมากขึ้น เราอาจจำลองเหตุการณ์จริงขึ้น โดยเขียนโปรแกรมตัวแบบของระบบคอมพิวเตอร์ และจำลองอุปกรณ์ต่าง ๆ ในระบบด้วยโครงสร้างข้อมูลที่เหมาะสม มีโปรแกรมนาฬิกาให้โปรแกรมทำงานเสมือนมีเหตุการณ์เกิดขึ้นจริง (โดยใช้ตัวแปรต่าง ๆ แทนความจริง) ขณะที่ระบบจำลองทำงาน เราสามารถเก็บสถิติต่าง ๆ ของระบบ เพื่อนำมาวิเคราะห์ประสิทธิภาพของวิธีการจัดตารางแบบต่าง ๆ ได้

ข้อมูลของโพรเซสที่เข้าระบบ สร้างโดยใช้ตัวแปรสร้างที่สุ่มค่าตัวเลขขึ้นมา (Random-number generator) ค่าที่มากที่สุด ได้แก่ ช่วงเวลาประมวลผล (CPU-burst time) เวลามาถึง (Arrival) เวลางานเสร็จ (Departure) เป็นต้น การสร้างค่าเหล่านี้ ทำโดยการใช้ความน่าจะเป็น (เช่น แบบคงที่ แบบ Exponential และแบบ Poisson) หรือแบบทั่วไป ซึ่งจะสามารถเลือกได้มากแบบกว่าการใช้วิธีวิเคราะห์แถวคอย เพราะไม่ต้องคำนวณผลลัพธ์เอง ผลลัพธ์จะได้จากโปรแกรมแบบจำลองการทำงานแทนระบบจริง

ผลลัพธ์จากการใช้ตัวแบบสุ่มเหล่านี้อาจไม่ถูกต้องเท่าที่ควร เนื่องจากข้อมูลจริงแตกต่างจากข้อมูลที่ได้จากโปรแกรม ดังนั้นเราอาจใช้ข้อมูลจริงจากระบบจริงเก็บเข้าไปในเทป (Trace tape) แล้วนำมาเป็นข้อมูลเข้าโปรแกรมแบบจำลองเพื่อศึกษาเปรียบเทียบ วิธีการจัดตารางแบบต่าง ๆ ให้ใกล้เคียงความจริงมากที่สุด ดังภาพที่ 6.9



ภาพที่ 6.9 การประเมินโดยจำลองการตัวจัดตาราง CPU
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.303)

การจำลองเหตุการณ์จริงนี้อาจมีค่าใช้จ่ายมากได้ เช่น ต้องใช้เวลาประมวลผลนาน ยิ่งต้องการผลลัพธ์ละเอียดถูกต้องยิ่งต้องทดลองนาน นอกจากนี้การออกแบบและเขียนโปรแกรมนี้ยังเสียเวลามากอีกด้วย

6.8.4 การปฏิบัติจริง (Implementation)

วิธีนี้มีปัญหาสำคัญ คือ ค่าใช้จ่ายสูงมาก ไม่เพียงแต่ค่าใช้จ่ายในการเขียนโปรแกรมและแก้ไขระบบปฏิบัติการเท่านั้น แต่ยังมีผลกระทบต่อผู้ใช้โดยตรงด้วย เพราะจะต้องมีการแก้ไขระบบบ่อย ๆ นอกจากนี้การเปรียบเทียบวิธีต่าง ๆ ด้วยวิธีนี้อาจได้ผลไม่ถูกต้อง เพราะสภาพแวดล้อมของระบบอาจเปลี่ยนไป

เราอาจกำหนดตัวแปรในระบบให้ผู้ควบคุมระบบสามารถแก้ไขวิธีจัดตารางได้ตลอดเวลา (ขณะทำงานจริง) เช่น ถ้าต้องการใช้พิมพ์เช็คอย่างเร่งด่วน (ปกติถือเป็นงานแบบกลุ่มและมีลำดับความสำคัญต่ำ) ผู้ควบคุมระบบอาจกำหนดให้งานนี้มีศักดิ์สูงเป็นการชั่วคราว แต่เป็นที่น่าเสียดายว่ามีน้อยระบบที่มีตัวแปรแบบนี้

6.9 สรุป

การจัดตารางการทำงานของซีพียูให้ได้ผลดี ต้องทราบถึงลักษณะการทำงานของโพรเซส ซึ่งโพรเซสโดยทั่วไปจะทำงานเป็นวงจรคือ ทำงานในซีพียูและรอคอยการรับส่งข้อมูลสลับกันไป โดยเริ่มต้นที่ทำงานในซีพียูก่อนเรียกว่า ช่วงประมวลผล แล้วก็จะหยุดเพื่อทำงานในอุปกรณ์ที่มีการรับส่งข้อมูลเรียกว่าช่วงรับส่งข้อมูล และต่อมาจะกลับมาช่วงประมวลผลอีกสลับกันไปจนสุดท้ายจะเป็นช่วงประมวลผล และก็สิ้นสุดโพรเซสจัดเวลาซีพียู เป็นงานของการจัดเวลาใช้งานซีพียู โดยมีหน้าที่หลักก็คือการตัดสินใจเลือกงานเข้ามาใช้งาน โดยมี Dispatcher เป็นเครื่องมือในการนำเอางานที่ได้รับคัดเลือกนี้เข้าและออกจากการใช้ซีพียู การมาก่อนบริการก่อน (FCFS) เป็นอัลกอริทึมแบบให้สิทธิ์ก่อน (Preemptive) ที่ง่ายที่สุดสำหรับการจัดเวลาซีพียู แต่ก็เป็นระบบการจัดการที่ทำให้งานที่ต้องการใช้ซีพียูเพียงแค่ระยะเวลาสั้น ๆ ต้องคอยงานที่ใช้ซีพียูนาน ๆ ส่วนงานสั้นได้ทำก่อน (SJF) ก็เป็นอัลกอริทึมที่สามารถเป็นทั้งแบบให้สิทธิ์ก่อนและไม่ให้สิทธิ์ก่อน SJF จะให้ค่าเฉลี่ยของการรอคอย (Waiting time) สั้นที่สุด แต่ในการใช้งานจริงจะมีปัญหาในเรื่องของการวัดระยะซีพียูถัดไปของแต่ละงาน SJF มีการทำงานคล้ายกับการมีลำดับขั้นความสำคัญของงาน โดยงานที่มีเวลาซีพียูสั้น ๆ จะเป็นงานที่มีความสำคัญมาก ทำให้มีโอกาสเข้าไปใช้ซีพียูก่อน ซึ่งอาจทำให้เกิดปัญหาการตกค้างของงานที่มีความสำคัญต่ำ ถ้าหากว่าระบบคอมพิวเตอร์มีงานมาก การแก้ไขก็ด้วยการใช้วิธีเพิ่มความสำคัญให้กับงานตามระยะเวลาที่ต้องคอย

อัลกอริทึมแบบวนรอบ เป็นระบบการจัดเวลาแบบให้สิทธิ์ก่อนเหมาะสมกับระบบคอมพิวเตอร์แบบแบ่งเวลา หรือการทำงานแบบ Interactive เพราะจะทำให้มีเวลาวนรอบที่คงที่แน่นอน เนื่องจากว่าอัลกอริทึมแบบวนรอบมีการกำหนดช่วงเวลาที่จำกัดในการเข้าใช้ซีพียูของแต่ละงานที่คงที่ที่เรียกว่าเวลาควอนตัม (Quantum time) ซึ่งงานทุกงานจะมีโอกาสได้รับซีพียูมาใช้อย่างมากก็เพียงแค่ระยะเวลาของควอนตัมที่กำหนดเท่านั้น ซึ่งการกำหนดค่าของควอนตัมนี้ ต้องมีการกำหนดให้พอดี ถ้ามีการกำหนดที่ยาวเกินไปการทำงานก็จะมีประสิทธิภาพใกล้เคียงกับ FCFS แต่ถ้าควอนตัมถูกกำหนดไว้สั้นเกินไป ระบบก็จะเสียเวลาไปกับการนำงานเข้าและออก หรือเวลาในการทำ Context switch มากเกินไปจนทำให้อัตรางานของระบบต่ำลง

คิวแบบหลายระดับ อาจมีการรวมเอาอัลกอริทึมหลายชนิดเข้าไว้ด้วยกันเพื่อให้สามารถรองรับระบบงานที่มีงานหลายประเภทมากขึ้น โดยจะให้บริการแก่งานต่าง ๆ อย่างเท่าเทียมกันตามระดับความสำคัญของงานนั้น ๆ และการใช้คิวแบบ Multilevel feedback ก็จะยอมให้มีการปรับเปลี่ยนระดับความสำคัญของตัวงานได้ ด้วยการยอมให้งานสามารถเลื่อนขึ้นหรือลงในคิวแบบหลายระดับเพื่อแก้ปัญหาของการที่งานบางชนิดจะไม่มีโอกาสได้รับซีพียูมาใช้เลย

การตัดสินใจว่าจะนำเอาอัลกอริทึมชนิดไหนมาใช้ก็มีวิธีการอยู่มากมาย การใช้วิธีคำนวณจากสูตรนั้นมีประโยชน์ในการคำนวณหาประสิทธิภาพของระบบโดยรวมแบบคร่าว ๆ ในขณะที่วิธีการจำลองแบบ

หรือ Simulation นั้น จะสามารถทำให้การตัดสินใจมีความถูกต้องมากขึ้น แต่ก็ต้องแลกมาด้วยแรงงานและเวลา ซึ่งจะทำให้มีผลกระทบต่อต้นทุนที่สูงขึ้น แต่ในบางระบบคอมพิวเตอร์ที่ต้องการประสิทธิภาพสูงสุดโดยไม่สนใจว่าต้องใช้ต้นทุนเท่าไร วิธี Simulation ก็จะต้องเป็นเรื่องที่หลีกเลี่ยงไม่ได้ อย่างไรก็ตาม วิธีที่ดีที่สุดก็คือ การสร้างอัลกอริทึมต่าง ๆ ขึ้นมาและทดลองใช้ระบบคอมพิวเตอร์ที่ทำงานกับงานจริง ในปัจจุบันสิ่งที่เป็นความต้องการของระบบคอมพิวเตอร์ก็คือ ความต้องการที่จะให้ระบบปฏิบัติการมีความสามารถในการปรับเปลี่ยนหรือปรับแต่งอัลกอริทึม ที่ใช้ได้ตามลักษณะของงานที่เปลี่ยนแปลงไป

แบบฝึกหัดท้ายบทที่ 1

1. จงอธิบายการให้สิทธิ์การจัดเวลา มีหลักการทำงานอย่างไร

2. จงอธิบายถึงข้อพิจารณาในการจัดเวลาของซีพียูมีอะไรบ้าง

3. จงอธิบายหลักการทำงานของอัลกอริทึมการจัดเวลาต่อไปนี้มาพอเข้าใจ

3.1. การจัดเวลาแบบมาก่อนได้ก่อน

3.2. การจัดเวลาแบบงานสั้นทำก่อน

3.3. การจัดเวลาตามลำดับความสำคัญ

3.4. การจัดเวลาแบบวนรอบ

3.5. การจัดเวลาแบบคิวหลายระดับ

4. การกำหนดการใช้ซีพียู โดยใช้อัลกอริทึมในข้อ 3 แต่ละอัลกอริทึมมีปัญหาสำคัญที่อาจเกิดขึ้นได้คืออะไร เกิดขึ้นได้อย่างไร และมีวิธีแก้ปัญหาที่เกิดขึ้นได้อย่างไร จงอธิบายพร้อมยกตัวอย่างประกอบ

5. ถ้าให้โพรเซสเข้าสู่ระบบตามตารางดังนี้

Process	Arrival Time	Burst Time (ms)
P1	0	5
P2	5	20
P3	10	10
P4	15	15

ถ้าโพรเซสเข้ามาตามลำดับคือ P1, P2, P3, P4 จงแสดงถึงการจัดเวลาซีพียูของโพรเซสตามอัลกอริทึมแบบ FCFS, SJF, Priority (ถ้าตัวเลขน้อยคือ ค่า Priority สูง) และ วิธีการของ RR เมื่อ Quantum = 1 จงหาค่าเฉลี่ยของการคอยในคิวของแต่ละโพรเซสโดยใช้อัลกอริทึมแบบ FCFS, SJF และ RR

6. ถ้าให้ set ของโพรเซสเข้ามาสู่ระบบ (ใน Ready queue) ตามตารางดังนี้

Process	Burst time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

ถ้าโพรเซสเข้ามาตามลำดับคือ P1, P2, P3, P4, P5

6.1. จงวาด Gantt Chart สำหรับการแสดงลำดับการอยู่ใน Ready Queue และการทำงานในรูปแบบของ FCFS, SJF, Non-preemptive Priority (ถ้าตัวเลขน้อยคือ ค่า Priority สูง) และ วิธีการของ RR เมื่อ quantum =1

6.2. จงหาค่า Turnaround Time ของแต่ละโพรเซสใน Algorithm ต่าง ๆ

6.3. จงหาค่า Waiting Time ของแต่ละโพรเซสใน Algorithm ต่าง ๆ

6.4. Scheduling Algorithm ในข้อใด (รูปแบบใด) ให้ค่า Average Minimum Time น้อยที่สุด

7. โพรเซสมาถึงระบบโดยเฉลี่ย 10 โพรเซสต่อวินาทีและแถวคอยมีความยาวเฉลี่ย 20 โพรเซส จงหาเวลารอคอยเฉลี่ยของแต่ละโพรเซสจะมีค่าเท่ากับเท่าไร

บรรณานุกรมประจำบทที่ 6

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

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

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

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

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

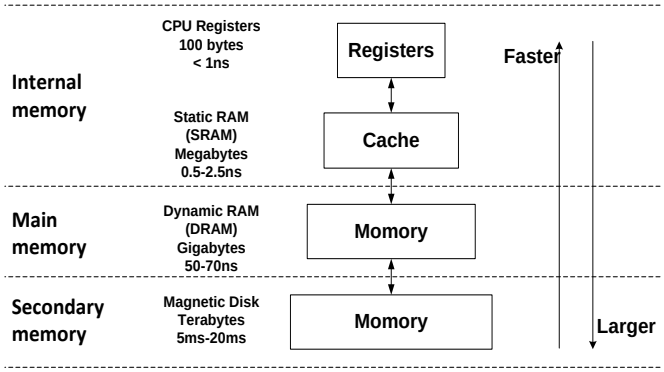
บทที่ 7
การจัดการหน่วยความจำ

การจัดการหน่วยความจำ เป็นงานอย่างหนึ่งของระบบปฏิบัติการ ถ้าหน่วยความจำมีปริมาณมากขีดความสามารถในการทำงานก็จะเพิ่มขึ้นด้วย โปรแกรมที่มีความสลับซับซ้อนและมีความสามารถมากก็ต้องการหน่วยความจำมาก ความต้องการหน่วยความจำของโปรแกรมและของโครงสร้างข้อมูลที่ต้องทำงานร่วมกับโพรเซสหรือระบบปฏิบัติการก็ยังเป็นที่ต้องการอยู่อีกมาก ดังนั้นจึงเป็นหน้าที่หลักของระบบปฏิบัติการที่จะต้องทำการจัดการหน่วยความจำที่มีอยู่ให้เกิดประโยชน์สูงสุด

หน่วยความจำจึงเป็นส่วนที่สำคัญที่สุดในระบบคอมพิวเตอร์ ถือเป็นศูนย์กลางการดำเนินการด้านต่าง ๆ ในระบบคอมพิวเตอร์ให้เป็นไปอย่างราบรื่นและมีประสิทธิภาพสูงสุด โดยภายในหน่วยความจำจะมีการทำงานหลายส่วน เช่น การทำงานของโปรแกรมจำนวนมาก ซึ่งต้องมีการแบ่งพื้นที่การใช้งานและวิธีการจัดการด้านต่าง ๆ และถ้าเราสามารถประหยัดเนื้อที่ของหน่วยความแม้นเพียงจำนวนไม่กี่พันไบต์อาจจะสร้างความแตกต่างระหว่างการทำงานของแต่ละโปรแกรม

7.1 ประเภทของหน่วยความจำ

หน่วยความจำ เป็นทรัพยากรสำคัญของระบบคอมพิวเตอร์ ทำหน้าที่เก็บข้อมูลต่าง ๆ รวมทั้งคำสั่ง และผลลัพธ์ที่ได้จากการทำงาน โดยลำดับชั้นของหน่วยความจำแบ่งออกเป็น 3 กลุ่ม คือ



ภาพที่ 7.1 จัดลำดับหน่วยความจำ

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

2. หน่วยความจำหลัก (Main memory) หน่วยความจำหลัก เป็นหน่วยความจำความเร็วสูงใช้สำหรับเก็บคำสั่งและข้อมูลระหว่างการทำงาน โดยเป็นหน่วยความจำที่ซีพียูสามารถเข้าถึงได้โดยตรง และรวดเร็ว

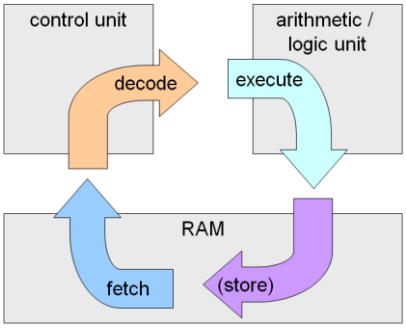
3. หน่วยความจำสำรอง (Secondary memory) หน่วยความจำสำรองเป็นหน่วยความจำที่มีความเร็วน้อยกว่าหน่วยความจำสองประเภทแรก ใช้สำหรับเก็บข้อมูลที่มีขนาดใหญ่ และเป็นข้อมูลที่ยังไม่ต้องการนำมาประมวลผล

7.2 แนวคิดพื้นฐานการจัดการหน่วยความจำหลัก

หน้าที่ของระบบปฏิบัติการในการจัดการกับหน่วยความจำหลักมี 4 ประการคือ

1. ควบคุมดูแลสถานะของแต่ละตำแหน่งของหน่วยความจำหลัก
2. ตัดสินว่าควรจัดสรรหน่วยความจำหลักขนาดเท่าไร ให้กับงานใด ณ ตำแหน่งใดของหน่วยความจำหลัก
3. จัดสรรหน่วยความจำหลักให้งานที่ได้เลือกแล้ว
4. ปลดปล่อยหน่วยความจำหลักให้ว่าง เมื่อทำงานเสร็จแล้ว

คำสั่งที่จะถูกดำเนินการได้โดยซีพียู จะต้องถูกดึงมาและเก็บที่ตำแหน่งในหน่วยความจำ รูปแบบการทำงานของรอบคำสั่งเครื่องและการปฏิบัติงานตามคำสั่ง (Instruction-execution cycle) จะมีขั้นตอนการทำงาน ดังนี้



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

ขั้นตอนที่ 1 ไปนำมา (Fetch) คือ การเริ่มต้นการทำงานซึ่งระบบจะทำการดึงคำสั่งแรกจากหน่วยความจำ

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

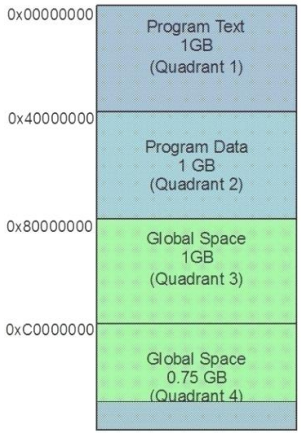
ขั้นตอนที่ 3 กระทำการ (Execution) คือ การทำงานต่อจากขั้นตอนที่ 2 ซึ่งหลังจากนั้นคำสั่งจะทำงานตามตัวดำเนินการที่ได้

ขั้นตอนที่ 4 จัดเก็บ (Store) ผลลัพธ์จะถูกเก็บกลับไปในหน่วยความจำหลักต่อไป

7.3 หน่วยความจำหลัก

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

และโดยทั่วไปแล้วนักเขียนโปรแกรมคอมพิวเตอร์ (Programmer) ล้วนแต่ต้องการระบบที่มีหน่วยความจำหลักแบบไม่จำกัด มีความเร็วสูง และมีหน่วยความจำที่มีความเสถียรสูงหรือเป็นแบบไม่ถูกลบเลือน ซึ่งหมายความว่าแม้ระบบไฟฟ้าจะขัดข้องข้อมูลต่าง ๆ ในหน่วยความจำหลักไม่สูญหายไปด้วย

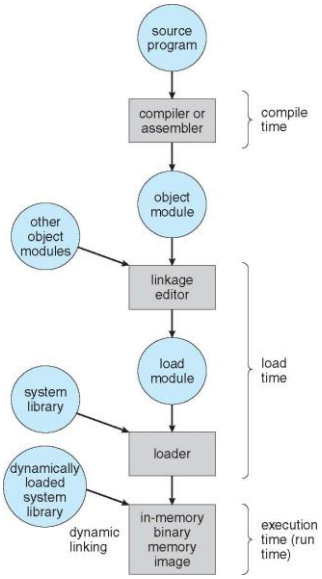


ภาพที่ 7.3 แสดง physical address ของหน่วยความจำหลัก
ที่มา: IBM, Developer Works. Retrieved June 25, 2014 from
http://www.ibm.com/developerworks/data/library/techarticle/dm-0406qi/

7.3.1 การเชื่อมโยงตำแหน่ง (Address Binding)

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

จึงได้มีการพัฒนาการเขียนโปรแกรมให้มีการใช้เลขที่อยู่เชิงสัญลักษณ์ ด้วยการกำหนดตัวระบุ เช่น ชื่อตัวแปร หรือซับรูทีน และมีการอ้างถึงหน่วยความจำหลักด้วยเลขที่อยู่สัมพัทธ์ (Relative address) ซึ่งช่วยแก้ปัญหาของการเขียนโปรแกรม โดยโปรแกรมสามารถโหลดลงตำแหน่งใดก็ได้ในหน่วยความจำหลัก



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

การจำแนกการเชื่อมโยงของคำสั่งและตำแหน่งของข้อมูล การกำหนดเลขที่อยู่ของโปรแกรม เพื่อแปลงไปเป็นเลขที่อยู่ในหน่วยความจำหลัก สามารถทำได้ในช่วงเวลาต่าง ๆ ของการทำงานของโปรแกรม มีขั้นตอนดังนี้

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

2. เวลาโหลดโปรแกรม (Load time) เมื่อนำโปรแกรมต้นฉบับที่มีการระบุตำแหน่งของหน่วยความจำหลักด้วยเลขที่อยู่สัมพัทธ์มาผ่านการแปลด้วยคอมไพเลอร์ จะได้อ็อบเจ็กของโปรแกรมที่มีการอ้างถึงตำแหน่งของหน่วยความจำหลักด้วยเลขที่อยู่สัมพัทธ์ เป็นหน้าที่ของโหลดเดอร์ ซึ่งเป็นซอฟต์แวร์ระบบที่จะนำโปรแกรมที่ต้องการเข้ามาบรรจุบนหน่วยความจำหลัก ดังนั้นทุกครั้งที่ทำการโหลดโปรแกรมมาใช้งาน โหลดเดอร์จะต้องหาตำแหน่งของหน่วยความจำหลักที่จะโหลดโปรแกรมเข้ามา โดยโหลดเดอร์ต้องติดต่อกับระบบปฏิบัติการ เพื่อให้ระบบปฏิบัติการหาพื้นที่ว่างของหน่วยความจำหลักที่มีขนาดมากพอที่จะบรรจุโปรแกรมได้ หากมีการเรียกใช้โปรแกรมหลายครั้ง โหลดเดอร์จำเป็นต้องหาตำแหน่งของหน่วยความจำหลักทุกครั้ง โดยไม่จำเป็นต้องเป็นตำแหน่งเดิม ทำให้โปรแกรมมีความยืดหยุ่นในการใช้งาน และมีคุณสมบัติของการย้ายที่อยู่ แต่ระบบจำเป็นต้องมีการกำหนดตำแหน่ง เพื่อเป็นการเชื่อมโยงระหว่างตัวแปรกับตำแหน่งที่แท้จริงของหน่วยความจำหลัก ดังนั้นการกำหนดตำแหน่งจะถูกกระทำในช่วงเวลาโหลด คือ ช่วงเวลาที่ทำการโหลดโปรแกรม โดยใช้โหลดเดอร์ย้ายที่อยู่ ทำหน้าที่สำคัญ คือ ทำการแปลจากเลขที่อยู่เชิงสัมพัทธ์ให้เป็นเลขที่อยู่เชิงสัมบูรณ์

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

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

- Absolute address แอดเดรสแท้จริงของโพรเซสที่อยู่ในพาร์ติชั่นของหน่วยความจำ
- Relative address แอดเดรสของคำสั่งหรือโปรแกรมของโพรเซสจากการคอมไพล์

7.3.2 Dynamic Loading

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

7.3.3 Dynamic Linking and Shared Libraries

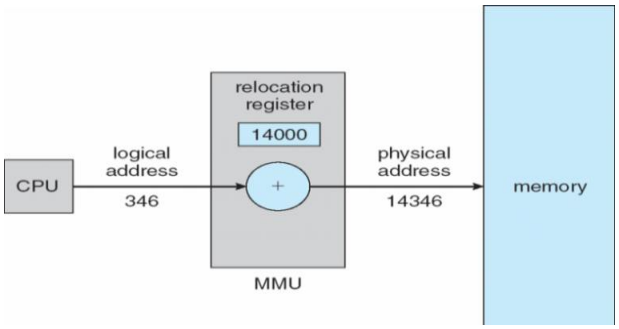
Linking จะเลื่อนออกไปจนกระทั่งอยู่ในช่วง Execution time มีชุดคำสั่งเล็ก ๆ เรียกว่า Stub ใช้สำหรับ Locate ไลบรารีของโปรแกรมย่อยที่เหมาะสมในหน่วยความจำหลัก Stub จะแทนที่ตัวเองด้วยแอดเดรสของโปรแกรมย่อยและจะทำงานตามโปรแกรมย่อยนั้น ซึ่งระบบปฏิบัติการจำเป็นต้องตรวจดูว่าโปรแกรมย่อยนั้นต้องอยู่ภายในแอดเดรสของโพรเซสในหน่วยความจำ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

7.4 ตำแหน่งที่ว่างทางกายภาพกับตำแหน่งที่ว่างทางตรรกะ

ตำแหน่งที่ถูกสร้างโดยซีพียูมักหมายถึงตำแหน่งทางตรรกะ (Logical address) ส่วนตำแหน่งในหน่วยความจำจะหมายถึง Physical address เป็นตำแหน่งบนหน่วยความจำคือ ตำแหน่งทางกายภาพ Logical และ Physical address เป็นตำแหน่งเดียวกันในช่วงของ Compile time และ Load time การเชื่อมโยง Logical address เข้ากับแต่ละ Physical address ถือเป็นหัวใจสำคัญในการจัดการหน่วยความจำหลัก

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

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



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

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

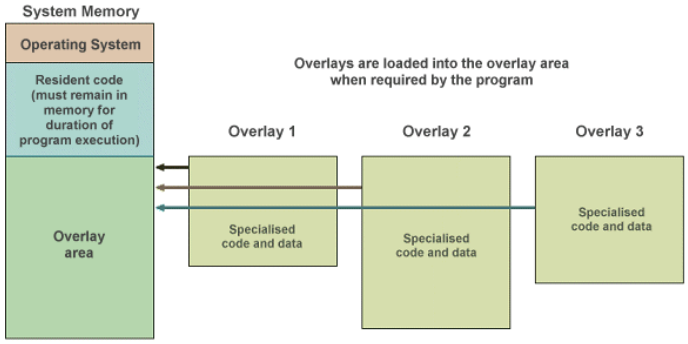
จะสังเกตได้ว่า โปรแกรมของผู้ใช้ไม่สามารถทราบค่าตำแหน่งที่แท้จริง (ทางกายภาพ) โปรแกรมอาจสร้างตัวชี้ไปยังตำแหน่ง 346 และทำการคำนวณ เก็บค่า อ่านค่า หรือเปรียบเทียบค่าในตำแหน่งอื่น ๆ กับค่าในตำแหน่ง 346 นี้ โปรแกรมของผู้ใช้ทำงานด้วยตำแหน่งทางตรรกะ ฮาร์ดแวร์ของเครื่องเป็นผู้จัดการจับคู่ตำแหน่งทางตรรกะนี้กับตำแหน่งจริง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

7.5 การจัดการหน่วยความจำหลัก

หน้าที่ของระบบปฏิบัติการในการจัดการกับหน่วยความจำหลัก คือ การควบคุมดูแลสถานะของแต่ละตำแหน่งของหน่วยความจำหลัก พร้อมตัดสินว่าควรจัดสรรหน่วยความจำหลักขนาดเท่าไร ให้กับโพรเซสใด ณ ตำแหน่งใดของหน่วยความจำหลัก และจัดสรรหน่วยความจำหลักให้โพรเซสที่ได้เลือกแล้ว ปลดปล่อยหน่วยความจำหลักให้ว่างเมื่อทำงานเสร็จแล้ว

7.5.1 วิธีการซ้อนทับ (Overlays)

โดยทั่วไปโพรเซสที่ต้องการประมวลผลด้วยซีพียู จะต้องถูกบรรจุอยู่ในหน่วยความจำหลัก ซึ่งโพรเซสจะต้องมีขนาดน้อยกว่าหรือเท่ากับขนาดของหน่วยความจำหลักลบด้วยขนาดของระบบปฏิบัติการ ดังนั้นหากมีความจำเป็นที่ต้องใช้โพรเซสที่มีขนาดมากกว่าขนาดของหน่วยความจำหลัก สามารถทำได้โดยวิธีซ้อนทับ (Overlays) ดังแสดงในภาพ 7.6

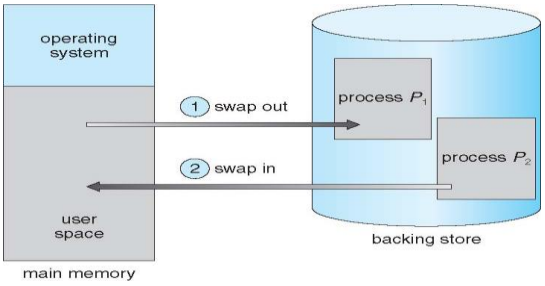


ภาพที่ 7.6 แสดงวิธีซ้อนทับ
ที่มา: TechnologyUK. Retrieved June 26, 2014 from http://www.technologyuk.net/computing/operating_systems/memory_management.shtml

โพรเซสที่จะทำงานด้วยวิธีซ้อนทับ จะต้องถูกแบ่งออกเป็นส่วนย่อยที่อิสระต่อกัน คือแต่ละส่วนไม่จำเป็นต้องถูกเรียกใช้งานพร้อมกัน โดยระบบจะนำเฉพาะส่วนของโพรเซสที่ต้องการใช้เข้ามาบรรจุในหน่วยความจำหลัก ส่วนอื่น ๆ ของโพรเซสให้เก็บไว้ในหน่วยความจำสำรอง

7.5.2 วิธีการสับเปลี่ยน (Swapping)

หน่วยความจำหลักมีขนาดพื้นที่จำกัด บางครั้งระบบมีพื้นที่หน่วยความจำหลักไม่เพียงพอกับการใช้งาน ทำให้ระบบจำเป็นต้องนำบางโพรเซสออกจากหน่วยความจำหลักก่อน เพื่อให้หน่วยความจำหลักมีพื้นที่ว่างมากเพียงพอที่จะบรรจุโพรเซสอื่น เป็นการสับเปลี่ยน (Swap) ให้โพรเซสอื่นทำงาน ดังแสดงในภาพที่ 7.7



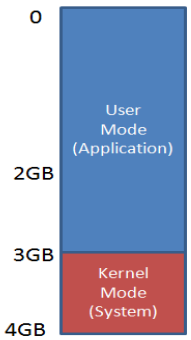
ภาพที่ 7.7 วิธีการสับเปลี่ยนการทำงานของโพรเซส
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.358)

ตัวอย่าง ระบบที่มีการจัดลำดับการทำงานของโพรเซส โดยใช้ลำดับความสำคัญที่กำหนดให้โพรเซสที่มีลำดับความสำคัญสูง สามารถทำงานก่อนโพรเซสที่มีลำดับความสำคัญต่ำกว่า

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

7.6 การจัดสรรหน่วยความจำแบบต่อเนื่อง

ทุกโปรแกรมมีความจำเป็นต้องใช้หน่วยความจำที่มีอยู่ในระบบ จะใช้มากหรือน้อยขึ้นอยู่กับการทำงานและขนาดของโปรแกรม โปรแกรมจะทำงานได้ก็ต่อเมื่อมันได้ถูกนำไปวางไว้ในหน่วยความจำแล้วเท่านั้น การที่โปรแกรมได้เข้าไปใช้หน่วยความจำของระบบเป็นเพราะการจัดสรรหน่วยความจำ (Memory allocation) ของระบบปฏิบัติการ



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

หน่วยความจำหลักจำเป็นจะต้องอำนวยความสะดวกให้กับระบบปฏิบัติการและความหลากหลายของผู้ใช้งานโพรเซส ดังนั้นในแต่ละโพรเซสจึงมีความต้องการใช้พื้นที่ในหน่วยความจำอย่างต่อเนื่อง จึงเป็น

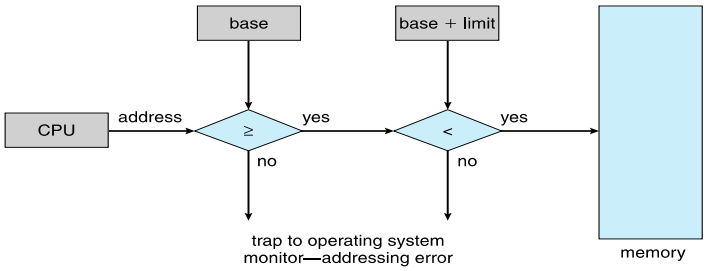
หน้าที่ของระบบปฏิบัติการในการจัดสรรพื้นที่หน่วยความจำในหน่วยความจำให้มีประสิทธิภาพสูงสุด ซึ่งโดยทั่วไปแล้วที่แสดงในภาพที่ 7.8 หน่วยความจำหลักจะถูกแบ่งออกเป็น 2 ส่วน คือ

1. หน่วยความจำระดับบน (High memory) ซึ่งเป็นส่วนที่ใช้ติดต่อกับส่วนของผู้ใช้
2. หน่วยความจำระดับล่าง (Low memory) ซึ่งเป็นส่วนของระบบปฏิบัติการ

ภาพที่ 7.8 จะแสดงให้เห็น หน่วยความจำระดับบนซึ่งเป็นส่วนที่ใช้ติดต่อกับส่วนของผู้ใช้ (User mode) และหน่วยความจำระดับล่างซึ่งเป็นส่วนของระบบปฏิบัติการ (Kemel mode)

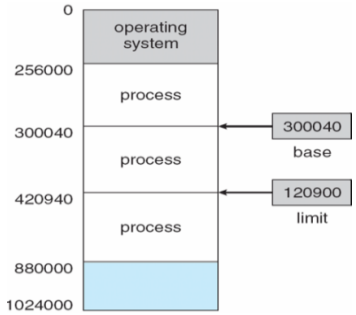
7.6.1 การจัดสรรพื้นที่แบบขนาดคงที่ (Fixed-Size Partition)

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



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

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



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

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

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

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

7.6.2 การแบ่งหน่วยความจำออกเป็นพาร์ติชั่น

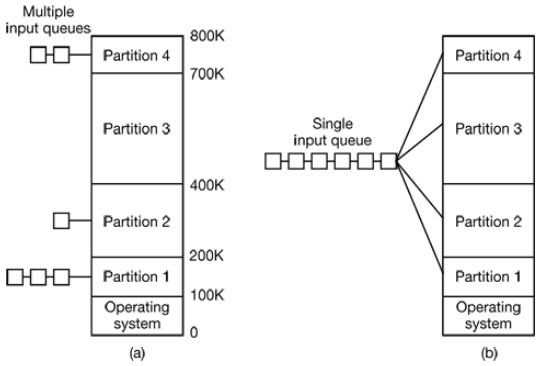
ข้อเสียของการจัดการหน่วยความจำหลักแบบงานเดียว คือ คอมพิวเตอร์สามารถทำงานได้ครั้งละ 1 งาน เพื่อเป็นการเพิ่มประสิทธิภาพการทำงานของซีพียู จึงพัฒนาให้ระบบคอมพิวเตอร์สามารถทำได้หลายงานพร้อมกัน เรียกว่า การทำงานแบบมัลติโปรแกรมมิ่ง ระบบคอมพิวเตอร์ส่วนใหญ่ในปัจจุบันจะอนุญาตให้มีหลายโพรเซสทำงานในเวลาเดียวกันได้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

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

7.6.2.1 การกำหนดขนาดพาร์ติชั่นคงที่ (Static Partition)

วิธีนี้ก่อนจะมีการทำงานใด ๆ หน่วยความจำหลักต้องถูกจัดสรรแบ่งออกเป็นส่วน ๆ ที่มีขนาดแน่นอน ซึ่งการแบ่งส่วนอาจกำหนดโดยโอเปอเรเตอร์ของระบบคอมพิวเตอร์ หรือกำหนดโดย

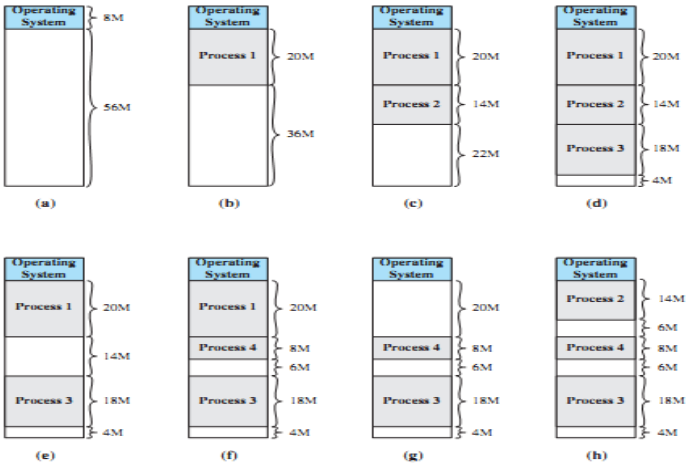
ระบบปฏิบัติการ ถ้ามีโพรเซสที่ต้องการทำงานและขอใช้พื้นที่ของหน่วยความจำหลัก ระบบปฏิบัติการจะต้องตรวจสอบดูขนาดของโพรเซสว่า ควรจะบรรจุลงพาร์ติชั่นใด ซึ่งเป็นไปได้ที่ขนาดของโพรเซสจะไม่พอดีกับขนาดของพาร์ติชั่น กรณีนี้มีผลให้มีพื้นที่บางส่วนของหน่วยความจำหลักที่ต้องสูญเสียไปโดยเปล่าประโยชน์ วิธีนี้เหมาะสำหรับกรณีที่ระบบรู้ขนาดของโพรเซสล่วงหน้า ทั้งนี้เพื่อระบบจะได้กำหนดขนาดของแต่ละส่วนของหน่วยความจำหลักที่เหมาะสมได้



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

7.6.2.2 การกำหนดขนาดของพาร์ติชั่นให้เปลี่ยนแปลงได้ (Dynamic Partition)

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



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

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

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

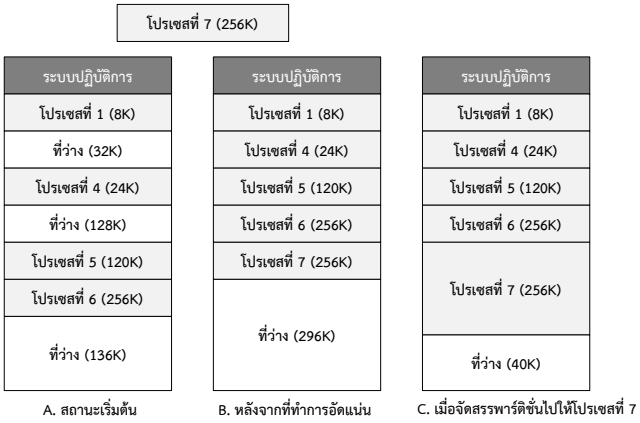
สมมติต่อมาโพรเซสที่อยู่ในหน่วยความจำหลักทั้งสาม ยังไม่สามารถทำงานได้แต่โพรเซสที่ 2 ต้องการที่จะทำงาน แต่พื้นที่ในหน่วยความจำหลักไม่เพียงพอให้โพรเซสที่ 2 ทำงาน ระบบจึงดึงโพรเซส 1 ออกมา ดังภาพที่ 7.12 (g)

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

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

7.6.3 การจัดการหน่วยความจำหลักแบบพาร์ติชั่นและย้ายที่อยู่

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



ภาพที่ 7.13 วิธีการการจัดการหน่วยความจำหลักแบบพาร์ติชั่นและย้ายที่อยู่

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

7.7 ปัญหาการจัดสรรหน่วยเก็บแบบพลวัต

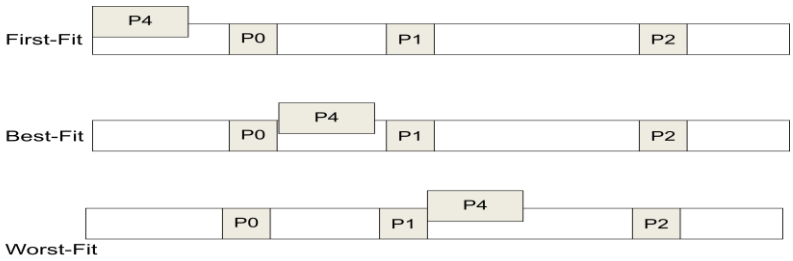
จากข้อจำกัดที่ว่า ทั้งโพรเซสต้องเข้าไปอยู่ในหน่วยความจำหลัก ทำให้มีบางส่วนของโพรเซสที่ไม่จำเป็นต้องใช้งาน แต่จำเป็นต้องถูกนำเข้าไปไว้ในหน่วยความจำหลัก ซึ่งเป็นการสิ้นเปลืองพื้นที่ของหน่วยความจำ ปัญหาข้างต้นเป็นปัญหาในระบบที่มีการจัดสรรพื้นที่แบบยืดหยุ่น (Dynamic storage-allocation problem) ระบบจะพยายามจัดหาพื้นที่ว่างตามขนาดที่ผู้ใช้ร้องขอ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

1. First-Fit คือ เลือกช่องโหว่แรกที่พบและมีขนาดใหญ่เพียงพอกับพื้นที่ที่ต้องการ เป็นวิธีที่ง่ายที่สุดและเสียเวลาน้อยที่สุด

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

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

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



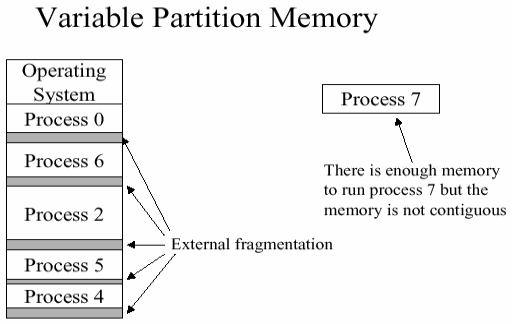
ภาพที่ 7.14 เปรียบเทียบการเลือกพื้นที่ด้วยวิธีต่าง ๆ

7.8 ปัญหาของการจัดการหน่วยความจำ

ปัญหาที่เกิดจากการจัดสรรพื้นที่ที่ตามมาคือ พื้นที่ของหน่วยความจำที่ว่างเป็นช่วง ๆ มีขนาดเล็กไปสำหรับงานที่รอคอยอยู่ประเภทปัญหาของการจัดการหน่วยความจำมีดังนี้

7.8.1 การสูญเปล่าของพื้นที่ย่อยภายนอก (External Fragmentation)

เป็นปัญหาที่เกิดจากการขอพื้นที่หน่วยความจำ เมื่อทำงานในการแจกพื้นที่และตามเก็บพื้นที่ไปสักช่วงเวลาหนึ่ง จะพบว่าพื้นที่ว่างถูกพื้นที่ใช้งานแบ่ง ทำให้พื้นที่ว่างต่อเนื่องยาว ๆ ไม่มีให้ใช้ การที่ระบบมีพื้นที่ว่างพอแต่ไม่สามารถนำมาใช้งานได้ เพราะพื้นที่ว่างถูกพื้นที่ใช้งานแบ่งเป็นพื้นที่ย่อย ๆ มากจนเกินไปจนไม่สามารถนำมาใช้งานได้

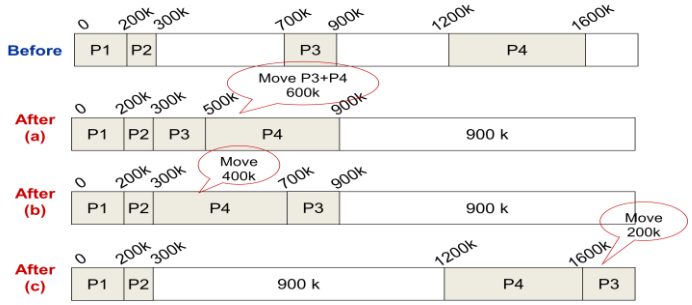


ภาพที่ 7.15 แสดงพื้นที่ว่างเริ่มมีการแตกเป็นส่วน (Fragmentation) เมื่อมีการคืนพื้นที่
ที่มา: B. Thomas Golisano College of Computing and Information Sciences. Retrieved June 27, 2014 from http://www.cs.rit.edu/~hpb/Lectures/99_440/all-inOne-11.html

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

7.8.2 การแก้ปัญหาการสูญเปล่าของพื้นที่ย่อยภายนอก

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



ภาพที่ 7.16 ตัวอย่างการกระชับหน่วยความจำแบบต่าง ๆ

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

7.8.3 การสูญเปล่าของพื้นที่ย่อยภายใน (Internal Fragmentation)

ปัญหาที่เกิดจากการจองพื้นที่หน่วยความจำมากกว่าขนาดข้อมูล พื้นที่ที่จองเกินนั้นจะเกิดเป็นช่องว่างของพื้นที่สูญเปล่าเพราะถูกจองไว้แต่ไม่ได้ใช้ เป็นปัญหาที่เกิดขึ้นภายในพื้นที่ที่จองไว้

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

7.9 การแบ่งพื้นที่เป็นหน้า

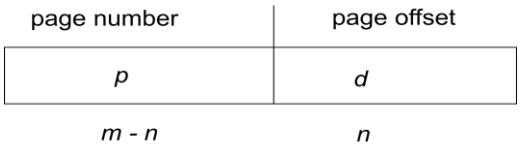
เพจจิ้ง (Paging) คือ การแบ่งหน่วยความจำเป็นหน้า ๆ ขนาดเท่ากัน หลักการของเพจจิ้ง คือ หน่วยความจำจะถูกแบ่งออกเป็นพื้นที่เท่า ๆ กัน เรียงต่อกันไปเรื่อย ๆ ไม่มีช่องว่าง จนหมดพื้นที่หน่วยความจำ การจองหน่วยความจำจะจองเป็นตัวเลขลงตัวเสมอ เช่น ข้อมูลขนาด 3 หน้าครึ่ง จะจองพื้นที่ 4 หน้า เป็นต้น ดังนั้นการรับ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

การแบ่งพื้นที่เป็นหน้า เป็นการจัดสรรพื้นที่ว่างบนหน่วยความจำ โดยทำให้โพรเซสที่อยู่บนหน่วยความจำได้โดยไม่ต้องเรียงต่อเนื่องกันทั้งโพรเซส เป็นการใช้พื้นที่ว่างที่อยู่กระจัดกระจายโดยไม่สูญเปล่า และไม่จำเป็นจะต้องบีบอัดพื้นที่ว่างในหน่วยความจำก่อน แบ่งออกเป็น 2 ประเภท ดังนี้

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

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

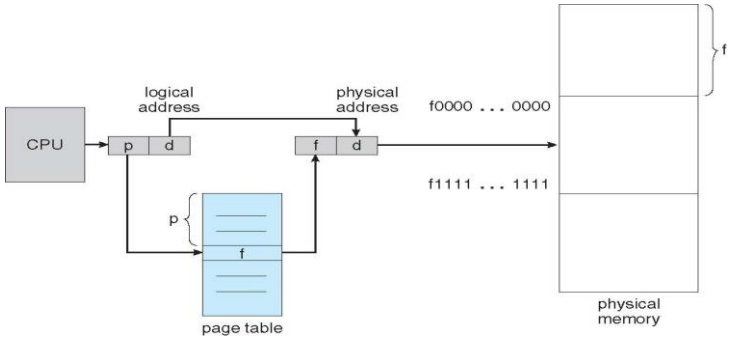
- หมายเลขเพจ (Page number: p) โดยหมายเลขเพจจะใช้ดรรชนี (Index) เพื่อชี้ไปยังตารางเพจ (Page table) ซึ่งภายในบรรจุตำแหน่งเริ่มต้น (Base address) ของแต่ละเพจในหน่วยความจำทางกายภาพ
- ขอบเขตเพจ (Page offset: d) คือ ตำแหน่งจริงในหน่วยความจำ ที่นำมารวมกับตำแหน่งเริ่มต้นที่ได้จากตารางเพจ เพื่อใช้คำนวณหาตำแหน่งของหน่วยความจำทางกายภาพ



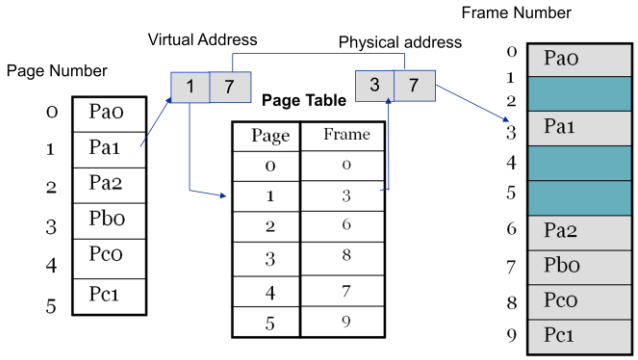
ภาพที่ 7.18 วิธีการอ้างอิงตำแหน่งจริงจากหน่วยความจำ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.369)

7.9.1 การทำงานของ Paging

Paging สามารถใช้จำนวนหน้าที่ไม่ติดกันได้ และในการจองพื้นที่ก็จะไม่เหลือพื้นที่ว่างคั่นไว้ จึงทำให้ไม่เกิดปัญหา External fragmentation



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



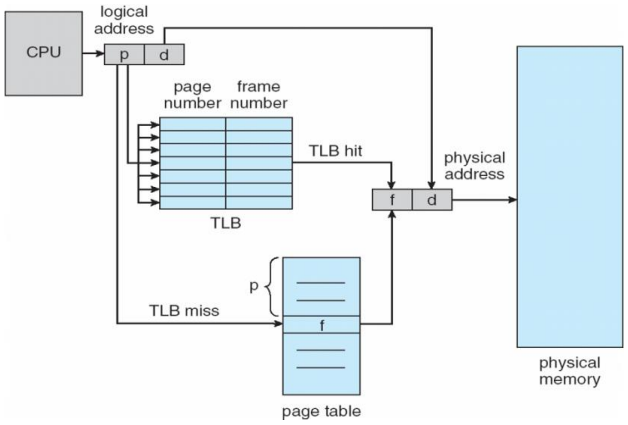
ภาพที่ 7.20 รูปแบบการแบ่งเพจในหน่วยความจำแบบตรรกะและหน่วยความจำแบบกายภาพ (Paging model of logical and physical memory)

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

7.9.2 ฮาร์ดแวร์กับการสนับสนุนการแบ่งหน้า (Hardware Support)

การนำฮาร์ดแวร์กับการสนับสนุนการจัดการตารางเพจทำได้หลายทาง ซึ่งการเข้าถึงตำแหน่งในหน่วยความจำแต่ละครั้ง จะต้องอ่านข้อมูลจากหน่วยความจำทางกายภาพถึงสองครั้ง ครั้งที่หนึ่งจากตารางเพจ และครั้งที่สองจากตำแหน่งข้อมูลทางกายภาพ ทำให้เสียเวลาและการเข้าถึงหน่วยความจำล่าช้า ดังนั้นวิธีมาตรฐานที่ใช้แก้ปัญหาจำเป็นต้องใช้ฮาร์ดแวร์ขนาดเล็กที่มีความเร็ว (Hardware cache) ในการอ่านและจัดเก็บข้อมูลสูง (Fast-lookup) ซึ่งเรียกฮาร์ดแวร์ชนิดนี้ว่า “Translation Look-aside Buffer (TLB)”

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



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

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

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

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

$$\text{Effective Memory-Access Time} = 0.80 \times 120 + 0.20 \times 220$$
$$= 140 \text{ Nanoseconds}$$

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

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

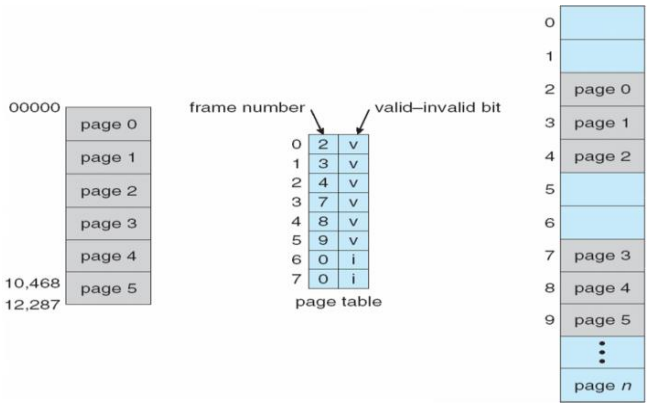
สรุปได้ว่า เวลาจะช้าลงไปถึง 22% ในการเข้าถึงหน่วยความจำ (คือจาก 100 นาโนวินาที เป็น 122 นาโนวินาที)

7.10 การป้องกันหน่วยความจำ

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

1. บิตใช้งานได้ (Valid bit) เป็นบิตสถานะที่บอกว่าข้อมูลในเพจถูกอ่านเข้าสู่หน่วยความจำทางกายภาพแล้ว และสามารถนำไปใช้งานได้ทันที

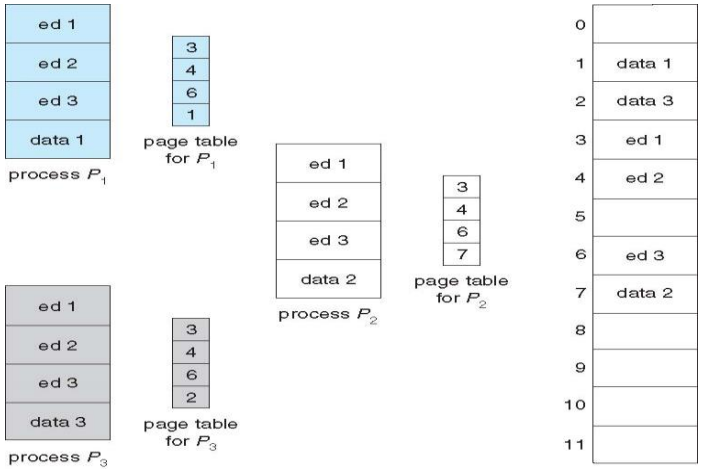
2. บิตใช้งานไม่ได้ (Invalid bit) เป็นบิตสถานะที่บอกว่าข้อมูลในเพจนั้นไม่มีอยู่ในหน่วยความจำทางกายภาพแล้ว และไม่สามารถนำไปใช้งานได้ อาจเกิดจากกรณีที่ระบบปฏิบัติการยังไม่ได้อ่านข้อมูลจากเพจเข้าสู่หน่วยความจำหลัก หรือข้อมูลของเพจที่ต้องการอ่านนั้นถูกสลับออกจากหน่วยความจำแล้ว ซึ่งจะทำให้เกิดข้อผิดพลาดขึ้นที่เรียกว่า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



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

7.11 การใช้เพจร่วมกัน

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



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

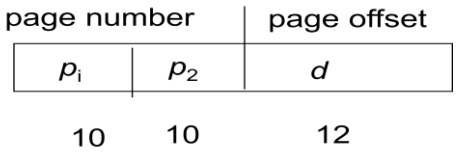
ภาพที่ 7.23 แสดงการใช้เพจร่วมกันระหว่างโพรเซส ซึ่งประกอบไปด้วย 3 โพรเซส คือ P1, P2, และ P3 ต้องการใช้งานโปรแกรม Text Editor ร่วมกันใน Page 0, Page 1 และ Page 3 ที่ถูกจัดเก็บไว้ในหมายเลขเฟรมที่ 3, 4 และ 6 ดังนั้นถ้าระบบปฏิบัติการมีผู้ต้องการใช้งาน 40 คน กำลังใช้งานโปรแกรม Text Editor ซึ่งใช้พื้นที่ 150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

7.11.1 โครงสร้างของตารางเพจ (Memory Protection)

เทคนิคการสร้างตารางเพจแบ่งออกเป็น 3 รูปแบบ ได้แก่

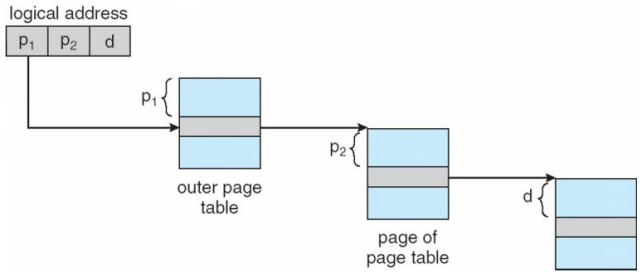
1. โครงสร้างแบบลำดับชั้น (Hierarchical paging) ระบบคอมพิวเตอร์สมัยใหม่จะสนับสนุนตำแหน่งพื้นที่ทางตรรกะ (Logical-address space) ขนาดใหญ่ (232 ถึง 264) ซึ่งโครงสร้างแบบลำดับชั้นเป็นการแบ่งพื้นที่ทางตรรกะออกเป็นตารางเพจหลายตาราง โดยใช้อัลกอริทึมในการแบ่งเพจแบบ 2 ระดับ (Two-level page table) ได้แก่

1. หมายเลขเพจ (page number)
2. ขอบเขตเพจ (page offset)



ภาพที่ 7.24 แสดงการแบ่งเพจแบบ 2 ระดับ หมายเลขเพจและขอบเขตเพจ

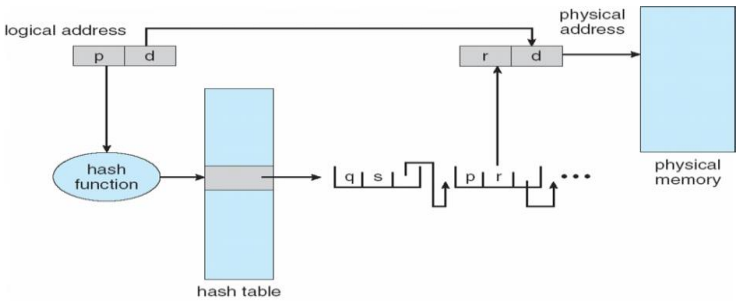
ตัวอย่าง ระบบคอมพิวเตอร์ขนาด 32 บิต ประกอบด้วยเพจขนาด (Page size) 4 KB ตำแหน่งทางตรรกะถูกแบ่งเป็น 2 ส่วน คือ ส่วนแรกขนาด 20 บิต เก็บหมายเลขเพจ (Page number) และส่วนที่สองขนาด 12 บิต เก็บขอบเขตเพจ (Page offset) ซึ่งแสดงโครงสร้างข้อมูลดังนี้



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

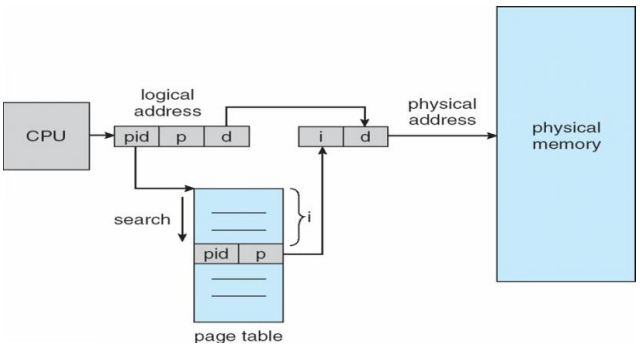
2. โครงสร้างแบบตารางเฮช (Hash page table) ระบบคอมพิวเตอร์ที่จะใช้โครงสร้างแบบนี้ต้องมีตำแหน่งขนาดพื้นที่มากกว่า 32 บิต โดยค่าของเฮชจะอยู่ภายในหมายเลขเพจเสมือน (Virtual-page number) ซึ่งแต่ละหน่วย (Elements) ในตารางเฮชภายในจะมีลิงก์ลิสต์ (Link list) เชื่อมโยงไปยังหน่วยที่อยู่ในพื้นที่เดียวกัน โดยข้อมูลในแต่ละหน่วยจะประกอบไปด้วย

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



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

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



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

7.11.2 Paging และการแก้ไขปัญหา External Fragmentation

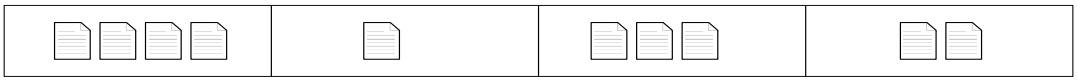
การจองพื้นที่ในเพจจิ้งจะจองเป็น nPage เสมอ โดยค่า n จะเป็นจำนวนเต็ม และ Page คือปริมาณข้อมูลใน 1 หน้ากระดาษ

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

7.12 การแก้ปัญหา Internal Fragmentation

ปัญหา Internal fragmentation คือ การใช้งานพื้นที่ไม่คุ้มค่า จากหลักการของเพจจิ้ง จะมีน้อยครั้งมากที่ขนาดของข้อมูลกับขนาดหน้าจะเท่ากันพอดี ส่วนใหญ่จะเหลื่อม ทำให้ต้องเสียหน้าเพิ่มขึ้นอีกหน้า เกิดช่องว่างไม่ได้ใช้ภายในพื้นที่ที่จอง

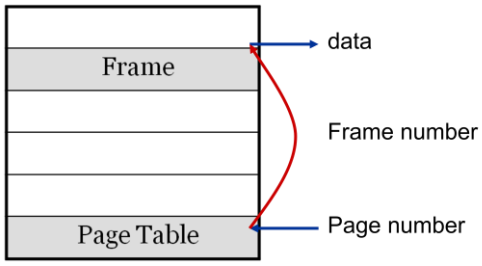
ตัวอย่าง ใน 1 Page เรากำหนดให้เก็บข้อมูลได้ 1024 ไบต์ ตัวข้อมูลมี 9555 ไบต์ เพราะฉะนั้นต้องจอง 10 Page แต่ในความเป็นจริง 10 Page คือพื้นที่เท่ากับ 10240 ไบต์ เกิด Internal Fragmentation ขนาด 685 ไบต์



ภาพที่ 7.28 ปัญหา Internal Fragmentation แสดงการเก็บข้อมูลไม่เต็มหน้า เกิดพื้นที่สูญเปล่า

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

ปัญหาจากตารางเพจถือเป็นหัวใจหลักในการทำงานของเพจจิ้ง การเพิ่มตารางทำให้ระบบต้องมีตัวจัดการตารางที่ดี เพราะถ้าไม่ดีอาจกลายเป็นตารางปัญหาของระบบก็เป็นได้ ในระบบเพจจิ้งจึงมีการนำตารางเพจมาใช้ การทำงานของระบบจะช้าลง เพราะในการทำงานแต่ละครั้งระบบต้องอ่านหน่วยความจำสองหน หนแรกเพื่อหาตำแหน่งเฟรม หนที่สองเพื่อทำงานกับข้อมูลที่ต้องการในเฟรมนั้น



ภาพที่ 7.29 การหาตำแหน่งเฟรม เพื่อทำงานกับข้อมูลที่ต้องการในเฟรมนั้น

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

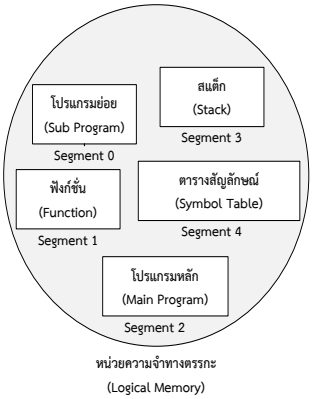
ภาพที่ 7.30 การเปรียบเทียบ (Page Number) มาเทียบในตาราง Page table แบบ Sequential Search

7.13 การแบ่งส่วน

การแบ่งหน่วยความจำเป็นหน้าทำให้เกิดหน่วยความจำทางตรรกะ (ซึ่งผู้ใช้มองเห็น) แตกต่างไปจากลักษณะของหน่วยความจำจริงอย่างหลีกเลี่ยงไม่ได้ โดยมีระบบแปลงตำแหน่งทางตรรกะให้เป็นตำแหน่งจริงทำให้ระบบทำงานได้ถูกต้อง แม้ว่าทั้งสองภาพจะมีความแตกต่างกันก็ตาม

7.13.1 วิธีพื้นฐาน (Basic Method)

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



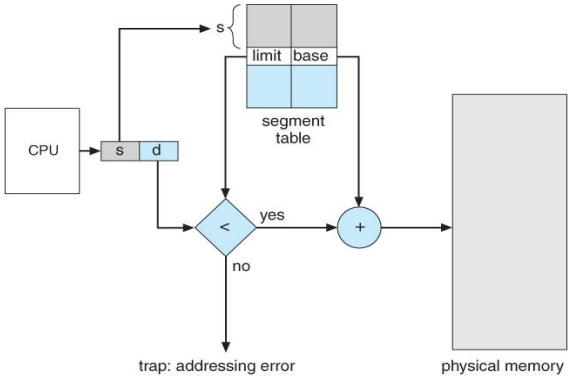
ภาพที่ 7.31 แสดงโครงสร้างแบบแบ่งส่วน (Segmentation)

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

7.13.2 ฮาร์ดแวร์ (Hardware)

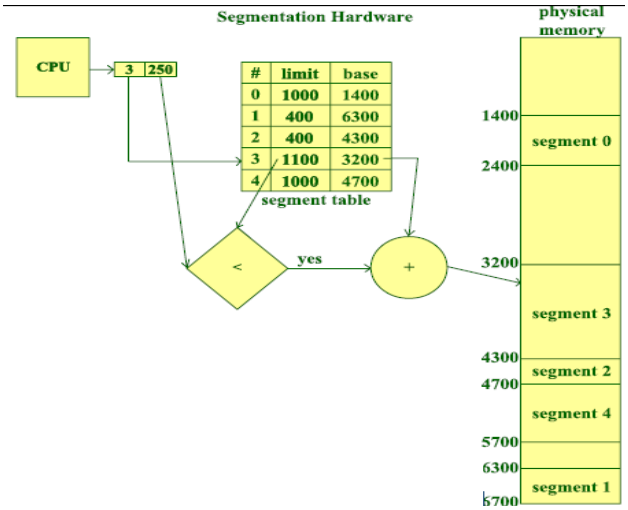
แม้ว่าผู้ใช้สามารถอ้างอิงส่วนต่าง ๆ ของโปรแกรมโดยใช้ตำแหน่งแบบ 2 มิติ แต่หน่วยความจำจริงยังคงเป็นแบบมิติเดียว คือเป็นแถวของคำเรียงต่อกันไป จึงต้องมีวิธีการจับคู่ตำแหน่งทางตรรกะแบบสอง

มิติ ให้เป็นตำแหน่งจริงมิติเดียว โดยใช้ตารางเลขตอน (Segment table) รูปต่อไป แสดงวิธีใช้ ตารางเลขตอน



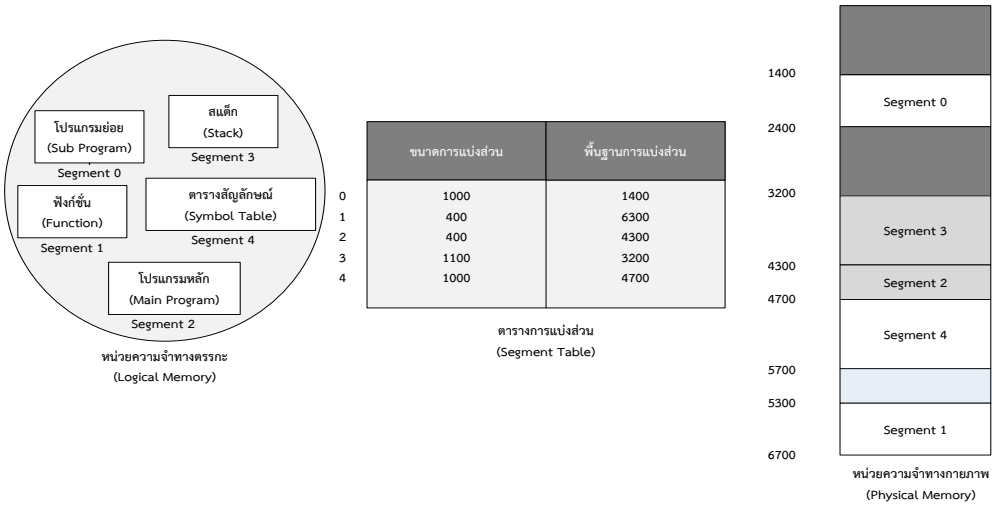
ภาพที่ 7.32 โครงสร้างการแบ่งส่วนหน่วยความจำด้วยฮาร์ดแวร์
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.366)

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



ภาพที่ 7.33 แสดงการทำงานการแบ่งส่วนหน่วยความจำด้วยฮาร์ดแวร์

จากภาพที่ 7.33 อธิบายได้ว่าหมายเลขตอนที่ 3 มีค่า offset เท่ากับ 250 มีค่าไม่เกินค่าขอบเขตของตอนคือ 1100 มีค่าฐานที่ชี้ไปยังตำแหน่งหน่วยความจำจริง 3200 ดังนั้น หมายเลขตอนที่ 3 นี้จะชี้ไปที่ตำแหน่งที่ 3200 + 250 = 3450 ตัวอย่างในรูปถัดไป มี 5 ตอน หมายเลข 0 จนถึง 4 แต่ละตอนเก็บอยู่ในหน่วยความจำทางกายภาพ ดังแสดงในรูป 7.34



ภาพที่ 7.34 แสดงโครงสร้างตารางการแบ่งส่วน (Segmentation Table)

ตัวอย่าง 1. จากภาพที่ 7.34 กำหนดให้มีการแบ่งส่วนออกเป็น 5 ส่วน มีหมายเลขตั้งแต่ 0 ถึง 4 โดยแต่ละส่วน ถูกจัดเก็บอยู่ในหน่วยความจำทางกายภาพ ภายในตารางการแบ่งส่วนมีข้อมูลอยู่ภายใน โดยกำหนดตำแหน่งเริ่มต้นเรียกว่า ตำแหน่งฐาน (Base) และขนาดของการแบ่งส่วนเรียกว่า Limit อธิบายได้ดังนี้

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

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

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

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

7.13.3 การสร้างตารางเลขตอน (Implementation of Segmentation Tables)

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

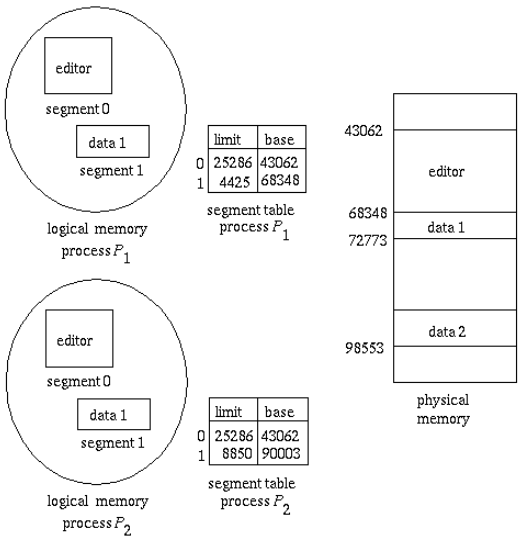
ถ้าโปรแกรมหนึ่งแบ่งเป็นตอน ๆ จำนวนมาก เราก็ไม่สามารถเก็บตารางเลขตอนใน รีจีสเตอร์พิเศษได้ ต้องเก็บตารางเลขตอน(ขนาดใหญ่)ไว้ในหน่วยความจำหลักแทน แล้วใช้รีจีสเตอร์พิเศษเก็บตัวชี้ไปยังตารางเลขตอนอีกที (Segment Table Base Register STBR) เนื่องจากโปรแกรมต่าง ๆ มีจำนวนตอนไม่เท่ากัน เราจึงอาจเก็บค่าจำนวนตอนไว้ในรีจีสเตอร์พิเศษเรียกว่า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

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

7.13.4 การป้องกันและการใช้ตอนร่วมกัน (Protection and Sharing)

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

ข้อดีอีกข้อหนึ่งก็คือ สามารถใช้ข้อมูลหรือโปรแกรมร่วมกันได้สะดวก กระบวนการแต่ละตัวจะมีตารางเลขตอนของตนเอง (เก็บอยู่ในตารางข้อมูลเฉพาะของกระบวนการ) กระบวนการหลายตัวอาจใช้ตอนร่วมกันได้ โดยให้ตัวชี้เลขตอนชี้ไปยังที่เดียวกันในหน่วยความจำจริง ดังภาพที่ 7.35



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

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

7.13.5 การสูญเปล่าพื้นที่ย่อย (Fragmentation)

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

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

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

7.14 สรุป

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

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

กระบวนการจัดการหน่วยความจำประกอบด้วย การย้ายตำแหน่ง ระบบปฏิบัติการในปัจจุบันยอมให้โปรแกรมทำงานพร้อมกันได้หลายงานแบบ Multiprogramming ซึ่งโพรเซสต่าง ๆ เข้าใช้งานหน่วยความจำร่วมกัน จึงต้องมีการสลับโปรแกรมให้เข้าออกหน่วยความจำได้ รวมถึงการเปลี่ยนแปลงค่าตำแหน่งในหน่วยความจำที่อ้างอิงถึงในโปรแกรมให้ถูกต้องตามตำแหน่งจริง ในการป้องกันพื้นที่ระบบปฏิบัติการสามารถป้องกันโพรเซสจากการถูกรบกวนทั้งทางตรงและทางอ้อม

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

แบบฝึกหัดท้ายบทที่ 7

1. จงอธิบายการทำงานระบบโปรแกรมเดียวมีวิธีการจัดการหน่วยความจำอย่างไร

2. จงอธิบาย Absolute Address กับ Relative Address แตกต่างกันอย่างไร

3. จงอธิบายMemory Management Unit : MMU มีหน้าที่อย่างไร

4. จงอธิบาย Static Partition กับ Dynamic Partition มีข้อดีข้อเสียแตกต่างกันอย่างไร

5. จงอธิบายถึงสาเหตุของการเกิด External Fragmentation เกิดจากอะไรและมีวิธีแก้ไขการเกิดได้อย่างไร

6. จงอธิบายสาเหตุของการเกิด Internal Fragmentation เกิดจากอะไรและมีวิธีแก้ไขการเกิดได้อย่างไร

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

8. จงอธิบายว่า อะไรคือวัตถุประสงค์หลักในการใช้ Paging และ Page Tables

9. พิจารณา Segment table ต่อไปนี้

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

จงตอบคำถามว่าอะไรคือ Physical Addresses ของ Logical Addresses ต่อไปนี้

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

บรรณานุกรมประจำบทที่ 7

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

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

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

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

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

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

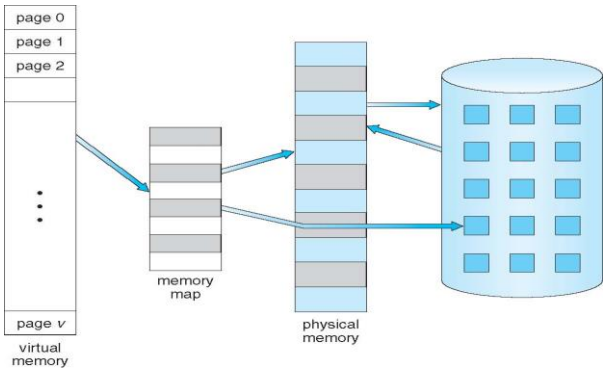
บทที่ 8
หน่วยความจำเสมือน

8.1 แนวคิดหน่วยความจำเสมือน

หน่วยความจำเสมือนเป็นวิธีหนึ่งที่สามารถทำให้โพรเซสทำงานได้ ถึงแม้ว่าโพรเซสนั้นจะไม่ได้อยู่ในหน่วยความจำหลักทั้งหมด โดยระบบปฏิบัติการจะทำหน้าที่เก็บบางส่วนของโปรแกรมที่กำลังทำงานไว้ในหน่วยความจำหลัก และเก็บส่วนที่เหลือไว้ในฮาร์ดดิสก์ ตัวอย่างเช่น โปรแกรมที่ต้องการหน่วยความจำ 100 MB สามารถทำงานบนเครื่องที่มีหน่วยความจำ 64 MB ได้โดยการเลือกเอาบางส่วนของโปรแกรมขนาด 64 MB เข้ามาทำงานในหน่วยความจำหลัก และบางส่วนเก็บไว้ในฮาร์ดดิสก์และทำการสลับเปลี่ยนไปมาเมื่อมีบางส่วนของโปรแกรมต้องการทำงาน ดังแสดงในภาพที่ 8.1

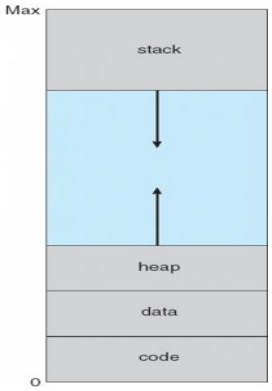
ข้อดีของวิธีนี้ คือ โปรแกรมของผู้ใช้สามารถมีขนาดใหญ่กว่าหน่วยความจำจริงได้ เพราะวิธีนี้จะทำการแยกส่วนของหน่วยความจำเชิงตรรกะ (Logical memory) ของผู้ใช้ออกจากหน่วยความจำเชิงกายภาพ (Physical memory) มีเพียงส่วนของโปรแกรมที่ต้องการอยู่ในหน่วยความจำเพื่อดำเนินการเท่านั้น พื้นที่ตำแหน่งเชิงตรรกะ (Logical 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 ในมุมมองนี้โพรเซสจะเริ่มต้นที่แหน่งที่อยู่ที่แน่นอน คือ ตำแหน่งที่ 0 และอยู่ในหน่วยความจำที่ติดกัน ดังแสดงในภาพที่ 8.2



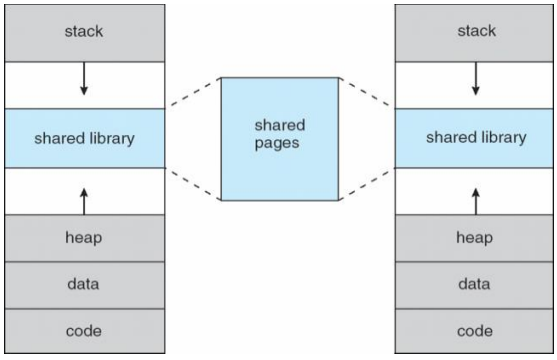
ภาพที่ 8.2 พื้นที่ว่างในหน่วยความจำเสมือน
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.399)

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



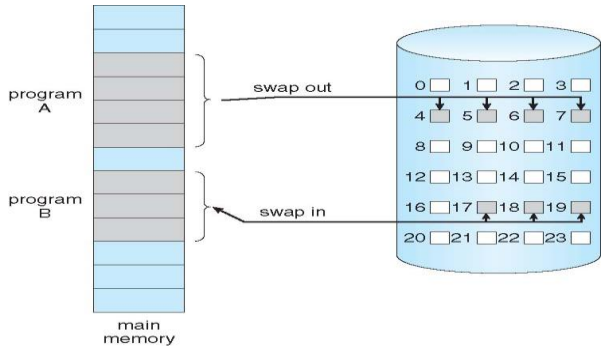
ภาพที่ 8.3 แชร์ Library ในหน่วยความจำเสมือน
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.400)

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) เราจะใช้ตัวสับเปลี่ยน ในการนำเฉพาะเพจที่ต้องการใช้เข้ามาไว้ในหน่วยความจำ มีข้อดีคือ

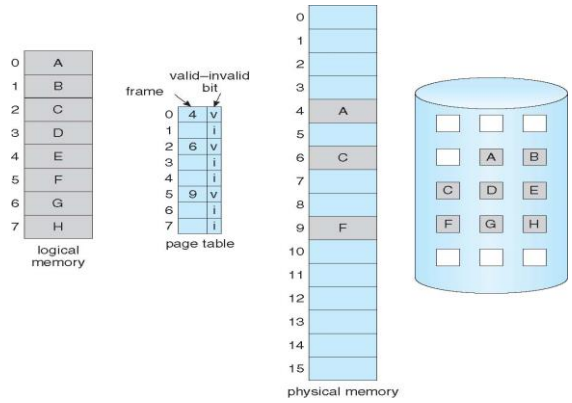
1. ลดการใช้ (Less I/O needed)
2. ลดการใช้หน่วยความจำ (Less memory needed)
3. โต้ตอบได้รวดเร็วกว่า (Faster response)
4. รองรับผู้ใช้ได้มากกว่า (More users)



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)

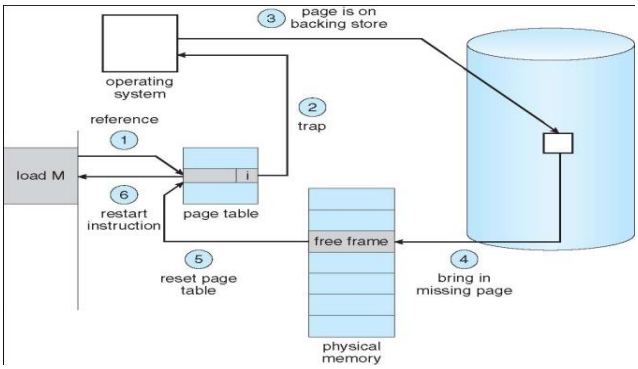
วิธีการนี้ต้องมีอุปกรณ์ทางฮาร์ดแวร์ คือ เพิ่มบิตสถานะ (Valid-invalid bit) อีกหนึ่งบิตต่อช่องในตารางเลขหน้า ถ้าสถานะมีค่าเป็นใช้ได้ (Valid) แสดงว่าหน้านั้นอยู่ในหน่วยความจำหลัก แต่ถ้าบิตสถานะมีค่าเป็นใช้ไม่ได้ (Invalid) แสดงว่าหน้านั้นอยู่ในจานบันทึก การใช้ตารางเลขหน้าจะทำงานเหมือนในระบบแบ่งหน้า คือ ตารางเลขหน้าจะต้องถูกนำลงในหน่วยความจำหลักพร้อมกับตัวโพรเซส 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) เมื่อเกิดการผิดหน้าขึ้นโพรเซสที่ถูกเรียกเพจหน้านั้นจะถูกยกเลิกไป 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” โพรเซสจึงสามารถทำงานต่อไปได้ เสมือนว่าหน้าที่ต้องการอยู่ในหน่วยความจำหลักตลอดเวลา

การที่มีหน่วยความจำเสมือนทำให้ระบบสามารถรันโปรแกรมต่าง ๆ ได้มากขึ้น ระบบมีพื้นที่หน่วยความจำรวมเพิ่มขึ้น ทำงานได้หลากหลายพร้อม ๆ กันได้มากขึ้น แต่การทำเช่นนี้มีข้อเสีย คือ เมื่อระบบสามารถรองรับเพจได้มากขึ้น ทำให้จำนวนข้อมูลในตารางเพจจะมีมากขึ้นตาม ถ้ามีข้อมูล 1 เพจ ต้องใช้ข้อมูล 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

- รายงานข้อผิดพลาดไปยังระบบปฏิบัติการ
- ใช้เก็บค่าต่าง ๆ ของรีจีสเตอร์ และสถานะของโพรเซส
- ตรวจดูว่าสัญญาณขัดจังหวะนั้นเป็นการผิดหน้า
- ตรวจดูว่าหน้าที่ต้องการถูกต้อง อยู่ในขอบเขตที่มีสิทธิใช้ และหาค่าตำแหน่งของหน้านี้จากการบันทึก
- ทำการร้องขอไปยังจานบันทึกให้อ่านหน้านั้นเข้ามายังพื้นที่ว่าง (Free frame)
- เข้าไปรอในแถวอุปกรณ์จนกระทั่งถึงคราวของเขา
- อุปกรณ์และจานบันทึกจะหมุนและกวาดไปยังตำแหน่งที่ต้องการ
- มีการโอนถ่ายข้อมูลจากอุปกรณ์ไปยังพื้นที่ว่าง
- ขณะที่รอคอย ตัวจัดตารางการทำงานอาจจัดให้โพรเซสอื่นทำงานไปก่อน

- สัญญาณขัดจังหวะอินเตอร์รัพจากจานบันทึกว่าอ่านเสร็จแล้ว
- จัดเก็บค่าของรีจีสเตอร์และสถานะของโพรเซสอื่นที่กำลังทำงานอยู่
- ตรวจดูสัญญาณจากจานบันทึก
- แก้ไขค่าในตารางเลขหน้าและตารางอื่น ๆ ที่เกี่ยวข้อง เพื่อแสดงว่าหน้าที่ที่ต้องการอยู่ในหน่วยความจำหลักแล้ว
- เข้าไปรอคอยอยู่ในแถวพร้อม เพื่อรอเข้าใช้หน่วยประมวลผล
- เมื่อถึงคราวบรรจุค่าต่าง ๆ ที่เก็บไว้ลงในรีจิสเตอร์และตารางเลขหน้าของโพรเซสแล้ว เริ่มทำคำสั่งเดิมที่หยุดไว้

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

ตัวอย่าง Demand Paging

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

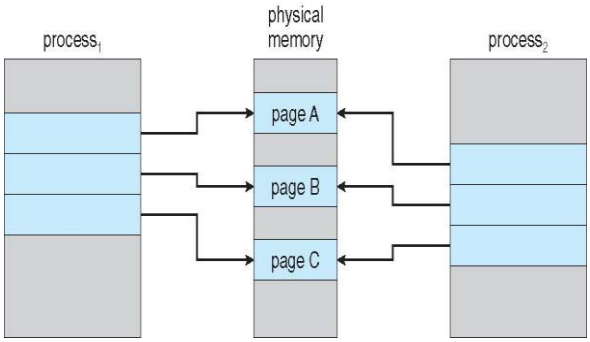
$$\begin{aligned} \text{Effective access time} &= (1 - p) \times (200) + p\ (8\ \text{มิลลิวินาที}) \\ &= (1 - p) \times 200 + p \times 8{,}000{,}000 \\ &= 200 + 7{,}999{,}800 \times p.\ \text{นาโนวินาที} \end{aligned}$$

8.3 เทคนิคการบันทึกข้อมูลด้วยการคัดลอกข้อมูล

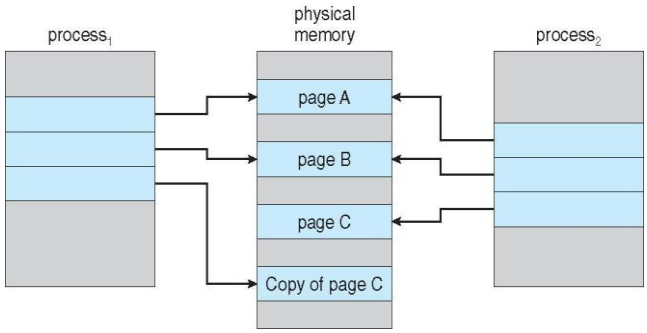
ในหัวข้อนี้ เราจะอธิบายโพรเซสที่เริ่มต้นทำงานอย่างรวดเร็วได้อย่างไร โดยในเพจ ประกอบด้วยคำสั่งแรก อย่างไรก็ตามโพรเซสใช้ฟังก์ชั่น fork() (เรียกผ่าน System call) ในการเริ่มต้นแบบทางอ้อม ระบบต้องการวิธีการ Demand Paging โดยใช้วิธีที่เหมือนกับ Page Sharing เทคนิคนี้จัดการสำหรับการสร้างโพรเซสแบบรวดเร็วและเล็ก จำนวนเพจใหม่ซึ่งจำเป็นต้องจัดสรรให้กับโพรเซสที่เพิ่งถูกสร้างขึ้นใหม่ การเรียกฟังก์ชั่น fork() กลับมาสร้างโพรเซสลูก (A child process) เหมือนกับของ โพรเซสแม่ 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



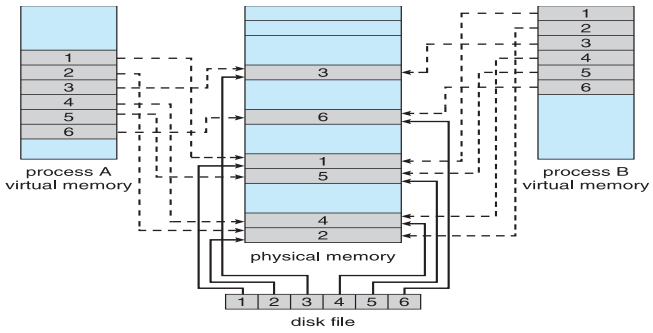
ภาพที่ 8.7 แสดงก่อน โพรเซส 1 จะเปลี่ยนแปลง Page C
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.408)



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 อัลกอริที่มการสับเปลี่ยนหน้า

นโยบายการสับเปลี่ยนหน้าเป็นการเลือกหน้าเก่าที่อยู่ในหน่วยความจำออกเพื่อจะแทนที่ด้วยหน้าใหม่ที่จะนำเข้ามา ซึ่งจะมีสิ่งที่ต้องพิจารณาหลายกรณีด้วยกัน เช่น

1. มีจำนวนเฟรมเท่าไหร่ที่สามารถให้กับโพรเซสแต่ละตัวได้

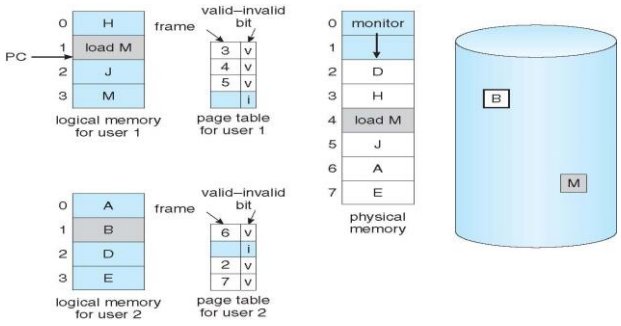
2. กลุ่มของหน้าที่ถูกพิจารณาให้สับเปลี่ยนหน้านั้นควรจะถูกจำกัดหรือไม่ เพื่อป้องกันไม่ให้โพรเซสที่ทำให้เกิดการผิดหน้าทำงาน

3. ในกลุ่มของหน้าที่เราพิจารณา เราจะเลือกหน้าใดที่จะนำไปสับเปลี่ยน

4. ถ้าเราให้หน่วยความจำแก่โพรเซสใด ๆ น้อยเท่าไหร่ ก็จะทำให้มีจำนวนโพรเซสเข้ามาใช้งานหน่วยความจำมากขึ้นเท่านั้น ซึ่งจะทำให้ความน่าจะเป็นในการที่จะค้นพบโพรเซสอย่างน้อยหนึ่งโพรเซสที่พร้อมจะทำงานมากขึ้น ทำให้เราเสียเวลาในการสลับหน้าน้อยลง

5. ถ้ามีจำนวนหน้าของโพรเซสหนึ่ง ๆ ที่สามารถอยู่ในหน่วยความจำหลักน้อย (จำนวนเฟรมน้อย) ก็จะทำให้เกิดอัตราการเกิดการผิดหน้าสูงมากขึ้น

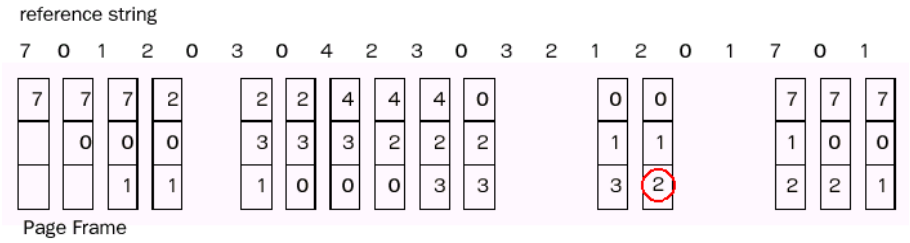
6. นอกเหนือจากการกำหนดขนาดของหน้าคงที่แล้ว การให้หน่วยความจำเพิ่มกับโพรเซสใดโพรเซสหนึ่งจะไม่มีผลกระทบต่ออัตราการผิดหน้ามากนัก



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

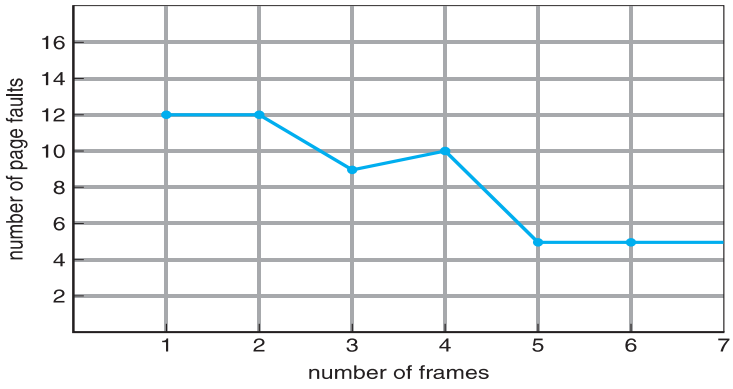


ภาพที่ 8.11 วิธีสับเปลี่ยนแบบมาก่อน-ออกก่อน (First-In-First-Out Algorithms : FIFO)

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



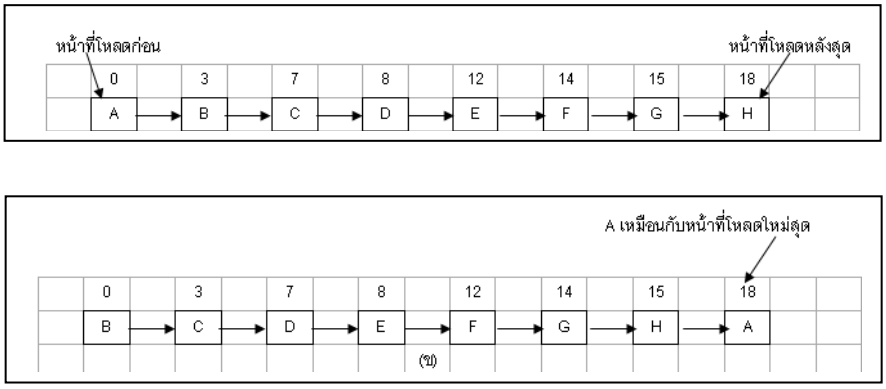
ภาพที่ 8.12 กราฟแสดงปรากฏการณ์เบลาดี้ ในวิธีสับเปลี่ยนแบบมาก่อน-ออกก่อน

เมื่อทดลองหาจำนวนครั้งของการผิดหน้าเทียบกับจำนวนเฟรมที่มี จะได้กราฟดังในภาพที่ 8.12 จะสังเกตได้ว่า เมื่อหน่วยความจำมีเฟรมทั้งหมด 4 เฟรม จะเกิดการผิดหน้าทั้งหมด 10 ครั้ง ซึ่งมากกว่าเมื่อมีทั้งหมด 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 หน้าที่อยู่มานานที่สุด คือ 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

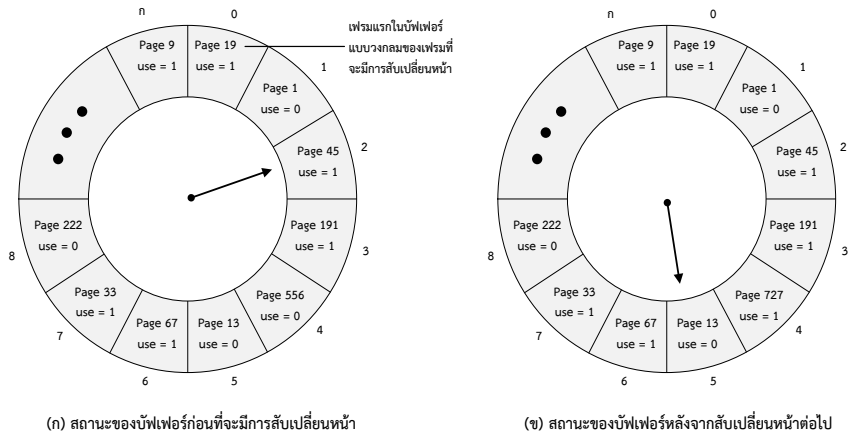


ภาพที่ 8.13 วิธีการสับเปลี่ยนหน้าแบบให้โอกาสครั้งที่สอง

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)

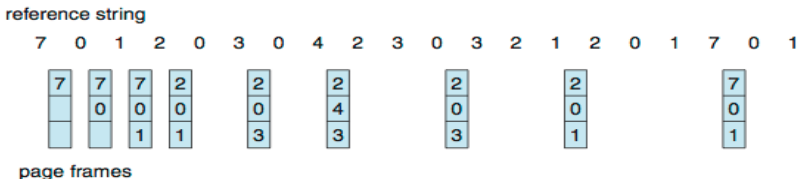
ใช้วิธีเรียงเฟรมทุกเฟรมเป็นรูปวงกลมเหมือนนาฬิกา ภาพที่ 8.14 และมีเข็มนาฬิกาชี้ไปที่หน้าที่เก่าที่สุด เมื่อเกิดการผิดหน้า หน้าที่มีเข็มนาฬิกาชี้อยู่จะถูกตรวจสอบ ถ้าบิต R มีค่าเป็น 0 หน้านั้นจะถูกสับเปลี่ยนออกไป และหน้าใหม่ก็จะถูกใส่เข้ามาในตำแหน่งเดิม พร้อมกันนั้นเข็มนาฬิกาจะทำการเลื่อนไปด้านหน้า 1 ตำแหน่ง 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



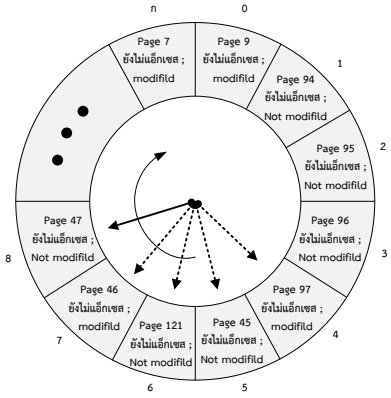
ภาพที่ 8.15 วิธีสับเปลี่ยนแบบที่ดีที่สุด

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

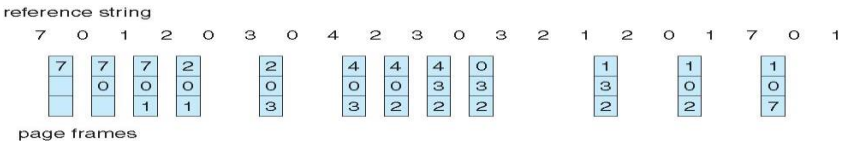
วิธี NRU นี้จะสุ่มเอาหน้าออกจากกลุ่มลำดับต่ำสุด (เช่น กลุ่มที่ 1 มีลำดับต่ำกว่ากลุ่มที่ 2 และ 3 ตามลำดับ) ที่มีหน้าของโพรเซสอยู่ นอกจากนั้นวิธีการนี้ยังดูเหมือนว่าจะพยายามที่จะนำเอาหน้าที่ถูกเปลี่ยนแปลงแต่ไม่ได้ถูกเรียกใช้งานหรืออ้างอิงถึงออก ทุกครั้งที่มีการครบวงรอบนาฬิกา (ประมาณ 20 มิลลิวินาที) มากกว่าหน้าที่ไม่มีการเปลี่ยนแปลงแต่ถูกเรียกใช้งานบ่อย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

8.5.6 การสับเปลี่ยนแบบใช้งานน้อยที่สุด-ออกก่อน (Least Recently Used : LRU)

วิธีนี้จะเป็นการใช้ข้อมูลในอดีตที่ผ่านมาประมาณการอนาคตอันใกล้ โดยอาจจะสับเปลี่ยนหน้าที่ไม่ได้ถูกเรียกใช้งานเป็นเวลานานที่สุดออก ซึ่งเรียกว่าเป็นแบบใช้งานน้อยที่สุดออกก่อน (LRU) วิธีแบบนี้

จะบันทึกเวลาที่แต่ละหน้าถูกอ้างอิงครั้งล่าสุดไว้ เมื่อต้องการเลือกหน้าเพื่อสับเปลี่ยนออก ก็จะเลือกหน้าที่ไม่ได้ใช้มาเป็นเวลานานสุด (หน้าที่มีตัวเลขเวลาน้อยที่สุด)

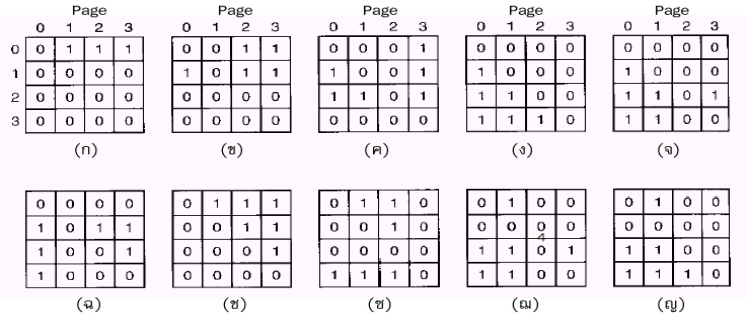
จะเห็นได้ว่าวิธีการนี้ได้มองย้อนไปในเวลาอดีต แทนที่จะมองไปในอนาคต แม้ว่า LRU น่าจะใช้งานได้ดีที่สุดในทางทฤษฎี แต่ก็ไม่ง่ายในทางปฏิบัติ เนื่องจากการใช้ LRU นั้น ระบบจะต้องสร้างลิงค์ลิสต์ของทุก ๆ หน้าในหน่วยความจำที่มีหน้าที่ใช้งานน้อยที่สุดอยู่ที่หัวแถว และหน้าที่ใช้งานบ่อยที่สุดอยู่ท้ายสุด และความยากของวิธีนี้ก็คือลิสต์จะต้องทำการปรับเปลี่ยนทุกครั้งที่มีการอ้างอิงถึง 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 เราจะได้เมตริกซ์ ดังภาพที่ 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

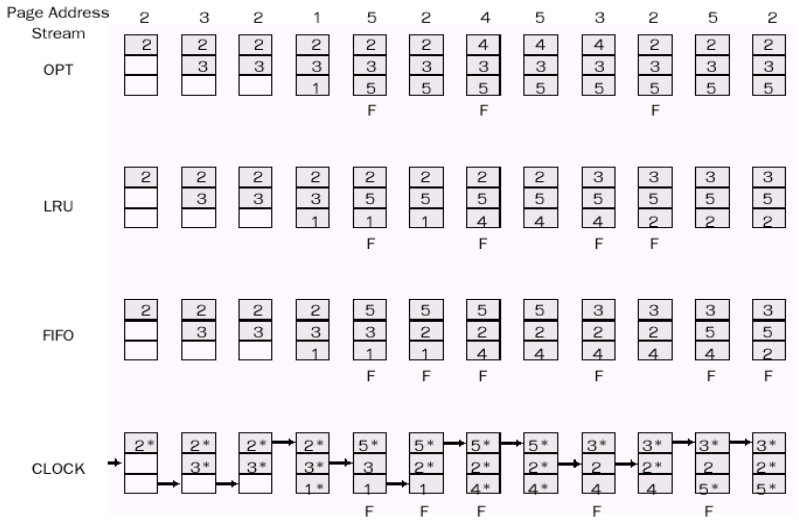
Frame 0 = (0 1 0 0) = 4 (ฐานสิบ)

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) การจัดการดังกล่าวทำให้ซีพียูสามารถใช้ข้อมูลที่อยู่ในหน่วยความจำหลัก หรืออยู่ในที่ใดก็ได้ (เช่น ในฮาร์ดดิสก์) ที่สามารถทำการสลับโพรเซสเหล่านั้นให้เข้ามาทำงานได้ แนวความคิดของหน่วยความจำเสมือนนี้ จะอนุญาตให้โพรเซสถูกแยกส่วนออกเป็นชิ้นเล็ก ๆ ได้ และชิ้นส่วนเล็ก ๆ ของโพรเซส ไม่จำเป็นที่จะต้องอยู่ติดกันในหน่วยความจำหลักขณะที่ทำงาน และยิ่งกว่านั้นคือ โพรเซสยังสามารถทำงานได้ โดยไม่จำเป็นต้องให้ชิ้นส่วนเล็ก ๆ ทั้งหมดของโพรเซสนั้นอยู่ในหน่วยความจำหลักด้วย

แบบฝึกหัดท้ายบทที่ 8

1. จงอธิบายแนวคิดของหน่วยความจำเสมือนคืออะไรมีข้อดีอย่างไร
2. เมื่อเกิดการผิดหน้า (Page fault) จงอธิบายว่าระบบปฏิบัติกานมีวิธีการดำเนินการอย่างไร
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

บรรณานุกรมประจำบทที่ 8

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/

บทที่ 9
การจัดการแฟ้มข้อมูล

หน้าที่สำคัญของระบบปฏิบัติการ คือ การจัดการแฟ้มข้อมูล (Files management) ซึ่งเป็นการช่วยอำนวยความสะดวกแก่ผู้ใช้งานในการเรียกใช้แฟ้มข้อมูลต่าง ๆ ผู้ใช้งานสามารถเรียกใช้คำสั่งเพื่อดำเนินการต่าง ๆ กับแฟ้มข้อมูลได้ เช่น การสร้างแฟ้มข้อมูล การเปิดแฟ้มข้อมูล หรือการลบแฟ้มข้อมูล การทำงานในระบบคอมพิวเตอร์ทั้งหมดจำเป็นต้องมีการเก็บและนำข้อมูลไปใช้งาน ขณะที่โพรเซสกำลังทำงานข้อมูลจะเก็บไว้ในหน่วยความจำ ถ้าเครื่องคอมพิวเตอร์ดับไม่ว่าด้วยสาเหตุใดก็ตาม ข้อมูลทั้งหมดจะสูญหายไปดังนั้นจึงจำเป็นต้องจัดเก็บข้อมูลเหล่านี้ไว้ในหน่วยจัดเก็บข้อมูลสำรอง ซึ่งอาจจะเป็นแผ่นดิสก์เก็ต ฮาร์ดดิสก์ หรืออุปกรณ์อื่น ๆ ในการจัดเก็บข้อมูลเหล่านี้มีจุดประสงค์เพื่อนำมาใช้งานต่อไป จึงจำเป็นต้องมีการกำหนดชื่อเพื่อแทนกลุ่มข้อมูล ซึ่งเราเรียกว่า แฟ้มข้อมูล

9.1 แนวคิดเกี่ยวกับแฟ้มข้อมูล

แฟ้มข้อมูล คือ ชื่อของสารสนเทศที่สัมพันธ์กัน ซึ่งถูกบันทึกไว้ในหน่วยเก็บข้อมูลสำรอง (Secondary storage) ในมุมมองของผู้ใช้แฟ้มข้อมูลคือการจัดสรรที่เล็กที่สุดของหน่วยเก็บข้อมูลสำรอง ซึ่งข้อมูลนี้ไม่สามารถเขียนไปยังหน่วยความจำหลักได้ เว้นแต่จะถูกจัดเก็บภายในแฟ้มของข้อมูล ปกตินั้นแฟ้มข้อมูลจะถูกแสดงโดยทางโปรแกรมซึ่งจะแสดงข้อมูลของแฟ้มข้อมูลด้วย ข้อมูลที่อยู่ในแฟ้มข้อมูลจะถูกแสดงออกมาเป็นตัวเลขตามตัวอักษรที่อยู่ข้างใน หรือเป็นเลขฐานสอง แฟ้มข้อมูลนั้นอาจจะเป็นอิสระต่อกัน เช่น แฟ้มข้อมูลของข้อความหรืออาจมีการจัดรูปแบบแฟ้มข้อมูลที่มีความเข้มงวด แฟ้มข้อมูลถูกกำหนดเป็นโครงสร้างตามชนิดของข้อมูลดังนี้

1. Text File คือ ลำดับของตัวอักษรที่เรียงกันในบรรทัด (หรือหน้า)

2. Source File คือ ลำดับของโปรแกรมย่อย (Subroutine) และฟังก์ชัน (อาจเป็นการประกาศค่าตามประโยค)

3. Object File คือ ลำดับของไบต์ ที่จัดเรียงในบล็อคที่ตัวเชื่อมโยง (Linker) ของระบบเข้า

4. Executable File คือ ลำดับของส่วนของรหัสโปรแกรมซึ่งตัว Load โปรแกรม (Loader) นำเข้ามายังหน่วยความจำและสั่งให้ทำงาน (Execute)

9.1.1 คุณลักษณะของแฟ้มข้อมูล (File Attributes)

จุดประสงค์ในการออกแบบระบบปฏิบัติการอย่างหนึ่งก็คือ ต้องการที่จะให้ผู้ใช้เป็นอิสระจากอุปกรณ์ใด ๆ (Device independent) ดังนั้น ในการเข้าถึงแฟ้มข้อมูลใด ๆ จะต้องมีรูปแบบเดียวกัน นอกจากนั้นวิธีการในการเข้าถึงแฟ้มข้อมูล ไม่จำเป็นต้องกำหนดรายละเอียดหรือหมายเลขตำแหน่งที่เก็บให้ยุ่งยากวุ่นวาย เพียงแค่ระบุชื่อและนามสกุลของแฟ้มข้อมูลให้ถูกต้องก็เพียงพอแล้ว คุณลักษณะของแฟ้มข้อมูลที่แตกต่างกันขึ้นอยู่กับระบบปฏิบัติการ แต่โดยปกติจะประกอบด้วยสิ่งเหล่านี้

- ชื่อ (Name) ชื่อแฟ้มข้อมูลที่เป็นสัญลักษณ์เป็นเพียงข้อมูลที่เก็บไว้ในรูปแบบที่อ่านได้โดยมนุษย์
- ตัวระบุ (Identifier) แท็กที่ไม่ซ้ำกันนี้มักจะเป็นหมายเลขตัวระบุแฟ้มข้อมูลภายในระบบแฟ้ม
- ประเภท (Type) ข้อมูลนี้จำเป็นสำหรับระบบที่สนับสนุนประเภทต่าง ๆ ของแฟ้มข้อมูล
- ตำแหน่ง (Location) ข้อมูลนี้เป็นตัวชี้ไปยังอุปกรณ์และตำแหน่งของแฟ้มที่อยู่ในฮาร์ดดิสก์
- ขนาด (Size) จะบอกขนาดปัจจุบันของแฟ้มข้อมูล (หน่วยเป็นไบต์) ซึ่งขนาดสูงสุดจะถูกกำหนดเอาไว้
- การป้องกัน (Protection) จะใช้ควบคุมสิทธิ์ในการเข้าถึงแฟ้มข้อมูลต่าง ๆ เช่น การอ่านเขียนแฟ้มข้อมูล
- วันเวลาของผู้ใช้ (Time, Date, and User Identification) ข้อมูลนี้จะถูกเก็บไว้เมื่อมีการแก้ไขแฟ้มข้อมูลนี้ในครั้งล่าสุด ซึ่งมีประโยชน์สำหรับป้องกันความปลอดภัย

ข้อมูลเกี่ยวกับแฟ้มข้อมูลทั้งหมดนั้นถูกเก็บอยู่ในไดเรกทอรี่ ซึ่งถูกจัดเก็บในหน่วยเก็บข้อมูลสำรอง โดยปกติไดเรกทอรี่จะประกอบไปด้วยชื่อแฟ้มข้อมูลและรายละเอียดที่ไม่ซ้ำกัน รวมทั้งคุณลักษณะของแฟ้มข้อมูล อาจจะใช้เวลานานในการบันทึกข้อมูลเหล่านี้ ในระบบที่มีอยู่หลายแฟ้มข้อมูล ไดเรกทอรี่ที่เก็บแฟ้มข้อมูลก็จะมีขนาดใหญ่ตามไปด้วย เนื่องมาจากแฟ้มข้อมูลในไดเรกทอรี่ต้องการพื้นที่ในการจัดเก็บข้อมูลเช่นเดียวกับข้อมูลชนิดอื่น ๆ

9.1.2 การดำเนินการกับแฟ้มข้อมูล (File Operations)

แฟ้มข้อมูลจะมีชนิดข้อมูลที่จับต้องไม่ได้ การที่จะกำหนดแฟ้มข้อมูลให้เหมาะสมได้ต้องพิจารณาตัวดำเนินการทั้งหมดที่อยู่ในแฟ้มข้อมูล ระบบปฏิบัติการจะช่วยสนับสนุนทางด้านการสร้าง เขียน และอ่านแฟ้มข้อมูล หรือบางทีก็เป็นการลบหรือตัดทอนแฟ้มข้อมูล ในการตรวจสอบแฟ้มข้อมูลนั้น ระบบปฏิบัติการจะใช้พื้นฐานต่าง ๆ ที่มีอยู่ในการตรวจสอบแฟ้มข้อมูล การดำเนินการจะทำได้ง่ายคล้ายกับการเปลี่ยนชื่อแฟ้มข้อมูลโดยมีการดำเนินการกับแฟ้มข้อมูลดังนี้

1. การสร้างแฟ้มข้อมูล (Creating a file) จะมีอยู่สองขั้นตอนในการสร้าง ขั้นตอนแรกต้องรู้ที่ว่างว่ามีอยู่ในระบบหรือไม่ ขั้นตอนที่สองการสร้างแฟ้มข้อมูลใหม่จำเป็นต้องสร้างอยู่ในไดเรกทอรี่

2. การเขียนแฟ้มข้อมูล (Writing a file) ในการเขียนแฟ้มข้อมูลเราจะทำให้ข้อมูลถูกเขียนขึ้นทั้งชื่อแฟ้มข้อมูลและรายละเอียดที่ถูกเขียนไปยังแฟ้มข้อมูล การระบุชื่อแฟ้มข้อมูลนั้นจะทำให้สามารถหาได้ว่า ที่ตั้งของแฟ้มข้อมูลอยู่ในไดเรกทอรี่ไหน เพื่อหาตำแหน่งของแฟ้มข้อมูลระบบต้องเก็บ ระบบจะทำการเก็บตัวชี้ (Write Pointer) ไปยังตำแหน่งที่หาเจอในครั้งถัดไปที่เขียนอีกครั้ง การชี้ตำแหน่งจะถูกปรับปรุงเมื่อเกิดการเขียนขึ้นทุกครั้ง

3. อ่านแฟ้มข้อมูล (Reading a file) หากเราต้องการอ่านแฟ้มข้อมูลเราก็ต้องรู้ถึงตำแหน่งของแฟ้มข้อมูลในไดเรกทอรี่ โดยต้องให้ตัวชี้ชี้ไปยังตำแหน่งที่อยู่ของแฟ้มข้อมูล ทุกครั้งที่มีการอ่านแฟ้มข้อมูลตัวชี้ (Read pointer) จะมีการปรับปรุงให้เป็นตำแหน่งของแฟ้มข้อมูลปัจจุบัน เพื่อประหยัดพื้นที่และลดความซ้ำซ้อนของระบบในการอ่านและเขียนแฟ้มข้อมูล

4. ที่เก็บแฟ้มข้อมูลภายในแฟ้ม (Repositioning within a file) เริ่มต้นจากการค้นหา ไดเรกทอรี่ที่ต้องการและกำหนดค่าให้ตัวชี้ตำแหน่งแฟ้มข้อมูลปัจจุบัน ที่สามารถค้นหาตำแหน่งที่อยู่ของแฟ้มข้อมูลในไดเรกทอรี่ได้ แฟ้มข้อมูลชี้ตำแหน่งคือ ตำแหน่งที่ทำการระบุค่าของที่เก็บแฟ้มข้อมูล การย้ายตำแหน่งภายในแฟ้มข้อมูลไม่จำเป็นต้องเกี่ยวข้องกับ I/O จริง ๆ เลยการทำงานของแฟ้มนี้เรียกว่า การค้นหา (Seek) ข้อมูล

5. การลบแฟ้มข้อมูล (Deleting a file) ในการลบแฟ้มข้อมูลเราจำเป็นต้องรู้ตำแหน่งของแฟ้มข้อมูลในไดเรกทอรี่ เริ่มจากการค้นหาไดเรกทอรี่ที่มีชื่อจากแฟ้มที่จะลบ โดยจะพบว่ามีการเชื่อมโยงกันของแฟ้มข้อมูลที่อยู่ในไดเรกทอรี่ การลบแฟ้มข้อมูลจึงต้องรู้ตำแหน่งที่แน่นอน

6. การตัดทอนแฟ้มข้อมูล (Truncating a file) เมื่อผู้ใช้ต้องการให้แฟ้มข้อมูลมีคุณลักษณะเหมือนเดิม แต่ต้องการลบเนื้อหาของแฟ้มข้อมูลแทนที่จะลบและสร้างใหม่ ผู้ใช้อาจจะลบเนื้อหาของแฟ้มข้อมูลในบางส่วนแทนที่จะทำการลบแฟ้มข้อมูลทั้งหมด แต่ทำการตัดทอนเนื้อหาบางส่วนที่ไม่เอาแทน ซึ่งช่วยให้พื้นที่ในการจัดเก็บแฟ้มข้อมูลน้อยลง

นอกจากนี้ยังมีการดำเนินการอื่น ๆ อีกเช่น การต่อท้าย (Appending) การเปลี่ยนชื่อ (Renaming) การคัดลอก (Copy) แฟ้มข้อมูลหรือคัดลอกแฟ้มไปสู่อุปกรณ์รับส่งข้อมูลอีกตัวหนึ่ง เช่น เครื่องพิมพ์ หรือ จอภาพ โดยต้องมีการสร้างแฟ้มใหม่และอ่านจากแฟ้มเก่าเพื่อเขียนลงแฟ้มใหม่ การทำงานของแฟ้มข้อมูลโดยส่วนใหญ่มักเกี่ยวข้องกับการค้นหาไดเรกทอรี เพื่อหาช่องที่เกี่ยวข้องกับชื่อแฟ้ม เพื่อหลีกเลี่ยงการค้นหานี้หลาย ๆ ระบบจะเปิดแฟ้มข้อมูลเมื่อถูกใช้ครั้งแรก ระบบปฏิบัติการจะเก็บตารางเล็ก ๆ ที่บรรจุสารสนเทศเกี่ยวกับการเปิดแฟ้มข้อมูลทั้งหมด (Open file table) เมื่อการทำงานเกี่ยวกับแฟ้มถูกร้องขอก็เพียงแต่ใช้ดัชนีในตาราง ทำให้ไม่ต้องใช้การค้นหาอีกต่อไป เมื่อไม่ต้องการใช้แฟ้มข้อมูลนั้นแล้วจะถูกปิดโดยกระบวนการและระบบปฏิบัติการจะเอาแฟ้มข้อมูลออกจาก Open file table

ในบางระบบเมื่อทำการปิดแฟ้มข้อมูล แฟ้มข้อมูลดังกล่าวจะถูกดำเนินการปิดเองโดยอัตโนมัติ แต่เมื่อการทำงานหรือโปรแกรมดังกล่าวเกิดความล้มเหลว ระบบส่วนใหญ่ก็จะใช้วิธีการนี้ในการเปิดปิดแฟ้มข้อมูล แต่ยังไงก็ตามระบบปฏิบัติการก็ต้องการความชัดเจนที่เกิดขึ้นกับการใช้งาน โดยระบบปฏิบัติการจะเช็คค่าความถูกต้องในสิทธิ์ของผู้ใช้ เพื่อให้ผู้ที่ใช้งานสามารถดำเนินการต่าง ๆ กับแฟ้มข้อมูลที่ใช้ได้ โดยสิทธิ์ของการเข้าถึงแฟ้มข้อมูลก็จะถูกเก็บไว้ในพื้นที่เล็ก ๆ ในระบบปฏิบัติการ โดยสรุปมีข้อมูลหลายส่วนที่เกี่ยวข้องกับการเปิดแฟ้มข้อมูล ดังนี้

- **ตัวชี้แฟ้มข้อมูล (File pointer)** ระบบจะไม่ติดตามแฟ้มข้อมูลที่กำลังอ่านและเขียนแฟ้มข้อมูลอยู่ แต่ระบบจะติดตามการอ่านเขียนครั้งสุดท้าย เหมือนตัวชี้ตำแหน่งแฟ้มข้อมูลปัจจุบัน (Current-file-position pointer) ตัวชี้ของแต่ละกระบวนการจะไม่ซ้ำกัน ดังนั้นจึงต้องมีการจัดเก็บแยกคุณลักษณะของแฟ้มข้อมูลบนดิสก์
- **การนับการเปิดแฟ้มข้อมูล (File-open count)** เมื่อแฟ้มข้อมูลที่กำลังใช้งานถูกปิดลง ระบบปฏิบัติการต้องใช้ตารางเปิดแฟ้มข้อมูล (Open file table) ซ้ำอีกครั้ง หรือไม่ก็ใช้ที่ว่างในตาราง โพรเซสหลาย ๆ โพรเซสอาจจะเปิดแฟ้มข้อมูลเดียวกันได้ ระบบต้องรอให้แฟ้มสุดท้ายถูกปิดเสียก่อน จึงจะลบข้อมูลในตารางเปิดแฟ้มอันสุดท้าย และลบตารางนั้นได้
- **ตำแหน่งของแฟ้มข้อมูลบนดิสก์ (Disk location of the file)** การทำงานของแฟ้มข้อมูลส่วนใหญ่ต้องการให้ระบบปรับปรุงข้อมูลภายในแฟ้มข้อมูล สำหรับข้อมูลที่จำเป็นเพื่อระบุ

ตำแหน่งแฟ้มข้อมูลบนดิสก์จะถูกเก็บไว้ในหน่วยความจำ เพื่อหลีกเลี่ยงที่จะต้องอ่านจากดิสก์สำหรับการทำงานแต่ละครั้ง

- **สิทธิ์ในการเข้าใช้ (Access rights)** แต่ละกระบวนการมีการเข้าถึงไม่เหมือนกัน ข้อมูลที่ถูกเก็บเอาไว้ในกระบวนการมีจำนวนมาก จึงต้องมีการกำหนดสิทธิ์ในการเข้าใช้ เพื่อป้องกันการขโมยข้อมูล โดยต้องดูจากคำขอใช้แฟ้มข้อมูลที่ร้องขอเข้ามา

ระบบปฏิบัติการบางระบบจะให้สิ่งอำนวยความสะดวกในการเชื่อมต่อหรือเปิดแฟ้มข้อมูล โดยจะมีคีย์ในการเปิดแฟ้มข้อมูล เพื่อป้องกันกระบวนการอื่นที่เราไม่ต้องการเข้าถึงแฟ้มข้อมูล แฟ้มข้อมูลที่ถูกล็อคมีประโยชน์สำหรับการใช้แฟ้มข้อมูลร่วมกันโดยหลาย ๆ กระบวนการ เช่น เราต้องล็อคแฟ้มข้อมูลเพื่อให้ผู้ใช้ที่มีสิทธิ์จริง ๆ มาแก้ไขในภายหลัง นอกจากนี้ยังอาจจะทำให้ระบบปฏิบัติการที่แตกต่างกันสามารถเข้ามาใช้หรือล็อคแฟ้มข้อมูลร่วมกันได้ โดยเมื่อมีการล็อคแฟ้มข้อมูลผู้ใช้งานจะต้องรู้ว่าระบบปฏิบัติการที่ตนเองกำลังใช้นั้นจะมีกลไกในการปลดล็อคแฟ้มข้อมูลที่จะใช้อย่างไรเพื่อให้เกิดความเท่าเทียมกัน ผู้ใช้จึงต้องมีความสำคัญในการปลดล็อดแฟ้มข้อมูลเหล่านี้

9.1.3 ประเภทของแฟ้มข้อมูล (File types)

เมื่อเรามีการออกแบบระบบแฟ้มที่อยู่ในระบบปฏิบัติการ เราก็ต้องพิจารณาว่าระบบปฏิบัติการควรรู้จักและสนับสนุนแฟ้มข้อมูลประเภทไหน เราต้องรู้ว่าระบบปฏิบัติการต้องการแฟ้มข้อมูลประเภทไหนที่สามารถทำงานร่วมกับแฟ้มข้อมูลในลักษณะที่เหมาะสมได้ เทคนิคโดยทั่วไปในการใช้ประเภทของแฟ้มข้อมูลคือ การที่เรารู้ถึงส่วนหนึ่งที่อยู่ในชื่อแฟ้มข้อมูล ชื่อจะแบ่งออกเป็นสองส่วน โดยชื่อและส่วนที่จะขยายจะถูกค้นด้วยอักขระ

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

ภาพที่ 9.1 ประเภทของแฟ้มข้อมูลทั่วไป
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.511)

ในที่นี้จะทำให้ระบบปฏิบัติการสามารถแยกออกว่าเป็นแฟ้มข้อมูลประเภทไหน โดยทั่วไปแล้วเราจะแบ่งแฟ้มข้อมูลออกเป็นชนิดต่าง ๆ ขึ้นอยู่กับการใช้งานของแฟ้มข้อมูลนั้น ๆ โดยส่วนมากเราจะแยกแยะได้จากนามสกุลหรือส่วนขยาย (Extension) ที่แตกต่างกัน ดังภาพที่ 9.1

9.1.4 โครงสร้างของแฟ้ม (File Structure)

ประเภทของแฟ้มข้อมูลนั้นยังสามารถใช้เพื่อแสดงโครงสร้างที่อยู่ภายในของแฟ้มข้อมูลได้ โดยโครงสร้างของแฟ้มข้อมูลจะต้องตรงกับความหมายของโปรแกรมที่ใช้งานอยู่ การเพิ่มเติมบางส่วนของแฟ้มข้อมูลจะต้องสอดคล้องกับโครงสร้างที่จำเป็นของระบบปฏิบัติการ ตัวอย่างเช่น ระบบปฏิบัติการได้มีการสร้างโครงสร้างเพื่อจะระบุที่อยู่ในหน่วยความจำ โดยการระบุตำแหน่งแรกของชุดคำสั่งบางระบบปฏิบัติการได้ขยายความคิดนี้เป็นชุดของระบบ เพื่อสนับสนุนโครงสร้างของแฟ้มข้อมูลกับชุดคำสั่งพิเศษสำหรับการดำเนินการจัดการกับโครงสร้างของแฟ้มข้อมูล

โครงสร้างข้อมูล หมายถึง ลักษณะการจัดแบ่งพิกัดต่าง ๆ ของข้อมูลสำหรับแต่ละระเบียน (Record) ในแฟ้มข้อมูลเพื่อให้คอมพิวเตอร์สามารถรับไปประมวลผลได้ ประกอบด้วยส่วนต่าง ๆ ดังนี้

1. บิต (Bit : Binary Digit) คือ หน่วยของข้อมูลที่เล็กที่สุดที่เก็บอยู่ในหน่วยความจำภายในคอมพิวเตอร์ ซึ่งบิตจะแทนด้วยตัวเลข คือ 0 หรือ 1 อย่างใดอย่างหนึ่งเท่านั้น เรียกตัวเลข 0 หรือ 1 ถ้ามีหนึ่งหลักว่าเป็นบิต 1 บิต

2. ไบต์ (Byte) หรือ ตัวอักขระ (Character) คือ หน่วยของข้อมูลที่นำบิตหลาย ๆ บิตมารวมกันแทนตัวอักษรแต่ละตัว เช่น A, B, ..., Z, 0, 1, 2, ... ,9 และสัญลักษณ์พิเศษอื่น ๆ เช่น $, &, +, -, *, / โดยตัวอักษร 1 ตัวจะแทนด้วยบิตจำนวน 7 หรือ 8 บิต ซึ่งตัวอักษรแต่ละตัวจะเรียกว่า ไบต์ เช่น ตัวอักษร A เมื่อเก็บอยู่ในคอมพิวเตอร์จะเก็บเป็น 1000001 ส่วนตัวอักษร B จะเก็บเป็น 1000010 เป็นต้น

3. เขตข้อมูล (Field) หรือคำ (Word) คือ หน่วยของข้อมูลที่เกิดจากการนำอักขระหลาย ๆ ตัวมารวมกัน เพื่อแทนความหมายของสิ่งหนึ่งเป็นคำที่มีความหมาย เช่น ชื่อ นามสกุล เป็นต้น

4. ระเบียน (Record) คือ หน่วยของข้อมูลที่มีการนำเขตข้อมูลหลาย ๆ เขตข้อมูล ที่มีความสัมพันธ์กันมารวมกัน หรือค่าของข้อมูลในแต่ละเขตข้อมูลมารวมกัน เพื่อแสดงรายละเอียดข้อมูลในเรื่องใดเรื่องหนึ่ง เช่น ระเบียนหนึ่ง ๆ ของพนักงานประกอบด้วย ฟิลด์ต่าง ๆ เช่น รหัสนักศึกษา ชื่อ หลักสูตร เป็นต้น

5. แฟ้มข้อมูล (File) คือ หน่วยของข้อมูลที่มีการนำระเบียนหลาย ๆ ระเบียนที่มีความสัมพันธ์กันมารวมกัน

6. ฐานข้อมูล (Database) คือ หน่วยของข้อมูลที่มีการนำแฟ้มข้อมูลหลาย ๆ แฟ้มข้อมูล ที่มีความสัมพันธ์กันมารวมกัน เช่น ฐานข้อมูลในระบบทะเบียนนักศึกษาจะประกอบด้วย แฟ้มข้อมูลรายวิชา นักศึกษา การลงทะเบียน ผลการเรียนประจำเทอม โปรแกรมวิชา และคณะ เป็นต้น

9.1.5 โครงสร้างของแฟ้มข้อมูลภายใน (Internal File Structure)

ในระบบดิสก์ มักมีขนาดของบล็อกที่กำหนดโดยขนาดของเซกเตอร์ การเข้าใช้พื้นที่ในดิสก์ทั้งหมด (Disk I/O) ถูกแบ่งเป็นหน่วยของบล็อก (Physical record) และทุกบล็อกมีขนาดเท่ากัน โดย Physical

record จะต้องตรงกับความยาวของ logical record การห่อ (Packing) จำนวนของ Logical record ไปสู่ Physical block เป็นวิธีโดยทั่วไปในการแก้ปัญหานี้

ขนาดของ Logical record ขนาดของ Physical block และเทคนิคการห่อ เป็นตัวกำหนดว่า Logical record จำนวนเท่าไรที่จะอยู่ในแต่ละ Physical block การห่อสามารถทำได้โดยโปรแกรมประยุกต์ของผู้ใช้หรือโดยระบบปฏิบัติการ

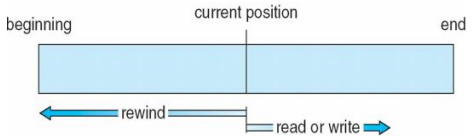
อีกกรณีหนึ่ง แฟ้มข้อมูลอาจถูกพิจารณาเป็นบล็อกแบบเรียงลำดับฟังก์ชัน I/O พื้นฐานทั้งหมดจะทำงานในแบบของบล็อก การแปลงจาก Logical record ไปเป็น Physical block คือปัญหาอย่างง่ายของซอฟแวร์ สังเกตได้ว่าพื้นที่ว่างในดิสก์มักถูกจัดสรรเป็นบล็อก บางส่วนของบล็อกสุดท้ายของแต่ละแฟ้มข้อมูลอาจจะเสียไป ถ้าแต่ละบล็อกมี 512 ไบต์ ดังนั้นแฟ้มขนาด 1949 ไบต์ ควรมี 4 บล็อก (2048 ไบต์) ที่เหลือ 99 ไบต์จะเสียไป นั่นคือ การสูญเสียพื้นที่ย่อยภายใน (Internal fragmentation) ฉะนั้นถ้าบล็อกขนาดใหญ่ก็จะเสียพื้นที่มากตามไปด้วย

9.2 วิธีการเข้าถึงแฟ้มข้อมูล

แฟ้มข้อมูลถูกใช้เก็บข้อมูลสารสนเทศ เมื่อถูกนำมาใช้ย่อมมีการเข้าถึงเพื่ออ่านข้อมูล โดยวิธีในการเข้าถึงข้อมูลนั้นก็แล้วแต่ระบบปฏิบัติการว่าเป็นชนิดไหน แต่การเข้าถึงโดยส่วนใหญ่จะมีวิธีดังนี้

9.2.1 วิธีเข้าถึงโดยลำดับ (Sequential Access)

เป็นเทคนิคที่ใช้ในการเก็บข้อมูลอย่างหนึ่ง เพื่อให้สามารถเรียกมาใช้ได้อย่างมีประสิทธิภาพ ใช้ในโปรแกรมประเภทคลังข้อมูล การเข้าถึงหน่วยเก็บข้อมูลหรือสื่อบางชนิด เช่น แถบแม่เหล็ก (Tape) ซึ่งจะเก็บข้อมูลไว้โดยเรียงไปตามลำดับ วิธีที่ใช้ในการเข้าถึงข้อมูลขึ้นอยู่กับระยะทางของตำแหน่งของข้อมูลที่บรรจุไว้ในสื่อนั้นดังแสดงในภาพที่ 9.2 หากข้อมูลถูกเก็บอยู่ตอนปลายกว่าจะเข้าถึงข้อมูลย่อมจะเสียเวลานาน



ภาพที่ 9.2 แสดงแฟ้มข้อมูลที่มีการเข้าถึงแบบเรียงลำดับ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.513)

9.2.2 การเข้าถึงโดยตรง (Direct Access)

หมายถึง การเข้าถึงข้อมูลโดยใช้เวลาในการค้นหาข้อมูลได้เร็วเท่ากันหมด ไม่ขึ้นกับตำแหน่งที่เก็บ หมายความว่า ไม่ว่าข้อมูลจะเก็บอยู่ที่ส่วนใดของสื่อที่ใช้บันทึก หัวอ่าน (Read head) ก็จะเจาะตรงลงไปอ่านได้เลย เช่น การอ่านข้อมูลจากจานบันทึก ซึ่งผิดกับการอ่านข้อมูลจากแถบบันทึกที่ต้องอ่านเรียงไปตามลำดับตั้งแต่ต้นเทปไปจนกว่าจะถึงข้อมูลที่ต้องการทุกครั้ง ทำให้ช้ากว่ากันมาก

โดยจำนวนของบล็อกจะถูกป้องกันโดยระบบปฏิบัติการที่จะใช้หมายเลขในการป้องกัน ซึ่งการเรียงลำดับของบล็อกคือ 0 ถัดไปก็เป็น 1 ในความเป็นจริงบล็อกที่อยู่ในฮาร์ดดิสก์อาจจะเป็น 14703 สำหรับจุดเริ่มต้นและ 3192 เป็นลำดับถัดไป การใช้หมายเลขกำกับในแต่ละบล็อกช่วยให้ระบบปฏิบัติการตัดสินใจใช้แฟ้มข้อมูลและช่วยป้องกันแฟ้มข้อมูลจากการเข้าถึงที่ไม่ต้องการได้ โดยระบบที่มีหมายเลขกำกับอยู่กับบล็อกจะเริ่มต้นที่ 0 ส่วนระบบอื่นจะเริ่มต้นที่ 1

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

ภาพที่ 9.3 การจำลองคำสั่งการเข้าถึงแบบเรียงลำดับบนแฟ้มข้อมูลที่มีการเข้าถึงแบบโดยตรง

9.2.3 วิธีการเข้าถึงอื่น ๆ (Other Access Methods)

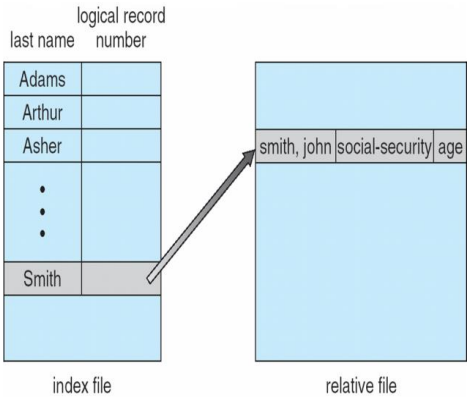
วิธีการเข้าถึงแบบอื่นที่สามารถทำได้นอกจากวิธีการเข้าถึงแบบ Direct-access วิธีการเหล่านี้โดยทั่วไปแล้วเกี่ยวข้องกับโครงสร้างของตัวชี้ของแฟ้มข้อมูล ตัวชี้ของแฟ้มข้อมูลนี้เหมือนกับหน้าดัชนี (Index) ที่อยู่ท้ายเล่มของหนังสือ ประกอบไปด้วยตัวชี้ตำแหน่งในหลาย ๆ ส่วน เอาไว้สำหรับค้นหาข้อมูลในแฟ้มข้อมูล ในการค้นหาครั้งแรกใช้ดัชนีและใช้ตัวชี้ในการเข้าถึงแฟ้มข้อมูล และค้นหาข้อมูลที่เราต้องการวิธีเสริมนี้เกี่ยวข้องกับการสร้างดัชนีให้แฟ้มข้อมูล ดัชนีนี้บรรจุตัวชี้บล็อกในการหาระเบียนในแฟ้มข้อมูล ขั้นแรกต้องค้นหาดัชนีแล้วใช้ตัวชี้ในการเข้าถึงแฟ้มข้อมูลโดยตรงและหาระเบียนที่ต้องการ

ตัวอย่าง แฟ้มข้อมูลเก็บราคาสินค้า แต่ละระเบียนประกอบด้วย รหัสสินค้า 10 หลัก ราคา 6 หลัก รวมเป็น 16 ไบต์ ถ้าดิสก์ของเรามีขนาด 1024 ไบต์/บล็อก สามารถเก็บได้ 64 ระเบียน/บล็อก แฟ้มที่มี 120,000 ระเบียนควรจะมี 2,000 บล็อก (2 ล้านไบต์) ดัชนีควรมี 2,000 ช่อง ของทุก ๆ 10 หลักหรือ 20,000 ไบต์ แล้วควรเก็บไว้ในหน่วยความจำ ในการค้นหาราคาของสินค้าเราสามารถค้นหาดัชนีในการค้นหานี้เราควรจะรู้ว่าบล็อกไหนบรรจุระเบียนที่ต้องการแล้วจึงเข้าถึงบล็อกนั้น โครงสร้างข้อมูลแบบนี้ทำให้เราสามารถค้นหาแฟ้มข้อมูลขนาดใหญ่ได้ด้วยการทำ I/O เพียงนิดเดียว

ถ้าแฟ้มข้อมูลมีขนาดใหญ่ แฟ้มดัชนีก็ต้องมีขนาดใหญ่เกินที่จะเก็บไว้ในหน่วยความจำได้ วิธีแก้วิธีหนึ่ง คือ สร้างดัชนีสำหรับแฟ้มดัชนี (Index file) แฟ้มข้อมูลปฐมภูมิ (Primary index file) ควรจะบรรจุตัวชี้ไปยังแฟ้มข้อมูลทุติยภูมิ (Secondary index file) ซึ่งควรจะชี้ไปยังข้อมูลจริง

ตัวอย่าง ดัชนีแบบเรียงลำดับของ IBM indexed sequential-access เป็นวิธีการเข้าถึงของ IBM (ISAM) ใช้ Master index เล็ก ๆ ซึ่งชี้ไปยังบล็อกของดิสก์ของดัชนีทุติยภูมิ (Secondary index) ดัชนีทุติยภูมิจะชี้ไปยังบล็อกของแฟ้มข้อมูลจริง แฟ้มข้อมูลถูกเก็บแบบเรียงลำดับโดยการกำหนดคีย์ขึ้นมาเพื่อหาข้อมูลที่ต้องการ ขั้นแรกเราต้องค้นหาแบบ Binary search เพื่อหา Master index เพื่อจัดเตรียมหมายเลขบล็อกของดัชนีทุติยภูมิ บล็อกถูกอ่านแล้วค้นหาแบบ Binary search อีกครั้ง

เพื่อหาบล็อกที่บรรจุระเบียนที่ต้องการขั้นสุดท้าย บล็อกนี้จะถูกค้นหาแบบเรียงลำดับ ด้วยวิธีการนี้ระเบียนต่าง ๆ สามารถระบุตำแหน่งจากคีย์ของมัน โดยการอ่านแบบโดยตรงทั้ง 2 ครั้ง ภาพที่ 9.4 แสดงสถานการณ์คล้าย ๆ กัน ซึ่งนำไปใช้โดยดัชนีและแฟ้มข้อมูลแบบสัมพันธ์ของ VMS index และ Relative files



ภาพที่ 9.4 ตัวอย่างของแฟ้มดัชนีและแฟ้มข้อมูลแบบสัมพันธ์
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.516)

9.3 โครงสร้างของไดเรกทอรี่

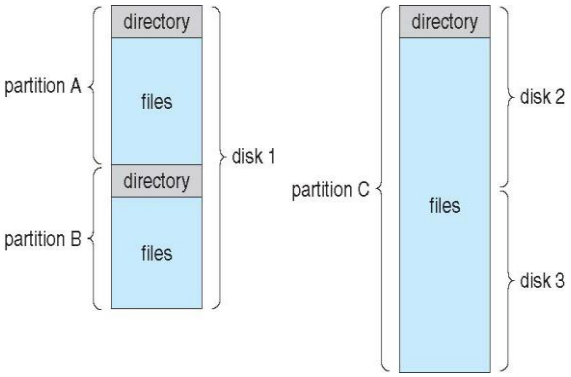
ไดเรกทอรี่ (Directories) เป็นที่เก็บรวบรวมชื่อของแฟ้มข้อมูล และข้อมูลที่สำคัญของแฟ้มข้อมูล โดยมีวัตถุประสงค์สำคัญคือ เพิ่มประสิทธิภาพการทำงานของระบบ และความสะดวกสำหรับผู้ใช้ ด้วยการช่วยให้ผู้ใช้สามารถเข้าถึงแฟ้มข้อมูลที่ต้องการได้อย่างรวดเร็วมากขึ้น รวมถึงช่วยให้ผู้ใช้สะดวกสบายในการใช้แฟ้มข้อมูล ทั้งการตั้งชื่อแฟ้มข้อมูล การจัดกลุ่มแฟ้มข้อมูลตามความต้องการของผู้ใช้

ระบบแฟ้มข้อมูลของคอมพิวเตอร์นั้นยังสามารถขยายออกไปได้อีกในบางระบบ บรรจุแฟ้มข้อมูลไว้เป็นล้านแฟ้มข้อมูลในดิสก์ขนาดเทราไบต์ (Terabyte) การที่จะจัดการกับข้อมูลทั้งหมดต้องมีวิธีการจัดการที่ดี วิธีการจัดการนั้นจะเกี่ยวกับการใช้ไดเรกทอรี่ ดังนั้นเราจะอธิบายโครงสร้างพื้นฐานบางอย่างของโครงสร้างไดเรกทอรี่ ที่เป็นจุดเด่นสำหรับการเก็บข้อมูล ดังนั้นสามารถอธิบายโครงสร้างพื้นฐานบางอย่างของโครงสร้างไดเรกทอรี่ ที่เป็นจุดเด่นสำหรับการเก็บข้อมูล

ในหนึ่งดิสก์ สามารถใช้ระบบแฟ้มข้อมูลได้หลาย ๆ ระบบบนดิสก์เดียว หรือจะแบ่งดิสก์ออกเป็นส่วน ๆ สำหรับแต่ละระบบแฟ้มข้อมูล ระบบแฟ้มข้อมูลแบ่งออกเป็นพาร์ติชั่น (Partitions) ใน IBM เรียก minidisks ในเครื่อง PC และ Macintosh เรียก Volume โดยทั่วไป แต่ละระบบบนดิสก์ถูกบรรจุอย่างน้อย 1 พาร์ติชั่น ผู้ใช้จำเป็นต้องเกี่ยวข้องกับ Logical directory และโครงสร้างของแฟ้มข้อมูล และสามารถมองข้ามปัญหาทางกายภาพในการจัดการพื้นที่ว่างของแฟ้มข้อมูล ด้วยเหตุนี้พาร์ติชั่นสามารถมองเป็นดิสก์เสมือนได้ (Virtual disk)

ในแต่ละ Volume สามารถแบ่งออกมาเพื่อเป็นดิสก์ได้ Volumes สามารถมีระบบได้หลายระบบปฏิบัติการ และอนุญาตให้บูทหรือรันได้มากกว่าหนึ่ง ในแต่ละ Volumes ที่มีระบบแฟ้มข้อมูลอยู่

จะต้องเก็บข้อมูลของระบบแฟ้มข้อมูลเอาไว้ด้วย ในข้อมูลนั้นจะเก็บ Device directory (หรือ Volume table of contents device directory) คือไดเรกทอรี่ที่บรรจุไปด้วย ชื่อ ที่อยู่ ขนาด และชนิดของแฟ้มข้อมูลทั้งหมดบน volumes ในภาพที่ 9.5



ภาพที่ 9.5 ตัวอย่างการจัดระเบียบของระบบแฟ้มข้อมูล
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.516)

เมื่อพิจารณาเข้าไปในโครงสร้างของไดเรกทอรี่โดยเฉพาะแล้ว การนำไดเรกทอรี่ไปใช้มีวิธีในการดำเนินการดังนี้

1. ค้นหาแฟ้มข้อมูล (Search for a file) เมื่อผู้ใช้หรือโปรแกรมเรียกใช้แฟ้มข้อมูลใด ๆ ระบบต้องค้นหาแฟ้มข้อมูลนั้นจากไดเรกทอรี่

2. สร้างแฟ้มข้อมูล (Create a file) แฟ้มข้อมูลใหม่จำเป็นต้องถูกสร้างและเพิ่มเข้าในไดเรกทอรี่

3. ลบแฟ้มข้อมูล (delete a file) เมื่อแฟ้มข้อมูลไม่ต้องการแล้ว ต้องสามารถลบมันออกจากไดเรกทอรี่

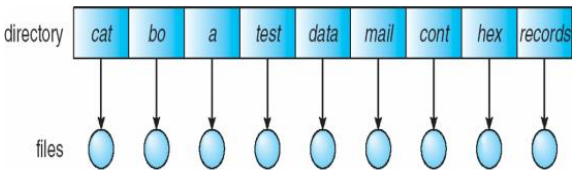
4. แสดงไดเรกทอรี่ (List a directory) การดูแฟ้มข้อมูลในไดเรกทอรี่ ต้องสามารถแสดงชื่อแฟ้มข้อมูลในไดเรกทอรี่ และเนื้อหาของไดเรกทอรี่สำหรับแต่ละแฟ้มในรายการ

5. เปลี่ยนชื่อแฟ้มข้อมูล (Rename a file) เพราะว่าชื่อของแฟ้มข้อมูลนั้นมีหลากหลายเนื้อหาและหลากหลายผู้ใช้ เราต้องสามารถเปลี่ยนชื่อ เมื่อเนื้อหาภายในถูกเปลี่ยนหรือมีการใช้แฟ้มข้อมูลนั้นเปลี่ยนแปลงไป

6. การข้ามระบบแฟ้มข้อมูล (Traverse the file system) การสามารถเข้าถึงในหลายไดเรกทอรี่และหลาย ๆ แฟ้มข้อมูลภายในโครงสร้างไดเรกทอรี่ สำหรับความน่าเชื่อถือ คือ การรักษาเนื้อหาและโครงสร้างของระบบแฟ้มข้อมูลทั้งหมด การรักษานี้มักทำได้โดยการคัดลอกแฟ้มข้อมูลทั้งหมดลงเทปแม่เหล็ก เทคนิคนี้เป็นการ Backup ในกรณีที่ระบบล่มหรือแฟ้มข้อมูลเสีย ในกรณีนี้แฟ้มข้อมูลควรถูกคัดลอกลงเทป และพื้นที่ว่างในดิสก์ของแฟ้มเหล่านั้นก็จะถูกปล่อยให้แฟ้มอื่นได้ใช้ต่อไป

9.3.1 ไดเรกทอรี่ระดับเดียว (Single-Level Directory)

โครงสร้างต่าง ๆ ของไดเรกทอรี่ คือ ไดเรกทอรี่ระดับเดียว แฟ้มข้อมูลทั้งหมดจะถูกเก็บไว้ในไดเรกทอรี่เดียว เป็นโครงสร้างไดเรกทอรี่ที่ง่ายต่อการศึกษาและเข้าใจดังแสดงในภาพที่ 9.6



ภาพที่ 9.6 โครงสร้างไดเรกทอรี่ระดับเดียว
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.519)

ไดเรกทอรี่ระดับเดียวมีความสำคัญอยู่ที่ข้อกำจัดของมัน อย่างไรก็ตามเมื่อตัวเลขของแฟ้มข้อมูลเพิ่มขึ้นหรือระบบมีมากกว่าหนึ่ง ผู้ใช้แฟ้มข้อมูลทั้งหมดที่อยู่ในไดเรกทอรี่เดียวกัน จะต้องมีชื่อไม่เหมือนกัน ถ้าผู้ใช้เรียกแฟ้มข้อมูลที่มีชื่อเดียวกันขึ้นมามันจะเป็นการฝ่าฝืนกฎที่แฟ้มข้อมูลต้องเป็นชื่อเฉพาะแฟ้มข้อมูลนั้น (Unique-name)

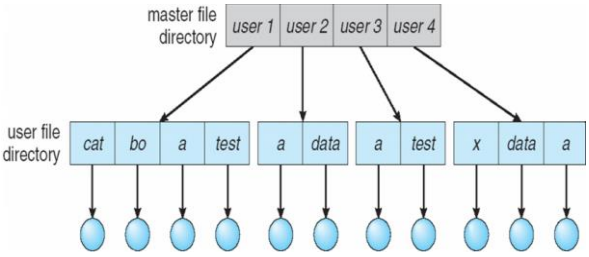
ตัวอย่าง ในห้องเรียนวิชาการเขียนโปรแกรมหนึ่งนั้นมีนักเรียน 23 คน ต้องการเรียกงานชื่อ prog2 อีก 11 คนเรียกโปรแกรม assign2 ถึงแม้ว่าแฟ้มข้อมูลที่ถูกเลือกโดยทั่วไปนั้นจะแสดงให้เห็นเนื้อหาของแฟ้มข้อมูล บ่อยครั้งที่มีเรื่องข้อจำกัดของความยาวของชื่อแฟ้มข้อมูล และยากที่จะทำให้ชื่อแฟ้มข้อมูลไม่ซ้ำกัน ระบบปฏิบัติการ MS-DOS ใช้ตัวอักษรในการตั้งชื่อแฟ้มข้อมูลความยาวทั้งหมดไม่เกิน 11 ตัวอักษร ขณะที่ระบบปฏิบัติการ UNIX ใช้ตัวอักษรไม่เกิน 255 ตัวอักษร

ยิ่งไปกว่านั้นผู้ใช้เพียงคนเดียวบนระบบปฏิบัติการ Single-level directory อาจจะเป็นเรื่องยากที่จะค้นหา จดจำแฟ้มข้อมูลทั้งหมดรวมไปถึงจำนวนแฟ้มข้อมูลที่เพิ่มขึ้นมาที่มีผู้ใช้จำนวนไม่น้อยที่มีแฟ้มข้อมูลเป็นร้อยบนเครื่องคอมพิวเตอร์เครื่องเดียว และเท่ากับจำนวนตัวเลขแฟ้มข้อมูลบนระบบอื่น มันคงจะเป็นเรื่องที่น่ากลัวมากที่เราจะเก็บเส้นทางทั้งหมดของแฟ้มข้อมูลไว้

9.3.2 ไดเรกทอรี่สองระดับ (Two-Level Directory)

พวกเราได้เห็นมาแล้วว่าไดเรกทอรี่ระดับเดียว บ่อยครั้งที่ทำให้สับสนชื่อแฟ้มข้อมูลระหว่างผู้ใช้หลายคน วิธีการมาตรฐานที่จะมาแก้เรื่องนี้ที่จะมาแบ่งไดเรกทอรี่สำหรับผู้ใช้แต่ละคน ในโครงสร้างของไดเรกทอรี่สองระดับ แต่ละผู้ใช้จะมีไดเรกทอรี่ของตนเอง (User file directory : UFD) โดยจะสร้างโครงสร้างจำลองขึ้นมาสำหรับผู้ใช้เพียงคนเดียว เมื่อผู้ใช้เริ่มใช้งานระบบหลัก (Master file directory : MFD) ถึงจะทำการค้นหา MFD จะเป็นตัวดัชนีสำหรับผู้ใช้ และแต่ละจุดจะเป็นที่ไปยังผู้ใช้แต่ละคน

เมื่อผู้ใช้อ้างอิงแฟ้มข้อมูลจะมีเพียง UFD เดียวที่จะถูกค้นหา ดังนั้นผู้ใช้หลาย ๆ คนถึงจะมีแฟ้มข้อมูลชื่อเหมือนกันได้ ตราบเท่าที่ยังมีแฟ้มข้อมูลทั้งหมดที่อยู่ในแต่ละ UFD ที่เหมือนกัน ที่จะสร้างแฟ้มข้อมูลสำหรับผู้ใช้ระบบปฏิบัติการจะทำการค้นหาว่าแฟ้มข้อมูลนั้นมีอยู่แล้วหรือไม่



ภาพที่ 9.7 โครงสร้างของไดเรกทอรี่สองระดับ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.519)

การระบุชื่อแฟ้มข้อมูลในไดเรกทอรี่สองระดับ เราต้องให้ทั้งชื่อของผู้ใช้และชื่อแฟ้มข้อมูล ไดเรกทอรี่สองระดับสามารถมองเป็นต้นไม้ ราก (Root) ของต้นไม้คือ MFD ทายาทโดยตรงคือ UFD ทายาทของ UFD คือ แฟ้มข้อมูลของตัวมันเอง แฟ้มข้อมูลคือ ใบไม้ของต้นไม้นั่นเอง ชื่อของผู้ใช้และชื่อของแฟ้มข้อมูลกำหนดเส้นทาง (Path) ในต้นไม้จากราก (MFD) สู่ใบ (แฟ้มที่ต้องการ) ดังนั้นชื่อของผู้ใช้และชื่อไฟล์กำหนดชื่อของเส้นทาง (Path name) แฟ้มข้อมูลทุกแฟ้มในระบบมีชื่อของเส้นทาง เพื่อระบุชื่อแฟ้มข้อมูลที่มีลักษณะเฉพาะ ผู้ใช้ต้องรู้ชื่อของเส้นทางของแฟ้มที่ต้องการ

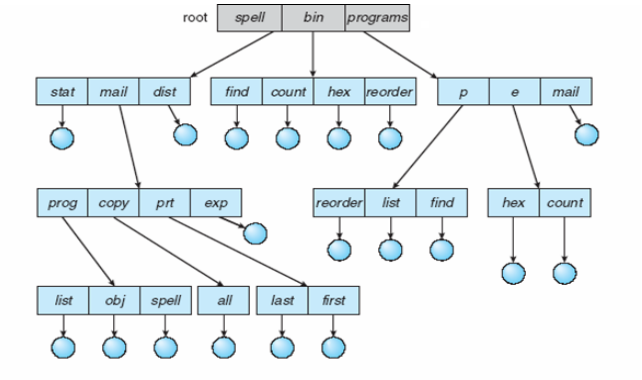
การลบ ระบบปฏิบัติการจะกำหนดขอบเขตการค้นหาอยู่ในแต่ละ UFD ฉะนั้นมันจะไม่สามารถจะไปลบแฟ้มข้อมูลที่ชื่อเหมือนกันของผู้ใช้คนอื่นได้

ตัวอย่าง ในระบบ MS-DOS อาจเป็น “C:\userb\test” แยกเป็น partition ชื่อของไดเรกทอรี่ และชื่อแฟ้มข้อมูลในระบบ VMS ไฟล์ “login.com” อาจเป็น “u:[sst.jdeck]login.com;1”

- “u” คือชื่อของ partition
- “sst” คือ ชื่อไดเรกทอรี่
- “jdeck” คือ ชื่อของไดเรกทอรี่ย่อย (Subdirectory)
- “1” คือ หมายเลขเวอร์ชัน

9.3.3 ไดเรกทอรี่ที่มีโครงสร้างแบบต้นไม้ (Tree-structured directory)

ในการใช้งานปกติ ผู้ใช้แต่ละคนจะมีไดเรกทอรี่ปัจจุบัน (Current directory) ซึ่งใช้เก็บแฟ้มข้อมูลที่ผู้ใช้สนใจในปัจจุบันหรือเป็นตำแหน่งที่กำลังใช้งานอยู่ เมื่อมีการอ้างอิงแฟ้มข้อมูลก็จะเกิดการค้นหาไดเรกทอรี่ปัจจุบัน ถ้าแฟ้มข้อมูลที่ต้องการไม่มีในไดเรกทอรี่ปัจจุบัน ผู้ใช้ต้องระบุชื่อของเส้นทาง (Path name) หรือเปลี่ยนไดเรกทอรี่ปัจจุบันไปเป็นไดเรกทอรี่อื่น โปรแกรมเรียกระบบจะทำให้ชื่อไดเรกทอรี่เป็นเหมือนพารามิเตอร์และใช้มัน เพื่อกำหนดไดเรกทอรี่ปัจจุบันใหม่ ดังนั้นผู้ใช้จึงสามารถเปลี่ยนไดเรกทอรี่ปัจจุบันของผู้ใช้ได้ตามต้องการ



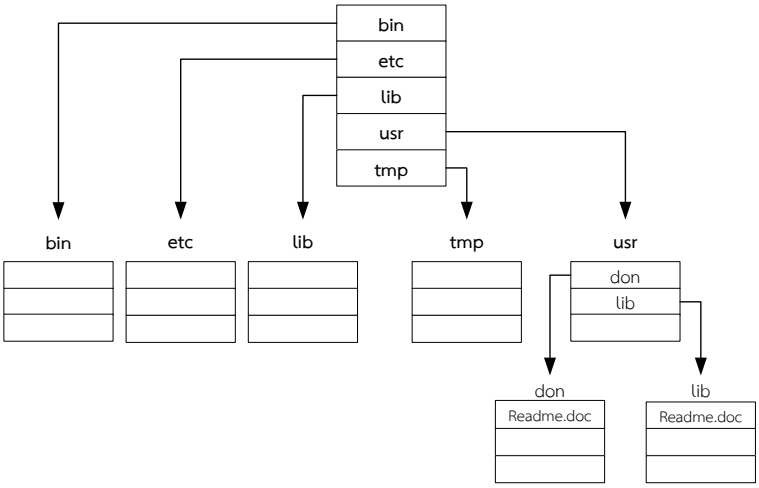
ภาพที่ 9.8 โครงสร้างของไดเรกทอรี่แบบต้นไม้
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.521)

การอ้างอิงถึงแฟ้มข้อมูลใด ๆ ก็ตาม จำเป็นต้องระบุที่อยู่ของแฟ้มข้อมูลนั้น ๆ ให้ถูกต้องว่าอยู่ในไดเรกทอรี่ใด หรือสับไดเรกทอรี่ใด ในการกำหนดที่อยู่หรือเส้นทางที่จะเข้าถึงแฟ้มข้อมูลนั้น ๆ เรียกว่า พาธ (Path) ดังนั้นถ้าต้องการอ้างถึงแฟ้มข้อมูลใด ๆ ในดิสก์ จำเป็นต้องระบุพาธให้ถูกต้องพร้อมชื่อแฟ้มข้อมูลวิธีการอ้างถึงชื่อแฟ้มข้อมูลดังนี้

1. การอ้างชื่อแฟ้มข้อมูลแบบสัมบูรณ์ (Absolute path name)

เป็นการอ้างถึงแฟ้มข้อมูลโดยเริ่มจากราก (Root) เสมอ ตามด้วยชื่อไดเรกทอรี่ย่อยไล่ลงมาตามลำดับชั้นของไดเรกทอรี่จนกระทั่งถึงไดเร็กทอรีที่บรรจุแฟ้มข้อมูลอยู่ และจบลงด้วยชื่อแฟ้มข้อมูลนั้น ๆ ตัวอย่างการอ้างถึงแฟ้มข้อมูลชื่อ Readme.doc ที่อยู่ภายใต้พาธ Root ไดเรกทอรี่ user และไดเรกทอรี่ lib

1) Windows หรือ MS-DOS	\user\lib\readme.doc
2) UNIX หรือ Linux	/user/lib/readme.doc



ภาพที่ 9.9 แบบไดเรกทอรี่ของ UNIX หรือ Linux

2. การอ้างชื่อแบบสัมพัทธ์ (Relative path name)

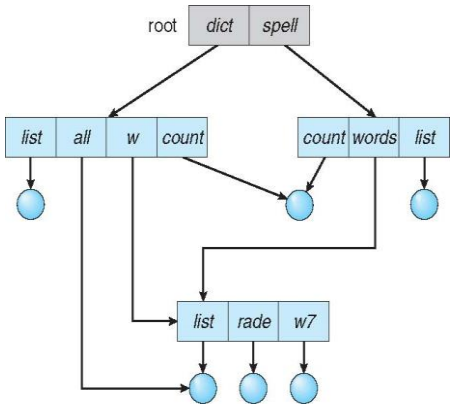
เป็นการอ้างถึงแฟ้มข้อมูลโดยที่ผู้ใช้จะต้องเข้าใจในเรื่องระบบไดเรกทอรี่ปัจจุบัน (Current directory) เนื่องจากการอ้างถึงชื่อแฟ้มข้อมูลจะเริ่มต้นจากไดเรกทอรี่ปัจจุบันแล้วไล่ไปตามลำดับชั้นของไดเรกทอรี่ที่แฟ้มข้อมูลนั้นอยู่และจบลงด้วยชื่อแฟ้มข้อมูลนั้น ตัวอย่างการอ้างถึงแฟ้มข้อมูลชื่อ Readme.doc ที่อยู่ภายใต้พาธ Root ไดเรกทอรี่ user และไดเรกทอรี่ lib โดยที่ไดเรกทอรี่ปัจจุบันอยู่ที่ไดเรกทอรี่ user

1) ระบบปฏิบัติการ Windows หรือ MS-DOS	\lib\readme.doc
2) ระบบปฏิบัติการ UNIX หรือ Linux	/lib/readme.doc

9.3.4 ไดเรกทอรี่กราฟแบบไม่เป็นวงจร (Acyclic-Graph Directory)

เมื่อพิจารณาเหตุการณ์โปรแกรมเมอร์สองคนที่ทำโครงงานร่วมกัน แฟ้มข้อมูลงานจะถูกเก็บในไดเรกทอรี่ย่อยที่แยกมาจากโครงการอื่น และโปรแกรมเมอร์ทั้งสองคนมีการตอบสนองต่อโครงงานเท่า ๆ กัน ทั้งสองคนต้องการมีโครงงานย่อยของตัวเอง ซึ่งไดเรกทอรี่ย่อยนี้ควรจะถูกใช้ร่วมกัน โดยปกติการใช้แฟ้มข้อมูลหรือไดเรกทอรี่ร่วมกันในระบบอาจจะมี 2 หรือมากกว่าต่อหนึ่งการใช้ร่วมกัน

การเปลี่ยนแปลงที่ทำโดยผู้ใช้คนหนึ่งควรจะทำให้คนอื่นเห็นความเปลี่ยนแปลงนี้ได้ในทันที แฟ้มข้อมูลใหม่ที่ถูกสร้างโดยผู้ใช้คนหนึ่ง จะต้องปรากฏในไดเรกทอรี่ย่อยที่ใช้ร่วมกันทันทีโดยอัตโนมัติ ข้อห้ามของการใช้แฟ้มข้อมูลหรือไดเรกทอรี่ร่วมกัน โครงสร้างแบบต้นไม้ห้ามให้มีการใช้แฟ้มข้อมูลหรือไดเรกทอรี่ร่วมกัน กราฟแบบไม่เป็นวงจร (Acyclic graph) อนุญาตให้มีการใช้ไดเรกทอรี่และแฟ้มข้อมูลร่วมกันได้ (ดังภาพที่ 9.10)



ภาพที่ 9.10 โครงสร้างของไดเรกทอรี่กราฟแบบไม่เป็นวงจร
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.523)

การใช้แฟ้มข้อมูลและไดเรกทอรี่ร่วมกันนั้นในทางปฏิบัติแล้วสามารถทำได้หลายวิธี ยกตัวอย่างในระบบ Unix การสร้าง Directory entry นั้นเรียกว่า Link โดยเป็นการใช้พอนย์เตอร์ชี้ไปยังแฟ้มข้อมูลหรือ

ไดเรกทอรี่ย่อย ตัวอย่างเช่น Link อาจเอาไปใช้ในชื่อของเส้นทางแบบสัมบูรณ์ (เรียกว่า Symbolic link) เมื่อเกิดการอ้างอิงแฟ้มข้อมูล เราจะค้นหาไดเรกทอรี่ถ้าไดเรกทอรี่ถูกทำเครื่องหมายเป็น Link ก็จะได้มาซึ่งชื่อของแฟ้มข้อมูลหรือไดเรกทอรี่จริง ๆ

อีกวิธีหนึ่งในการใช้แฟ้มข้อมูลร่วมกัน คือ การคัดลอกสารสนเทศทั้งหมดไปไว้ในไดเรกทอรี่ที่ใช้ร่วมกันทั้งคู่ ดังนั้นไดเรกทอรี่ทั้งคู่ต้องเหมือนกันและเท่ากัน แต่ Link จะแตกต่างกันอย่างชัดเจนจากไดเรกทอรี่ต้นฉบับ ดังนั้นไดเรกทอรี่ทั้งสองจึงไม่เท่ากัน โดยไดเรกทอรี่ต้นฉบับไม่เท่ากับตัวสำเนาที่ใช้ร่วมกัน การคัดลอกไดเรกทอรี่ทำให้ต้นฉบับและตัวสำเนาสามารถแยกจากกันได้

Acyclic graph สามารถปรับให้เข้ากับสถานการณ์ โดยการสร้างต้นไม้แบบง่าย ๆ ได้ แต่เป็นเรื่องที่ยุ่งยากแฟ้มข้อมูลอาจจะมีหลาย Path name ดังนั้นมันแตกต่างอย่างชัดเจนที่จะอ้างถึงแฟ้มข้อมูลเดียวกัน ถ้าเราต้องการค้นหาแฟ้มข้อมูล เก็บสะสมสถิติของแฟ้มข้อมูลทั้งหมดหรือสำรองแฟ้มข้อมูลทั้งหมดเอาไว้ พวกเราไม่สามารถท่องไปในโครงสร้างแบบแชร์ได้มากกว่าหนึ่งครั้ง

อีกปัญหาหนึ่งที่พบคือถ้ามีพื้นที่ว่างในการแชร์แฟ้มข้อมูล เมื่อต้องการพื้นที่ตรงนั้นมาใช้ใหม่ เมื่อได้ทำการลบแฟ้มข้อมูลนั้นไปแล้ว พอนย์เตอร์ยังคงอยู่และมันถูกแขวนไว้โดยไม่ได้อ้างอิงแฟ้มข้อมูลไหนในระบบที่ใช้การแชร์แบบ Link ในสถานการณ์แบบนี้เป็นเรื่องที่ง่ายมาก Link จะหายไปก็ต่อเมื่อแฟ้มข้อมูลนั้นถูกลบออกไป และผู้ใช้จะต้องเข้าใจว่าแฟ้มข้อมูลนั้นถูกลบออกไปแล้วหรือถูกแทนที่ Microsoft windows ก็ใช้วิธีนี้เหมือนกัน

ปัญหาการลบนั้นยังคงอยู่ จนกระทั่งจุดอ้างอิงทั้งหมดถูกลบทิ้งไป อาจจำเป็นจะต้องมีกลไกมากำหนดเมื่อการอ้างอิงครั้งสุดท้ายถูกลบไป การเก็บรายการของการอ้างอิงทั้งหมดไว้เมื่อมีการสร้าง Link ขึ้นมาใหม่ก็จะถูกเพิ่มเข้าไปในรายการเมื่อมีการลบรายการ Link ก็จะถูกลบทิ้งไปด้วย แฟ้มข้อมูลนั้นจะถูกลบทิ้งเมื่อแฟ้มข้อมูล reference นั้นว่างเปล่า

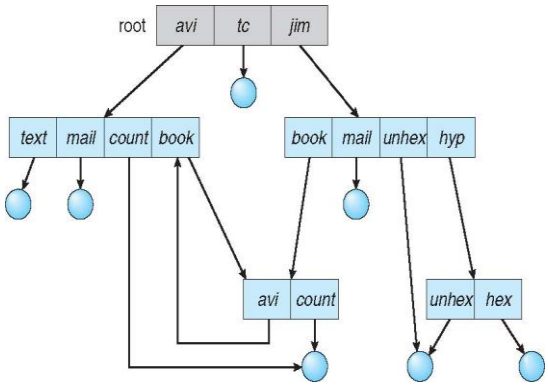
สิ่งที่น่าหนักใจก็คือแฟ้มข้อมูลขนาดใหญ่ การอ้างอิงสามารถแก้ไขได้โดยการเก็บเฉพาะจำนวนเท่านั้น ถ้าจำนวนของแฟ้มข้อมูลนั้นเป็น 0 แฟ้มข้อมูลนั้นคือแฟ้มข้อมูลที่ถูกลบไปแล้ว ในการเลี่ยงปัญหาการแชร์ บางระบบไม่อนุญาตให้แชร์ Link หรือไดเรกทอรี่ เช่น MS-DOS โครงสร้างของไดเรกทอรี่ คือต้นไม้ที่มากกว่า Acyclic graph

9.3.5 ไดเรกทอรี่แบบกราฟโดยทั่วไป (General Graph Directory)

ปัญหาหลักของการใช้ Acyclic graph คือ มันไม่ใช่โครงสร้างที่สมบูรณ์แบบ ถ้าเราเริ่มจาก Two-level และอนุญาตให้ผู้ใช้สร้างไดเรกทอรี่ย่อย ผลก็คือจะกลายเป็นโครงสร้างแบบ Tree-structure ถึงเราจะเพิ่มแฟ้มข้อมูลและไดเรกทอรี่ย่อยอีก ก็ยังเป็นไดเรกทอรี่แบบต้นไม้อยู่ดี แต่ถ้าเราเพิ่มการเชื่อมโยงของไดเรกทอรี่ที่มีอยู่แล้ว โครงสร้างแบบต้นไม้ก็จะเสียไป เป็นผลให้เกิดโครงสร้างแบบกราฟอย่างง่าย

ความได้เปรียบของ Acyclic graph คือ ความสัมพันธ์ที่มีความเรียบง่ายของอัลกอริทึมที่จะท่องไปในกราฟ การค้นหาแฟ้มข้อมูลในไดเรกทอรี่ที่ใช้ร่วมกัน (Share subdirectory) แล้วไม่พบในครั้งแรกควรที่จะเลี่ยงการท่องไปในการแชร์ของพื้นที่ที่แบ่งไว้ของ Acyclic graph เป็นครั้งที่สอง เหตุผลก็เพื่อประสิทธิภาพเป็นหลัก ถ้ามีการค้นหาแบบเฉพาะเจาะจงโดยไม่ต้องทำการค้นหา จำเป็นต้องการเลี่ยงการค้นหาอีกครั้งหนึ่ง การค้นหาครั้งที่สองทำให้เราเสียเวลามาก

ถ้าให้เกิดความสมบูรณ์แล้วการอนุญาตให้มีวงจรในไดเรกทอรี่ คือต้องการหลีกเลี่ยงการค้นหาข้อมูลซ้ำเป็นครั้งที่สอง ถ้าออกแบบอัลกอริทึมไม่ดีจะทำให้เกิดการวนซ้ำ (Loop) ไม่รู้จบ การค้นหาในวงจรก็จะไม่มีวันจบสิ้น วิธีแก้คือจำกัดจำนวนของไดเรกทอรี่ที่จะเข้าใช้ในระหว่างการค้นหา



ภาพที่ 9.11 ไดเรกทอรี่แบบกราฟโดยทั่วไป
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.525)

การเก็บรวบรวมขยะเป็นสิ่งที่จำเป็น เนื่องจากเป็นสิ่งที่เป็นไปได้ในการเกิดวัฏจักรที่แสดงอยู่ในกราฟ ดังนั้นกราฟที่มีโครงสร้างไม่เป็นวัฏจักรจะง่ายในการนำไปใช้ เป็นเรื่องที่ยากในการหลีกเลี่ยงการใช้วัฏจักรเข้าไปเชื่อมต่อในโครงสร้างที่มีอยู่ อย่างไรก็ตามเราสามารถทราบได้ว่าวัฏจักรนั้นจะเสร็จสมบูรณ์ได้เมื่อไร ในกระบวนการขั้นตอนในการลบวัฏจักรในกราฟที่แสดงออกมา จะต้องใช้กระบวนการในการใช้งานที่สิ้นเปลือง

โดยเฉพาะอย่างยิ่งเมื่อเราจะทำการจัดเก็บกราฟลงในดิสก์ เป็นกระบวนการคิดที่ง่ายในกรณีพิเศษที่ใช้กับไดเรกทอรี่ และสามารถเชื่อมโยงข้ามไดเรกทอรี่ที่มีการกีดขวางได้ วัฏจักรที่จะหลีกเลี่ยงและไม่ใช้กรณีพิเศษจะมีตำแหน่งที่จะเกิดข้อผิดพลาด มีอัลกอริทึมในการค้นหาวงจรในกราฟแต่ใช้เวลาในการค้นหามาก ดังนั้นโดยทั่วไปโครงสร้างไดเรกทอรี่แบบต้นไม้น่าใช้กว่าโครงสร้างของกราฟแบบไม่มีวงจร

9.4 การป้องกันการสูญหายของข้อมูล

เมื่อข้อมูลถูกเก็บไว้ในระบบคอมพิวเตอร์ สิ่งที่เราต้องการคือความปลอดภัยทางด้านกายภาพและความปลอดภัยทางด้านการเข้าถึงข้อมูล ความเชื่อถือของการเกิดความซ้ำซ้อนของสำเนาแฟ้มข้อมูลคอมพิวเตอร์ในทุกวันนี้มีระบบหรือโปรแกรมที่ทำงานโดยอัตโนมัติอยู่แล้ว การคัดลอกแฟ้มข้อมูลระหว่างดิสก์นั้น เราสามารถทำได้เป็นระยะ ดังนั้นควรมีการแบ็คอัพข้อมูลเอาไว้เพื่อป้องกันการที่แฟ้มข้อมูลถูกทำลายโดยบังเอิญ แฟ้มข้อมูลในระบบนั้นสามารถที่จะรองรับความเสียหายจากปัญหาทางด้านฮาร์ดแวร์ได้ (เช่น ข้อผิดพลาดในการอ่านและเขียน) ไม่มีไฟเลี้ยง หรือเกิดความผิดพลาดล้มเหลวจากการที่หัวอ่านข้อมูลสกปรก อุณหภูมิภายในเครื่องสูง หรือจากปัญหาอื่น ๆ ข้อผิดพลาดที่อยู่ในระบบแฟ้มข้อมูล ยังสามารถทำให้แฟ้มข้อมูลสูญหายได้

ดังนั้นการป้องกันการสูญหายของข้อมูลจึงเป็นเรื่องสำคัญ ซึ่งอาจจะไม่เห็นความสำคัญในขณะที่ใช้เครื่องเพียงคนเดียวหรือไม่ได้มีการติดต่อกับเครื่องอื่น ๆ แต่มันจะสำคัญมากขึ้น เมื่อมีการติดต่อระหว่างกันมีการแชร์แฟ้มข้อมูลระหว่างกัน การป้องกันข้อมูลจึงมีความสำคัญมากขึ้น

9.4.1 ชนิดของการเข้าถึงแฟ้มข้อมูล (Types of Access)

ความจำเป็นในการป้องกันแฟ้มข้อมูลนั้น เป็นผลมาจากการที่มีการแชร์แฟ้มข้อมูลแล้วสามารถเข้าถึงแฟ้มข้อมูลนั้นได้ เมื่อมีแฟ้มข้อมูลที่ผู้ใช้ไม่ต้องการเปิดเผยจำเป็นต้องมีการป้องกันเพื่อไม่ให้ผู้อื่นเข้ามาใช้งาน เมื่อเป็นเช่นนี้เราจึงต้องมีการป้องกันข้อมูลที่สมบูรณ์ โดยการไม่ให้เข้าถึงข้อมูลหรือก็คือให้เราเข้าถึงแฟ้มข้อมูลเพียงคนเดียว ทั้งสองวิธีนั้นเป็นวิธีการป้องกันที่ดีสำหรับการใช้งานทั่วไป และอะไรคือการควบคุมการเข้าถึงการใช้งาน

การป้องกันโดยการควบคุมการเข้าถึงของข้อมูลโดยการกำหนดสิทธ์ของผู้ใช้ว่ามีประเภทเป็นอะไร ซึ่งเราสามารถกำหนดมันได้ โดยเครื่องแม่ข่ายจะมีประเภทในการกำหนดชนิดดังนี้

- อ่าน (Read) อ่านข้อมูลจากแฟ้มข้อมูล
- เขียน (Write) เขียนหรือเขียนซ้ำแฟ้มข้อมูล
- ดำเนินการ (Execute) อ่านแฟ้มข้อมูลแล้วเอาเข้าไปในหน่วยความจำแล้วดำเนินการกับแฟ้มข้อมูลนั้น
- เพิ่ม (Append) เขียนข้อมูลใหม่จนกระทั่งจบแฟ้มข้อมูล
- ลบ (Delete) ลบแฟ้มข้อมูลออกไปทำให้เกิดพื้นที่ว่าง
- รายการ (List) แสดงชื่อและส่วนประกอบต่าง ๆ ของแฟ้มข้อมูล

การดำเนินการอื่น ๆ เช่น การคัดลอก เปลี่ยนชื่อ และแก้ไขแฟ้มข้อมูล ระบบสามารถดำเนินการได้ทั้งหมด สำหรับชุดคำสั่งที่มีจำนวนมากและซับซ้อนเหล่านี้จะทำงานในระดับที่สูงขึ้นไป โดยจะดำเนินการด้วยระบบโปรแกรมที่จะทำให้เครื่องคอมพิวเตอร์สามารถลดการใช้ทรัพยากรลงไปได้

ดังนั้นเมื่อสารสนเทศถูกเก็บเข้าไว้ในระบบคอมพิวเตอร์ สิ่งที่ต้องทำคือการป้องกันข้อมูลจากความเสียหายทางกายภาพ (ความไว้ใจได้) และการเข้าถึงข้อมูลอย่างไม่เหมาะสม (การป้องกัน)

9.4.2 รายการเข้าถึงแฟ้มข้อมูลและกลุ่ม (Access Lists and Groups)

ส่วนใหญ่วิธีในการป้องกันปัญหานั้นจะขึ้นอยู่กับข้อมูลเฉพาะของตัวผู้ใช้ ผู้ใช้ที่แตกต่างกันอาจจะต้องกำหนดสิทธิ์เพื่อให้เข้าไปใช้แฟ้มข้อมูลหรือไดเรกทอรี่ที่กำหนดไว้เท่านั้น โดยทั่วไปนั้นการเชื่อมโยงของแฟ้มข้อมูลในแต่ละไดเรกทอรี่จะใช้วิธีการเข้าถึงโดยการควบคุม (Access-control list : ACL) ซึ่งวิธีการนี้ต้องระบุถึงชื่อผู้ใช้ และประเภทของการเข้าถึงจะอนุญาตให้ใช้ได้เฉพาะรายเท่านั้น ACL นำมาใช้ในการกำหนดสิทธิ์การเข้าถึงฟังก์ชั่นการทำงานต่าง ๆ ในการกำหนดสิทธิ์นี้จะแบ่งการกำหนดค่าต่าง ๆ ออกเป็นสองส่วนหลัก ๆ คือ

1. ส่วนของ AROs (Access Request Objects) ตรงนี้จะเป็นการกำหนด Object ที่ทำการร้องขอ (Request) ฟังก์ชั่นการทำงาน (Action) ซึ่งโดยส่วนใหญ่แล้วจะหมายถึง กลุ่มของผู้ใช้ หรือตัวผู้ใช้เอง สามารถกำหนดเป็น n-level ได้ หมายถึง กลุ่มของผู้ใช้หรือผู้ใช้ จะสามารถแบ่งเป็นระดับย่อย ๆ ได้หลายระดับ

2. ส่วนของ ACOs (Access Control Objects) ตรงนี้จะใช้กำหนดทรัพยากรต่าง ๆ ในระบบของเราที่ต้องการจะกำหนดสิทธิ์ให้กับผู้ใช้บางกลุ่มหรือบางคนในการเข้าถึงทรัพยากรต่าง ๆ ซึ่งส่วนนี้สามารถกำหนดเป็นระดับย่อยได้เช่นกัน ประเภทในการเข้าถึงแฟ้มข้อมูล โดยการควบคุมจำแนกสิทธิ์ของผู้ใช้ออกเป็น 3 ประเภทคือ

- เจ้าของ (Owner) เป็นเจ้าของแฟ้มข้อมูลสามารถกำหนดสิทธิ์ให้แก่ผู้ใช้งานทั่วไปได้
- กลุ่ม (Group) เป็นกลุ่มของผู้ใช้ซึ่งจะแชร์แฟ้มข้อมูลใช้งานระหว่างกันได้และต้องการ เข้าถึงที่คล้ายกันเป็นกลุ่มหรือทำงานกลุ่ม
- คนอื่นทั้งหมด (Universe) เป็นกลุ่มของผู้ใช้ทั้งหมดที่มีอยู่ในระบบการป้องกันที่เชื่อมโยงกับแฟ้มข้อมูล

ตัวอย่าง Sara กำลังเขียนหนังสือเล่มใหม่ เธอจ้างคนที่จบปริญญาตรีมา 3 คน (Jim, Dawn, Jill) มาช่วยข้อความในหนังสือเก็บไว้ในแฟ้มข้อมูลชื่อ Book การป้องกันแฟ้มข้อมูลทำได้ดังนี้

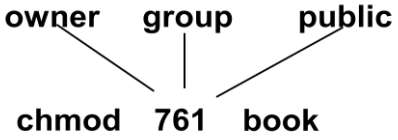
Sara ต้องสามารถทำงานบนแฟ้มข้อมูลนี้ได้ทุกอย่าง Jim, Dawn และ Jill ควรจะอ่านและเขียนแฟ้มข้อมูลนี้ได้เท่านั้น พวกเขาไม่ควรลบแฟ้มข้อมูลนี้ได้ ผู้ใช้คนอื่นควรจะอ่านแฟ้มข้อมูลนี้ได้ (โดย เธอต้องการให้คนทั่วไปได้อ่านและให้ Feedback กลับมา)

เพื่อให้การป้องกันสำเร็จ เราต้องสร้างกลุ่มใหม่ (New group) เรียกว่า Text ให้มีสมาชิกคือ Jim, Dawn และ Jill ชื่อของกลุ่ม text ต้องเกี่ยวข้องกับแฟ้มข้อมูล Book และการเข้าถึงอย่างถูกต้อง เราต้องตั้งตามนโยบายข้างต้น

ตัวอย่างในระบบ UNIX มีการกำหนดบิตขึ้น 3 บิต rwx โดย r หมายถึง ควบคุมการเข้าถึงแบบอ่าน (Read), w หมายถึง ควบคุมการเขียน (Write), x หมายถึง ควบคุมการทำงาน (Execution) ดังนั้นจากตัวอย่างของเรา การป้องกันแฟ้มข้อมูล Book ทำได้ดังนี้

		R W X
สำหรับเจ้าของ (Owner) Sara ทั้ง 3 บิต ถูกตั้งค่าให้เป็นค่า	7 ⇒	1 1 1
สำหรับกลุ่ม (Group) text บิต r และ w ถูกตั้งค่าให้เป็นค่า	6 ⇒	1 1 0
สำหรับคนอื่น (Universe) มีแต่บิต x ที่ตั้งค่าให้เป็นค่า	1 ⇒	0 0 1

ดังนั้นในคำสั่งในระบบ UNIX



ตัวอย่างการแสดงรายการไดเรกทอรี่จาก UNIX แสดงดังภาพที่ 9.12 ฟิลด์แรกบรรยายถึงการป้องกันแฟ้มข้อมูลหรือไดเรกทอรี่ ตัวอักษรแรก d หมายถึงไดเรกทอรี่ย่อย นอกจากนั้นยังแสดงจำนวน Link ของแฟ้มข้อมูล ชื่อเจ้าของ ชื่อของกลุ่ม ขนาดของแฟ้มข้อมูลในหน่วยของไบต์ วันที่สร้าง และชื่อของแฟ้มข้อมูล (ตามด้วยนามสกุล)

```
-rw-rw-r--   1 pbg   staff    31200  Sep 3 08:30   intro.ps
drwx------   5 pbg   staff      512  Jul 8 09.33   private/
drwxrwxr-x   2 pbg   staff      512  Jul 8 09:35   doc/
drwxrwx---   2 pbg   student    512  Aug 3 14:13   student-proj/
-rw-r--r--   1 pbg   staff     9423  Feb 24 2003   program.c
-rwxr-xr-x   1 pbg   staff    20471  Feb 24 2003   program
drwx--x--x   4 pbg   faculty    512  Jul 31 10:31  lib/
drwx------   3 pbg   staff     1024  Aug 29 06:52  mail/
drwxrwxrwx   3 pbg   staff      512  Jul 8 09:35   test/
```

ภาพที่ 9.12 ตัวอย่างการแสดงรายการของไดเรกทอรี่

9.4.3 แนวทางการป้องกันอื่น ๆ

อีกวิธีการในการป้องกันปัญหาที่จะเชื่อมโยงกับรหัสผ่านที่แฟ้มข้อมูล เช่นเดียวกับการเข้าสู่ระบบคอมพิวเตอร์นั้น มักจะควบคุมโดยรหัสผ่านเข้าใช้แฟ้มข้อมูลแต่ละแฟ้มข้อมูลสามารถควบคุม ในทำนองเดียวกัน หากรหัสผ่านจะถูกเลือกแบบสุ่มและการเปลี่ยนแปลงบ่อยจะมีประสิทธิภาพในการจำกัดการเข้าถึงแฟ้มข้อมูล อย่างไรก็ตามการใช้รหัสผ่านมีข้อเสียเหมือนกัน ขั้นแรกจำนวนของรหัสผ่านที่ผู้ใช้ต้องจำอาจมีขนาดใหญ่ สองหากรหัสผ่านหนึ่งรหัสใช้สำหรับทุกแฟ้มข้อมูลเมื่อถูกค้นพบ แฟ้มข้อมูลทั้งหมดสามารถเข้าถึงได้ บางระบบอนุญาตให้ผู้ใช้เชื่อมโยงรหัสผ่านกับไดเรกทอรี่ย่อย แทนที่จะเชื่อมโยงกับแต่ละแฟ้มข้อมูล ปัญหานี้ระบบปฏิบัติการ IBMVM / CMS ยอมให้มีสามรหัสผ่านสำหรับ minidisk โดยแต่ละระดับของรหัสผ่านใช้สำหรับอ่าน เขียน และ multiwrite

บางระบบปฏิบัติการเช่น MS-DOS และระบบปฏิบัติการก่อนหน้า Mac OS X ด้านการป้องกันแฟ้มข้อมูลนั้นมีเพียงเล็กน้อย ในสถานการณ์ที่ระบบเก่าเหล่านี้จะมีการวางแฟ้มข้อมูลไว้ในเครือข่ายที่แชร์แฟ้มข้อมูลและการสื่อสารที่จำเป็นต้องมีการป้องกัน การออกแบบระบบปฏิบัติการใหม่จะทำได้ง่ายกว่าการเพิ่มคุณสมบัติให้ระบบปฏิบัติการที่มีอยู่ การปรับปรุงดังกล่าวมักจะทำให้มีประสิทธิภาพลดน้อยลง

ในระบบ Multilevel โครงสร้างไดเรกทอรี่ต้องป้องกันไม่เพียงแต่ละแฟ้มข้อมูลแต่ยังต้องป้องกันชุดของแฟ้มข้อมูลในไดเรกทอรี่ย่อย การทำงานของไดเรกทอรี่ต้องมีการป้องกันที่ค่อนข้างแตกต่างจากการดำเนินงานแฟ้มข้อมูล และจำเป็นต้องควบคุมการสร้างและลบแฟ้มข้อมูลในไดเรกทอรี่ นอกจากนี้อาจต้องการควบคุมว่าผู้ใช้สามารถตรวจสอบการทำงานของแฟ้มข้อมูลในไดเรกทอรี่ บางครั้งการรู้จักชื่อของแฟ้มข้อมูลก็มีความสำคัญในตัวเอง ดังนั้นรายการเนื้อหาในไดเร็กทอรี่ต้องป้องกันการดำเนินงานในทำนองเดียวกัน หากเส้นทางดูที่ชื่อแฟ้มข้อมูลในไดเรกทอรี่ของผู้ใช้จะต้องได้รับอนุญาตให้เข้าถึงทั้ง ไดเรกทอรี่และแฟ้มข้อมูลในระบบที่มีหลายแฟ้มข้อมูลและหลายชื่อ การกำหนดเส้นทางที่ทำให้ผู้ใช้อาจมีสิทธิในการเข้าใช้ที่แตกต่างกันหนึ่งแฟ้มข้อมูลขึ้นอยู่กับการใช้ชื่อพาธ

9.5 สรุป

ในการจัดเก็บข้อมูลลงในสื่อเก็บข้อมูลจะต้องมีการตั้งชื่อกลุ่มข้อมูล ชื่อที่ว่านี้เรียกว่าแฟ้มข้อมูล ในการเข้าถึงแฟ้มข้อมูล ไม่จำเป็นต้องทราบตำแหน่งที่อยู่ของแฟ้มข้อมูล ระบบปฏิบัติการจะจัดการให้ผ่านทางการดำเนินการจากโปรแกรมของระบบที่เรียกว่า System call ซึ่ง System call เหล่านี้จะช่วยในการสร้าง ลบ อ่าน และเขียนแฟ้มข้อมูลต่าง ๆ ได้ ระบบปฏิบัติการยังช่วยเก็บไฟล์ในกลุ่มเดียวกันไว้ในไดเรกทอรี่ ซึ่งไดเรกทอรี่เป็นแฟ้มข้อมูลประเภทหนึ่ง ที่ช่วยแบ่งกลุ่มแฟ้มข้อมูลที่จัดเก็บในสื่อที่เป็นกลุ่มเพื่อให้การใช้งานมีประสิทธิภาพ ทั้งโครงสร้างไดเรกทอรี่มีทั้งแบบไดเรกทอรี่เดี่ยว ไดเรกทอรี่สองระดับ และไดเรกทอรี่หลายระดับ

ธรรมชาติลักษณะทั่วไปของไดเรกทอรี่สองระดับ คือโครงสร้างแบบต้นไม้ โครงสร้างแบบต้นไม้ของไดเรกทอรี่อนุญาตให้ผู้ใช้สร้างไดเรกทอรี่ย่อย ในการจัดระเบียบไดเรกทอรี่กราฟแบบไม่เป็นวงจร โครงสร้างของไดเรกทอรี่ผู้ใช้จะใช้ในการแบ่งปันแฟ้มข้อมูลและไดเรกทอรี่ย่อย แต่มีความซับซ้อนในการค้นหาและลบโครงสร้างทั่วไปของกราฟ จะช่วยให้เกิดความยืดหยุ่นในการใช้งานร่วมกันของแฟ้มข้อมูลและไดเรกทอรี่ แต่บางครั้งต้องเก็บรวบรวมขยะเพื่อกู้คืนไว้ในส่วนที่ไม่ได้ใช้ในพื้นที่ของดิสก์

เนื่องจากแฟ้มข้อมูลเป็นข้อมูลหลักในเกือบทุกระบบของคอมพิวเตอร์ การป้องกันแฟ้มข้อมูลจึงมีความต้องการมาก การเข้าถึงแฟ้มข้อมูลสามารถควบคุมแยกกันสำหรับแต่ละประเภท การเข้าถึง อ่าน เขียน ทำงาน ลบรายการไดเรกทอรี่ และการใช้ไดเรกทอรี่ร่วมกัน การป้องกันแฟ้มข้อมูลสามารถป้องกันโดยรหัสผ่านก่อนเข้าถึงรายการ หรือเทคนิคอื่น ๆ ในแต่ละระบบปฏิบัติการ

แบบฝึกหัดท้ายบทที่ 9

1. จงอธิบายคุณลักษณะของแฟ้มข้อมูลคืออะไร มีรายละเอียดอย่างไรบ้าง
2. จงอธิบายถึงวัตถุประสงค์ของการดำเนินการใช้ฟังก์ชั่นเพื่อใช้ในการเปิดแฟ้มข้อมูล และปิดแฟ้มข้อมูล
3. จงอธิบายประเภทของแฟ้มข้อมูลที่คุณรู้จักและบอกว่าประเภทของแฟ้มข้อมูลนี้ใช้โปรแกรมเกี่ยวกับอะไรหรือดำเนินการอย่างไร
4. จงยกตัวอย่างของโปรแกรมที่ใช้วิธีการเข้าถึงแฟ้มข้อมูลแบบ Sequential และ แบบ Random
5. จงอธิบายโครงสร้างข้อมูล (แฟ้มข้อมูล Structure) หมายถึงอะไร ลักษณะการจัดแบ่งพิกัดต่าง ๆ ของข้อมูลในแฟ้มข้อมูล เพื่อให้คอมพิวเตอร์สามารถรับไปประมวลผลได้มีอะไรบ้าง
6. จงอธิบายข้อแตกต่างระหว่างไดเรกทอรี่แบบกราฟโดยทั่วไปกับแบบ ไดเรกทอรี่กราฟแบบไม่เป็นวงจร
7. จงอธิบายการกำหนดชื่อพาธมีไว้เพื่ออะไร และยกตัวอย่างการกำหนดชื่อพาธของระบบปฏิบัติการวินโดว์ และระบบปฏิบัติการลินุกส์ เพื่อให้สามารถเรียกใช้แฟ้มข้อมูลนั้นจาก Root
8. จงอธิบายประเภทในการเข้าถึงแฟ้มข้อมูลโดยการควบคุมจำแนกสิทธิ์ของผู้ใช้ การป้องกันและกำหนดสิทธิ์ในระบบปฏิบัติการยูนิกส์
9. ให้นักศึกษาค้นคว้าเพิ่มเติมว่า ในระบบปฏิบัติการในปัจจุบันที่กำหนดแนวทางป้องกันการเข้าถึง

แฟ้มข้อมูล และการเข้าถึงไดเรกทอรี่และไดเรกทอรี่ย่อยว่าใช้หลักการอย่างไร โดยยกตัวอย่างมาอย่างน้อย 1 ระบบปฏิบัติการ

บรรณานุกรมประจำบทที่ 9

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

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

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

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

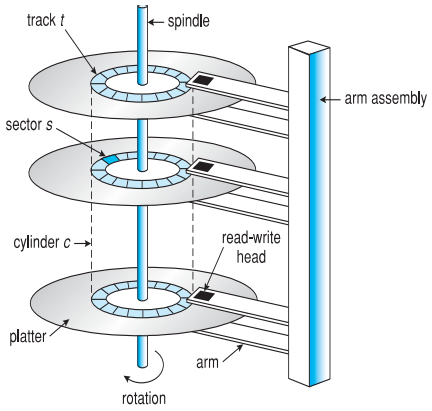
Andrew S. Tanenbaum, Herbert Bos. (2014). **Modern Operating Systems. 4th ed.** Prentice Hall, Pearson Education International.

บทที่ 10
โครงสร้างของหน่วยเก็บข้อมูลสำรอง

ในบทนี้จะอธิบายถึงโครงสร้างของหน่วยเก็บข้อมูลในระดับต่ำที่สุดในระบบแฟ้มข้อมูล คือ โครงสร้างของหน่วยเก็บข้อมูลสำรอง

10.1 โครงสร้างของดิสก์

ดิสก์ คือ ก้อนของหน่วยเก็บข้อมูลสำรองสำหรับระบบคอมพิวเตอร์ยุคใหม่ ส่วนประกอบภายในฮาร์ดดิสก์จะประกอบด้วยแผ่นวงกลมที่มีขนาด 2-5.25 นิ้วเรียงซ้อนกัน ซึ่งเราเรียกแผ่นวงกลมนี้ว่า Disk ซึ่งจะถูกบรรจุอยู่ภายในกล่องที่ปิดสนิทกันฝุ่นและอากาศ เพื่อไม่ให้เข้าไปทำความเสียหายแก่ดิสก์ด้านในการอ่านข้อมูลนั้นภายในฮาร์ดดิสก์จะมีหัวอ่าน 2 หัวต่อแผ่นดิสก์ 1 จาน ซึ่งจะช่วยกันอ่านและเขียนข้อมูล ซึ่งพื้นผิวของจานเราจะเรียกว่า Platter โดยจะถูกแบ่งส่วนภายในจานซึ่งเรียกว่า Track ส่วนวิธีการนับจะนับจากวงนอกสุดของจาน จะเป็น Track 0 ถ้าจำนวนของ Track ยิ่งมากความจุของฮาร์ดดิสก์จะมีมากขึ้นตามด้วย และ Track แบ่งออกเป็นส่วน ๆ ตามแนวเส้นผ่านศูนย์กลางที่เรียกว่า เซ็กเตอร์ (Sector) และไซลินเดอร์ (Cylinders) จะใช้เรียกตำแหน่งของ Track ของ Platter



ภาพที่ 10.1 แสดงกลไกของดิสก์
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.468)

เครื่องขับดิสก์ (Disk drive) ยุคใหม่ ถูกพูดถึงเหมือนเป็นอาร์เรย์ 1 มิติขนาดใหญ่ของบล็อกทางตรรกะ (Logical block) ซึ่งบล็อกทางตรรกะคือหน่วยที่เล็กที่สุดของการโอนย้ายข้อมูล ขนาดของบล็อกทางตรรกะมักมีขนาด 512 ไบต์ ถึงแม้ว่าดิสก์บางตัวจะสามารถทำ “การจัดระเบียบระดับต่ำ” (Low-level formatted) เพื่อเลือกขนาดของบล็อกที่ต่างออกไป เช่น 1024 ไบต์ อาร์เรย์ 1 มิติของบล็อกทางตรรกะถูกจับคู่ (Map) ไปบนเซกเตอร์ของดิสก์แบบเรียงลำดับ เซกเตอร์ 0 คือ เซกเตอร์แรกของ แทร็

กแรกบนไซลินเดอร์นอกสุด การจับคู่ดำเนินการผ่านแทร็กแล้วผ่านแทร็กไปยังไซลินเดอร์ แล้วผ่าน ไซลินเดอร์จากนอกสุดไปสู่ชั้นในสุด

โดยใช้การจับคู่แบบนี้ ทำให้การแปลงหมายเลขบล็อกทางตรรกะมาเป็นตำแหน่งของดิสก์จริง ๆ ซึ่งประกอบด้วย หมายเลขไซลินเดอร์ หมายเลขแทร็กภายในไซลินเดอร์นั้น และหมายเลขเซกเตอร์ภายในแทร็กนั้น สามารถเป็นไปได้ในทางปฏิบัติ เป็นการยากที่จะแปลงเลขดังกล่าวด้วยเหตุผล 2 ประการ

1. ดิสก์ส่วนใหญ่มีเซกเตอร์เสีย แต่การจับคู่จะทำได้โดยใช้เซกเตอร์อื่นในดิสก์แทน
2. จำนวนของเซกเตอร์ต่อแทร็กไม่คงที่ แทร็กมาจากจุดศูนย์กลางของดิสก์ ยิ่งแทร็กยาวมากจำนวนเซกเตอร์ก็จะมากตาม

ดังนั้นดิสก์ในยุคใหม่จะถูกจัดระเบียบเป็นโซนของไซลินเดอร์ จำนวนของเซกเตอร์ต่อ แทร็ก คือค่าคงที่ภายในโซน แต่เมื่อเราเคลื่อนจากโซนภายในไปสู่โซนภายนอก จำนวนของเซกเตอร์ต่อแทร็กจะเพิ่มขึ้น ตัวอย่างเช่น แทร็กในโซนนอกสุดมีเซกเตอร์ 40% มากกว่าแทร็กในโซนในสุด จำนวนของเซกเตอร์ต่อแทร็กมีจำนวนเพิ่มขึ้นเมื่อเทคโนโลยีของดิสก์ได้รับการปรับปรุง และเป็นเรื่องปกติที่จะมีเซกเตอร์ต่อแทร็กมากกว่า 100 เซกเตอร์ในโซนภายนอกของดิสก์ ในทำนองเดียวกันจำนวนของไซลินเดอร์ต่อดิสก์ก็จะเพิ่มขึ้น

10.2 การจัดตารางของดิสก์

หนึ่งในความรับผิดชอบของระบบปฏิบัติการ คือ ต้องใช้ฮาร์ดแวร์อย่างมีประสิทธิภาพ สำหรับเครื่องขับดิสก์นั่นหมายถึง มีเวลาในการเข้าถึงอย่างรวดเร็ว (Fast access time) และ Disk bandwidth ในเรื่องเวลาในการเข้าถึงมี 2 องค์ประกอบหลัก คือ

1. เวลาในการค้นหา (Seek time) คือ เวลาที่แขนของดิสก์เคลื่อนหัวอ่านไปสู่ไซลินเดอร์ที่มีเซกเตอร์ที่ต้องการ

2. เวลาในการหมุนหัวอ่าน (Rotational latency) คือ เวลาในการรอคอยที่เพิ่มขึ้นสำหรับดิสก์ในการหมุนเซกเตอร์ที่ต้องการมาสู่หัวอ่าน

Disk bandwidth คือ จำนวนไบต์ทั้งหมดที่ถูกโอนย้ายมาจากเวลาทั้งหมดตั้งแต่การร้องขอบริการครั้งแรกจนถึงการโอนย้ายครั้งสุดท้ายเสร็จสิ้น เราสามารถปรับปรุงทั้งเวลาในการเข้าถึง และ Bandwidth โดยการจัดตารางบริการของการร้องขอรับ-ส่งข้อมูลของดิสก์ในลำดับที่ดี เมื่อมีโพรเซสที่ต้องการรับข้อมูลจากดิสก์หรือส่งข้อมูลไปสู่ดิสก์ จึงเกิดการเรียกระบบไปสู่ระบบปฏิบัติการ การร้องขอเจาะจงสารสนเทศหลายชิ้น ดังนี้คือ

- การปฏิบัติการนี้ คือ การนำเข้าหรือการส่งออก
- ตำแหน่งของดิสก์ที่ใช้ในการโอนย้ายข้อมูลคืออะไร
- ตำแหน่งของหน่วยความจำสำหรับการโอนย้ายข้อมูลคืออะไร
- จำนวนของไบต์ที่ถูกโอนย้ายข้อมูลเป็นเท่าไร

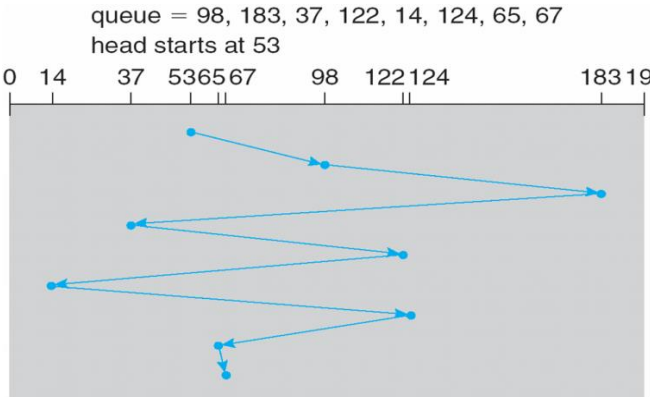
- ถ้าเครื่องขับดิสก์และตัวควบคุมว่าง การร้องขอจะได้รับบริการทันที ถ้ามันไม่ว่างการร้องขอใหม่จำเป็นต้องนำไปไว้ในแถวคอย

สำหรับระบบ Multiprogramming ที่มีหลายโพรเซสเข้าแถวคอยของดิสก์ (Disk queue) อาจมีการร้องขออยู่หลายตัว ดังนั้นเมื่อการร้องขอหนึ่งได้รับบริการแล้ว ระบบปฏิบัติการจึงมีโอกาสที่จะเลือกการร้องขอมาให้บริการถัดไป

10.2.1 การจัดตารางแบบมาก่อน-ได้ก่อน (FCFS Scheduling)

รูปแบบที่ง่ายที่สุดของการจัดตารางของดิสก์ คือ มาก่อน-ได้ก่อน (First-come, First-served : FCFS) พิจารณาตัวอย่าง ถ้าแถวคอยของดิสก์ มีการร้องขอการรับส่งข้อมูลของบล็อกบนไซลินเดอร์ดังนี้

98 , 183 , 37 , 122 , 14 , 124 , 65 , 67 ตามลำดับ ถ้าหัวอ่านเริ่มต้นที่ไซลินเดอร์ที่ 53 มันจะเริ่มเคลื่อนในครั้งแรกจาก 53 ไป 98 แล้วจึงไป 183 , 37 , 122 , 14 , 124 , 65 และสุดท้ายที่ 67 สำหรับการเคลื่อนย้ายหัวอ่านทั้งหมด 640 ไซลินเดอร์ การจัดตารางนี้คือแผนภาพดังภาพที่ 10.2



ภาพที่ 10.2 กราฟแสดงการจัดตารางของดิสก์แบบมาก่อน-ได้ก่อน (FCFS)
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.474)

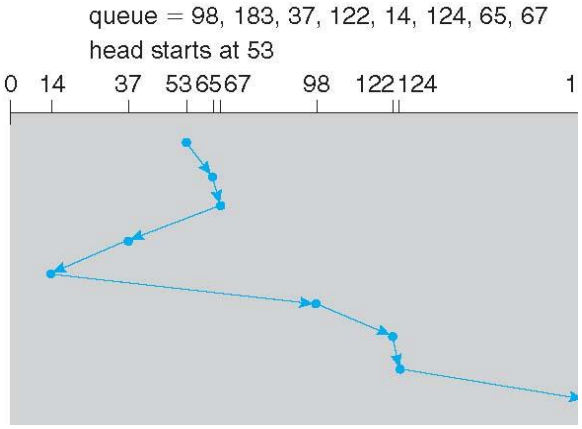
ปัญหาของการจัดตารางวิธีนี้ดูได้จากเส้นกราฟที่มีการเหวี่ยงกว้างจาก 122 ไป 14 แล้วกลับมา 124 ถ้าการร้องขอสำหรับไซลินเดอร์ 37 และ 14 ได้รับบริการต่อกัน ก่อนหรือหลังการร้องขอที่ 122 และ 124 การเคลื่อนย้ายหัวอ่านทั้งหมดน่าจะลดลงอย่างมาก ดังนั้นสมรรถนะ (Performance) ควรจะได้รับการปรับปรุง

10.2.2 การจัดตารางแบบเวลาในการค้นหาสั้นที่สุดได้ก่อน (SSTF Scheduling)

การที่จะบริการการร้องขอทั้งหมดที่อยู่ใกล้ตำแหน่งของหัวอ่านในขณะนั้นก่อนจะย้ายหัวอ่านไกลออกไปเพื่อบริการการร้องขออื่น สมมติฐานนี้คือพื้นฐานสำหรับวิธีเวลาในการค้นหาสั้นที่สุดได้ก่อน (Shortest-seek Time-first : SSTF) วิธี SSTF จะเลือกการร้องขอที่มีเวลาในการค้นหาน้อยที่สุดจาก

ตำแหน่งปัจจุบันของหัวอ่าน เมื่อเวลาในการค้นหาเพิ่มขึ้นด้วยจำนวนของไซลินเดอร์ที่ถูกอ่าน โดยหัวอ่าน SSTF จะเลือกการร้องขอที่ใกล้ที่สุดกับตำแหน่งปัจจุบันของหัวอ่าน

จากตัวอย่างที่ผ่านมา จะเห็นได้ว่าการร้องขอที่ใกล้กับตำแหน่งเริ่มต้นของหัวอ่านที่สุด (53) คือ ที่ไซลินเดอร์ 65 เมื่อเราเคลื่อนมาที่ไซลินเดอร์ 65 การร้องขอที่ใกล้ที่สุดถัดไปคือที่ไซลินเดอร์ 67 จากนั้นการร้องขอที่ไซลินเดอร์ 37 จะใกล้กว่า 98 ดังนั้น 37 จะได้รับบริการถัดไป ต่อมาทำการบริการการร้องขอที่ไซลินเดอร์ 14 แล้วเป็น 98 , 122 , 124 และสุดท้ายที่ 183 (ดังภาพที่ 10.3) ผลลัพธ์ของวิธีการจัดตารางแบบนี้การเคลื่อนย้ายหัวอ่านทั้งหมดมีเพียง 236 ไซลินเดอร์เท่านั้น น้อยกว่า 1 ใน 3 ของระยะทางของวิธี FCFS วิธีนี้ทำให้สมรรถนะได้รับการปรับปรุงอย่างมาก



ภาพที่ 10.3 กราฟแสดงการจัดตารางของดิสก์แบบเวลาในการค้นหาสั้นที่สุดได้ก่อน (SSTF)
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.475)

การจัดตารางแบบ SSTF จำเป็นที่สุดสำหรับรูปแบบของการจัดตารางแบบ SJF และเหมือนกับ SJF มันอาจจะเกิดปัญหาการอดตาย (Starvation) ยังจำได้หรือไม่ว่าการร้องขออาจมาถึงที่เวลาต่าง ๆ กัน สมมติว่าเรามีการร้องขอ 2 ตัวในแถวคอย สำหรับไซลินเดอร์ 14 และ 186 และขณะที่กำลังบริการการร้องขอจาก 14 การร้องขอใหม่ใกล้ 14 มาถึง การร้องขอใหม่นี้จะได้รับบริการเป็นรายถัดไป ทำให้การร้องขอที่ 186 ต้องรอก่อน ในขณะที่การร้องขอนี้กำลังได้รับบริการ การร้องขออีกตัวที่ใกล้ 14 มาถึงอีก ในทางทฤษฎีถ้าสายการร้องขออย่างต่อเนื่องที่ใกล้อีกตัวหนึ่งมาถึงจะเป็นเหตุให้การร้องขอสำหรับไซลินเดอร์ 186 จะต้องรออย่างไม่สิ้นสุด สิ่งนี้จะเป็นการเพิ่มแถวคอยการร้องขอให้ยาวขึ้น

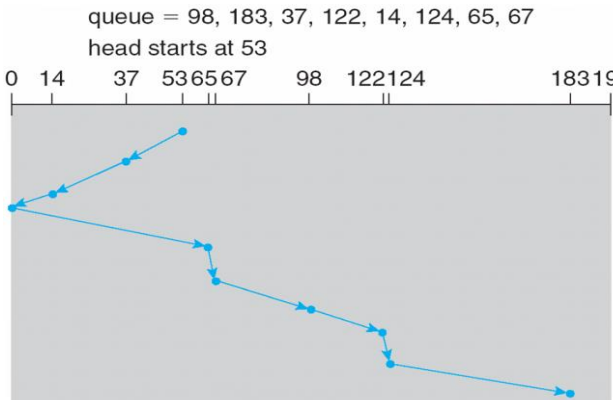
ถึงแม้ว่าวิธี SSTF คือ การปรับปรุงอย่างมากจาก FCFS แต่มันยังไม่ดีที่สุด จากตัวอย่าง เราสามารถทำได้ดีกว่าโดยการเคลื่อนหัวอ่านจาก 53 ไป 37 ถึงแม้ว่า 37 จะไม่ใกล้ที่สุด จากนั้นไป 14 ก่อนจะกลับไป 65 , 67 , 98 , 122 , 124 และ 183 กลยุทธ์นี้เป็นการลดการเคลื่อนย้ายหัวอ่านทั้งหมดเหลือ 208 ไซลินเดอร์

10.2.3 การจัดตารางแบบกวาด (SCAN Scheduling)

ในวิธีแบบกวาด (SCAN) แขนของดิสก์เริ่มต้นที่จุดสิ้นสุดจุดหนึ่งของดิสก์ (A) และเคลื่อนไปยังจุดสิ้นสุดอื่น (B) การบริการการร้องขอจะทำได้เมื่อมันมาถึงในแต่ละไซลินเดอร์ จนกระทั่งมันไปถึง

จุดสิ้นสุดอื่นของดิสก์ (B) ที่จุดสิ้นสุดอื่น (B) ทิศทางของการเคลื่อนของหัวอ่านจะถูกย้อนกลับและให้บริการต่อไป หัวอ่านจะกวาดไปข้างหน้าและข้างหลังผ่านดิสก์ไปเรื่อย ๆ

พิจารณาตัวอย่างเดิมก่อนจะใช้วิธีจัดตารางแบบกวาด กับการร้องขอบนไซลินเดอร์ 98, 183, 37, 122, 14, 124, 65 และ 67 เราจำเป็นต้องรู้ทิศทางของการเคลื่อนหัวอ่านก่อน โดยถ้าตอนนี้ตำแหน่งปัจจุบันของหัวอ่านคือ 53 ถ้าแขนของดิสก์กำลังเคลื่อนไปทาง 0 หัวอ่านจะบริการ 37 และจากนั้นเป็น 14 ที่ไซลินเดอร์ 0 แขนจะย้อนกลับและจะเคลื่อนผ่านจุดสิ้นสุดอื่นของดิสก์ (ที่ไม่ใช่ 0) การบริการการร้องขอจะทำที่ 65 , 67 , 98 , 122 , 124 และ 183 (ดังภาพที่ 10.4)



ภาพที่ 10.4 กราฟแสดงการจัดตารางของดิสก์แบบกวาด (SCAN)
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.476)

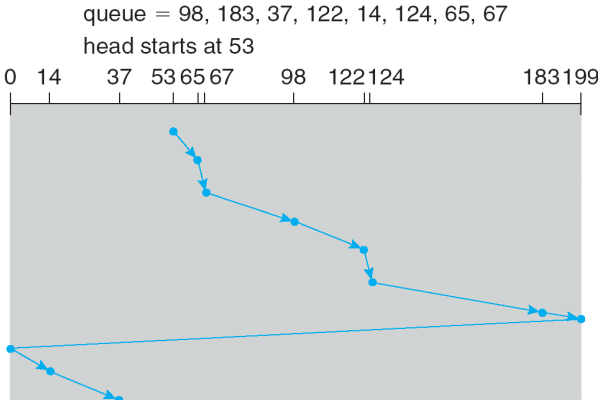
ถ้าการร้องขอที่จะเข้ามาอยู่ในแถวคอยหน้าของหัวอ่าน มันก็จะได้รับบริการเกือบจะทันที ส่วนการร้องขอที่มาถึงหลังหัวอ่าน ก็จะต้องรอจนกระทั่งแขนเคลื่อนไปถึงจุดสิ้นสุดของดิสก์แล้วย้อนกลับมาอีกครั้ง วิธีแบบกวาด (SCAN algorithm) นี้บางครั้งก็เรียกว่า วิธีแบบบันไดเลื่อน (Elevator algorithm) เพราะแขนของดิสก์มีพฤติกรรมคล้ายบันไดเลื่อนในตึก การบริการเริ่มต้นกับการร้องขอทั้งหมดในขาขึ้นและจากนั้นจะบริการการร้องขอย้อนกลับมาอีกทาง

สมมติว่าการกระจายของการร้องขอสำหรับไซลินเดอร์เป็นแบบเดียวกัน พิจารณาความหนาแน่นของการร้องขอเมื่อหัวอ่านมาถึงจุดสิ้นสุดจุดหนึ่งแล้วย้อนทิศทางกลับ ณ จุดนี้ มีการร้องขอเพียงเล็กน้อยที่สัมพันธ์กันที่อยู่หน้าหัวอ่าน ดังนั้นไซลินเดอร์เหล่านี้ก็จะได้รับบริการ แต่ความหนาแน่นที่มากที่สุดของการร้องขออยู่ที่อีกจุดสิ้นสุดหนึ่งของดิสก์ การร้องขอเหล่านี้ก็ต้องรอนานมาก ดังนั้นทำไมจึงไม่คิดที่จะไปทางด้านนั้นก่อน นั่นคือความคิดของวิธีถัดไป

10.2.4 การจัดตารางแบบกวาดเป็นวง (C-SCAN Scheduling)

การกวาดเป็นวง (Circular SCAN เรียกย่อว่า C-SCAN) คือ การเปลี่ยนแปลงของการกวาดซึ่งถูกออกแบบมาเพื่อจัดการกับเวลารอคอยที่มากกว่า 1 รูปแบบ เหมือนกับแบบกวาด แบบกวาดเป็นวงจะเคลื่อนหัวอ่านจากจุดสิ้นสุดจุดหนึ่ง (A) ของดิสก์ไปสู่อีกจุดหนึ่ง (B) มันจะกลับไปสู่จุดเริ่มต้นของดิสก์ในทันที โดยไม่บริการการร้องขอใด ๆ ในขากลับนี้ (ดังภาพที่ 10.5) การจัดตารางแบบกวาดเป็นวงนี้เป็น

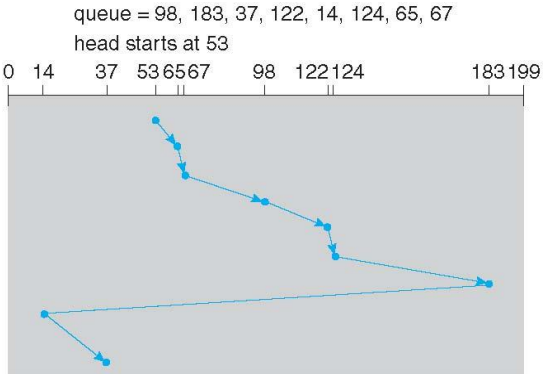
สิ่งจำเป็นสำหรับการจัดการกับไซลินเดอร์แบบวงกลม ซึ่งล้อมรอบจากไซลินเดอร์สุดท้ายไปสู่ไซลินเดอร์แรก



ภาพที่ 10.5 กราฟแสดงการจัดตารางของดิสก์แบบกวาดเป็นวง (C-SCAN)
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.476)

10.2.5 การจัดตารางแบบ LOOK (LOOK Scheduling)

จะเห็นว่า ทั้งแบบกวาดและกวาดเป็นวงจะเคลื่อนแขนดิสก์ไปทั่วทั้งดิสก์ ในทางปฏิบัติไม่มีวิธีไหนที่เอาไปใช้จริงได้ ในความเป็นจริงแขนดิสก์จะไปไกลเพียงแค่การร้องขอสุดท้ายเท่านั้นในแต่ละทิศทาง จากนั้นมันก็จะย้อนกลับไปอีกทางทันทีโดยไม่ต้องไปจุดสิ้นสุดของดิสก์ การทำเช่นนี้เรียกว่า LOOK และ C-LOOK เพราะมันมองการร้องขอก่อนจะเคลื่อนไปยังทิศทางที่ได้ (ดังภาพที่ 10.6)



ภาพที่ 10.6 กราฟแสดงการจัดตารางของดิสก์แบบ C-LOOK
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.477)

10.2.6 การเลือกวิธีการจัดตารางของดิสก์ (Selection of a Disk-Scheduling Algorithm)

วิธี SSTF ดูจะธรรมดาและความเป็นไปได้ตามธรรมชาติ วิธี SCAN และ C-SCAN ใช้ได้ดีสำหรับระบบที่มีภาระหนักบนดิสก์ เพราะมันจะเกิดปัญหาการแช่แข็งได้ยาก สำหรับรายการของการร้องขอ

โดยเฉพาะ มันเป็นไปได้ที่จะกำหนดลำดับที่ดีที่สุด แต่ในแง่ของการคำนวณจำเป็นต้องหาการจัดตารางที่ดีที่สุด ซึ่งอาจจะไม่ประหยัดไปกว่าแบบ SSTF หรือ SCAN

สำหรับดิสก์ยุคใหม่เวลาในการหมุนหัวอ่าน (Rotational latency) มีขนาดใกล้เคียงกับเวลาในการค้นหาเฉลี่ย (Average seek time) แต่เป็นเรื่องยากสำหรับระบบปฏิบัติการที่จะจัดตารางเพื่อปรับปรุงเวลาในการหมุนหัวอ่าน เพราะดิสก์ยุคใหม่ไม่เปิดเผยตำแหน่งทางกายภาพของบล็อกทางตรรกะ ผู้ผลิตดิสก์ได้ช่วยแก้ปัญหานี้โดยใช้วิธีการจัดตารางของดิสก์ในฮาร์ดแวร์ควบคุม (Controller hardware) ที่สร้างไว้ในเครื่องขับดิสก์ (Disk drive) ถ้าระบบปฏิบัติการส่งการร้องขอเป็นกลุ่มมาที่ตัวควบคุม ตัวควบคุมสามารถจัดแถวคอยพวกมันและจากนั้นก็ทำการจัดตาราง เพื่อปรับปรุงทั้งเวลาในการค้นหาและเวลาในการหมุนหัวอ่าน ถ้ามีแต่สมรรถนะในการรับส่งข้อมูลเท่านั้นที่เราสนใจ ระบบปฏิบัติการจะยินดีมากที่จะโอนความรับผิดชอบของการจัดตารางของดิสก์ไปให้ฮาร์ดแวร์ของดิสก์ทำแทน

ในทางปฏิบัติ ระบบปฏิบัติการมีข้อจำกัดอื่น ๆ สำหรับการบริการการร้องขอ เช่น การจัดสรรหน้าตามคำขอ (Demand paging) อาจมีศักดิ์สูงกว่าการรับส่งข้อมูลของโปรแกรมประยุกต์ (Application I/O) และการเขียนก็เป็นเรื่องที่เร่งด่วนกว่าการอ่าน ถ้า Cache กำลังทำงานออกนอกเพจ ที่ว่าง ดังนั้นอาจจะต้องการการรับประกันลำดับของกลุ่มของดิสก์ที่จะเขียน เพื่อทำให้ระบบแฟ้มข้อมูลแข็งแรงเพื่อเผชิญกับการชนของระบบ (System crash) ลองมาพิจารณาดูว่าอะไรจะเกิดขึ้นถ้าระบบปฏิบัติการจัดสรรหน้าของดิสก์ให้แฟ้มข้อมูล แล้วโปรแกรมประยุกต์เขียนข้อมูลลงหน้านั้นก่อนที่ระบบปฏิบัติการจะมีโอกาสปรับปรุงหรือแจ้งให้ดิสก์ทราบว่าหน้านั้นว่าง เพื่อทำให้ความต้องการเหมาะสม ระบบปฏิบัติการอาจจะเลือกที่จะทำการจัดตารางของดิสก์เอง แล้วป้อนการร้องขอให้ตัวควบคุมดิสก์ทีละตัว

10.3 การจัดการดิสก์

บทบาทที่สำคัญหน้าที่หนึ่งของระบบปฏิบัติการคือ หน้าที่สำหรับการจัดการเวลาในการใช้ดิสก์ (Disk scheduling) การฟอร์แมตดิสก์ การจัดการเนื้อที่บนดิสก์ที่เสียหาย การจัดการพื้นที่ว่างบนดิสก์ และการดูแลรักษาดิสก์

10.3.1 การจัดระเบียบดิสก์ (Disk Formatting)

ก่อนที่ดิสก์จะสามารถบรรจุข้อมูลได้ มันต้องถูกแบ่งเป็นเซกเตอร์ซึ่งตัวควบคุมดิสก์สามารถอ่านและเขียนได้ กระบวนการนี้เรียกว่า “การจัดระเบียบระดับต่ำ” (Low-level formatting) หรือ การจัดระเบียบเชิงกายภาพ (Physical formatting) การจัดระเบียบระดับต่ำจะเติมโครงสร้างข้อมูลพิเศษลงไปในแต่ละเซกเตอร์ของดิสก์ โครงสร้างข้อมูลสำหรับเซกเตอร์ประกอบด้วย หัวอ่าน (Header) พื้นที่ของข้อมูลที่มักมีขนาด 512 ไบต์ และส่วนท้ายของหัวอ่าน (Trailer) มีการบรรจุสารสนเทศที่ถูกใช้โดยตัวควบคุมดิสก์ เช่น หมายเลขเซกเตอร์ และ รหัสถูก-ผิด (Error-correcting code : ECC) เมื่อตัวควบคุมเขียนเซกเตอร์ของข้อมูลในระหว่างการรับส่งปกติ ECC จะถูกทำให้ข้อมูลถูกต้องและเป็นข้อมูลล่าสุด ด้วยค่าที่คำนวณจากไบต์ทั้งหมดในพื้นที่ข้อมูล เมื่อเซกเตอร์ถูกอ่าน ECC จะถูกคำนวณใหม่และถูกเปรียบเทียบกับค่าที่เก็บเอาไว้ ถ้าหมายเลขที่ถูกเก็บไว้ และที่ได้จากการคำนวณต่างกัน การไม่เข้าคู่นี้ชี้ให้เห็นว่า พื้นที่ข้อมูลของเซกเตอร์กลายเป็นข้อผิดพลาด และเซกเตอร์ของดิสก์นั้นอาจจะไม่ดี (เกิด Bad sector) ECC คือ รหัสถูก-ผิด เพราะมันบรรจุสารสนเทศที่เพียงพอ ซึ่งถ้ามีข้อมูล 1 หรือ 2 บิตถูกทำให้ผิดพลาด ตัว

ควบคุมจะสามารถแยกได้ว่าบิตไหนถูกเปลี่ยนแปลง และสามารถคำนวณได้ว่า ค่าที่ถูกต้องควร จะเป็นค่าใด กระบวนการของ ECC จะถูกทำอย่างอัตโนมัติโดยตัวควบคุมไม่ว่าเซกเตอร์จะถูกอ่านหรือถูกเขียน

ฮาร์ดดิสก์ส่วนใหญ่มักถูกจัดระเบียบระดับต่ำ จะถูกดำเนินการมาจากโรงงานซึ่งเป็นส่วนหนึ่งของขั้นตอนในการผลิต การจัดระเบียบนี้ทำให้การผลิตสามารถทดสอบดิสก์และเริ่มต้นจับคู่จากหมายเลขบล็อกทางตรรกะเพื่อไม่ให้เกิดข้อบกพร่องของเซกเตอร์บนดิสก์ สำหรับฮาร์ดดิสก์ทั้งหลาย เมื่อตัวควบคุมดิสก์ถูกชี้แนะให้ทำการจัดระเบียบระดับต่ำ ทำให้สามารถบอกได้ว่ามีจำนวนไบต์ของข้อมูลที่ว่างระหว่าง Header และ Trailer ของเซกเตอร์ทั้งหมดเป็นจำนวนเท่าไร มันเป็นไปได้ที่จะเลือกขนาดต่าง ๆ เช่น 256, 512 และ 1024 ไบต์ การจัดระเบียบดิสก์ด้วยเซกเตอร์ขนาดใหญ่ หมายความว่า มีเซกเตอร์เพียงเล็กน้อยในแต่ละแทร็ก และยังหมายถึงว่ามีหัวอ่านและ trailer เพียงเล็กน้อยที่จะถูกเขียนบนแต่ละแทร็ก ดังนั้นจึงเป็นการเพิ่มพื้นที่ว่างให้แก่ข้อมูลของผู้ใช้ ระบบปฏิบัติการบางระบบสามารถจัดการกับเซกเตอร์ขนาด 512 ไบต์ เท่านั้น

เพื่อใช้ดิสก์เก็บแฟ้มข้อมูล ระบบปฏิบัติการยังคงต้องการบันทึกโครงสร้างข้อมูลบนดิสก์ โดยทำงานเป็น 2 ขั้นตอน คือ

1. ทำการแบ่งส่วน (Partition) ดิสก์เป็นหนึ่งหรือหลายกลุ่มของไซลินเดอร์ ระบบปฏิบัติการสามารถจัดการแต่ละส่วนเป็นเหมือนดิสก์ที่แยกจากกัน ตัวอย่างเช่น พาร์ติชั่นหนึ่งอาจจะเก็บโปรแกรมของระบบปฏิบัติการ ในขณะที่อีกพาร์ติชั่นเก็บแฟ้มข้อมูลของผู้ใช้หลังจากทำการแบ่งส่วน

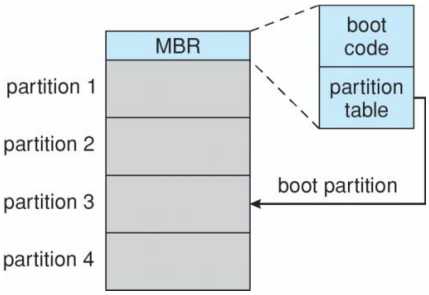
2. การจัดระเบียบเชิงตรรกะ (Logical formatting) หรือ “การทำระบบแฟ้มข้อมูล” (Making a file system) ในขั้นตอนนี้ ระบบปฏิบัติการจะเก็บโครงสร้างข้อมูลของระบบแฟ้มข้อมูลเริ่มแรกไว้บนดิสก์ โครงสร้างข้อมูลเหล่านี้อาจรวมถึงแผนที่ของที่ว่างและการจัดสรรพื้นที่ (FAT) และ ไดเรกทอรี่ว่างเริ่มต้น

10.3.2 บูตบล็อก (Boot Block)

เมื่อคอมพิวเตอร์เริ่มทำงาน (เช่นเปิดเครื่อง หรือบูตเครื่องใหม่) มันจำเป็นต้องมีโปรแกรมเริ่มต้นเพื่อการทำงาน โปรแกรมนี้คือ Bootstrap เป็นโปรแกรมการเริ่มต้นทั้งหมดของระบบ เริ่มจากซีพียู ตัวควบคุมอุปกรณ์ และหน่วยความจำหลัก แล้วจึงเริ่มนำระบบปฏิบัติการเข้าสู่ระบบ โดยมันจะหาแก่น (kernel) ของระบบปฏิบัติการในดิสก์ แล้วนำแก่นนั้นเข้าสู่หน่วยความจำและกระโดดไปสู่ตำแหน่งเริ่มต้นเพื่อเริ่มการทำงานของระบบปฏิบัติการ

คอมพิวเตอร์ส่วนใหญ่โปรแกรม Bootstrap จะถูกเก็บไว้ใน ROM (Read-only memory) เพราะ ROM ไม่จำเป็นต้องใช้ตั้งแต่แรก และมันเป็นตำแหน่งที่แน่นอนคงที่ซึ่งหน่วยประมวลผลสามารถเริ่มทำงานเพื่อเปิดเครื่องหรือ Reset เครื่อง และเพราะ ROM ใช้อ่านได้อย่างเดียว มันจึงไม่สามารถจะติดไวรัสได้ แต่ปัญหาก็คือ การเปลี่ยนแปลงโค๊ดของ Bootstrap จำเป็นต้องเปลี่ยนชิป ROM ด้วย ด้วยเหตุนี้ ระบบส่วนใหญ่จะเก็บโปรแกรม Bootstrap loader เล็ก ๆ ไว้ใน Boot ROM เมื่อต้องการใช้งานก็ค่อยนำโปรแกรม Bootstrap แบบเต็มเข้ามาจากดิสก์ ดังนั้นโปรแกรม Bootstrap แบบเต็มจะสามารถเปลี่ยนแปลงได้ง่าย เราเพียงแต่นำเวอร์ชั่นใหม่เขียนลงบนดิสก์ โปรแกรม Bootstrap แบบเต็มถูกเก็บไว้ในส่วนที่เรียกว่า Boot blocks ที่ตำแหน่งตายตัวบนดิสก์ ดิสก์ที่มีส่วนที่ใช้บูต คือ Boot disk หรือ System disk

ตัวควบคุมดิสก์จะอ่านคำสั่งใน Boot ROM จาก Boot block เข้าสู่หน่วยความจำ (ยังไม่มีตัวควบคุมอุปกรณ์ใด ๆ ถูกนำเข้าหน่วยความจำในตอนนี้) จากนั้นก็เริ่มทำงานตามคำสั่งดังกล่าว โปรแกรม Bootstrap แบบเต็มสามารถที่จะนำระบบปฏิบัติการจากตำแหน่งที่ไม่ตายตัวมาไว้บนดิสก์ได้ แล้วจึงเริ่มให้ระบบปฏิบัติการทำงาน ดังนั้นโปรแกรม Bootstrap แบบเต็มจึงมีขนาดเล็กได้ ตัวอย่างเช่น MS-DOS ใช้บล็อกขนาด 512 ไบต์ สำหรับการบูตเครื่อง (ดังภาพที่ 10.7)



ภาพที่ 10.7 โครงร่างของดิสก์ในระบบ MS-DOS
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.481)

10.3.3 บล็อกเสีย (Bad Block)

ในระบบดิสก์อย่างง่าย เช่น ดิสก์ที่มีตัวควบคุมแบบ IDE เราสามารถจัดการกับบล็อกเสียได้ (โดยผู้ใช้สั่งให้ทำ) เช่น คำสั่ง format ของ MS-DOS เป็นการจัดระเบียบทางตรรกะและจะทำการหาบล็อกเสียบนดิสก์ ถ้าหาเจอจะเขียนค่าพิเศษไว้ในช่องของ FAT ที่ตรงกันเพื่อบอกให้โปรแกรมย่อยต่าง ๆ ไม่ให้ใช้บล็อกนั้น ถ้าบล็อกเกิดเสียในระหว่างการทำงานปกติ โปรแกรมพิเศษ (เช่น Chkdsk) ต้องถูกสั่งให้ทำงานเพื่อค้นหาบล็อกเสียและล็อคบล็อกเหล่านั้นไว้ไม่ให้ใช้ ข้อมูลที่อยู่ในบล็อกเสียก็จะเสียไปโดยปริยาย

สำหรับดิสก์แบบ SCSI สามารถกู้บล็อกเสียคืนได้ โดยตัวควบคุมจะเก็บรายการของบล็อกเสียบนดิสก์ รายการนี้ถูกเก็บมาตั้งแต่ตอนจัดระเบียบดิสก์ระดับต่ำจากโรงงาน และถูกทำให้ทันสมัยตลอดเวลาของการใช้ดิสก์ การจัดระเบียบระดับต่ำจะจัดเซกเตอร์อะไหล่ที่ระบบปฏิบัติการมองไม่เห็นไว้ แล้วตัวควบคุมจะทำการแทนที่เซกเตอร์ทางตรรกะที่เสียแต่ละเซกเตอร์ด้วยเซกเตอร์อะไหล่ วิธีการนี้เรียกว่า การเก็บเซกเตอร์อะไหล่ (Sector sparing) หรือการเปลี่ยนที่ (Forwarding) ตัวอย่างของ การเปลี่ยนแปลงเซกเตอร์เสียอาจเป็นดังนี้

- ระบบปฏิบัติการพยายามอ่านบล็อกทางตรรกะที่ 87
- ตัวควบคุมจะทำการคำนวณรหัสถูก-ผิด (ECC) และพบว่าเซกเตอร์นั้นเสีย มันจะรายงานการค้นพบนี้ไปสู่ระบบปฏิบัติการ
- คราวหน้าเมื่อบูตระบบใหม่ คำสั่งพิเศษจะทำงานเพื่อบอกตัวควบคุม SCSI เพื่อแทนเซกเตอร์ที่เสียด้วยเซกเตอร์อะไหล่
- หลังจากนั้น เมื่อระบบร้องขอบล็อกทางตรรกะที่ 87 การร้องขอนั้นจะถูกแปลไปยังตำแหน่งของเซกเตอร์ที่ถูกแทนที่โดยตัวควบคุม

จะเห็นได้ว่าการเปลี่ยนทิศทางโดยตัวควบคุมอาจจะทำให้การจัดตารางดิสก์ของระบบปฏิบัติการผิดพลาด ด้วยเหตุนี้ดิสก์ส่วนใหญ่ถูกจัดระเบียบ เพื่อเตรียมเซกเตอร์อะไหล่ไว้เล็กน้อยในแต่ละไซลินเดอร์ เมื่อบล็อกเสียถูกพบ ตัวควบคุมจะใช้เซกเตอร์อะไหล่จากไซลินเดอร์เดียวกันถ้าทำได้ อีกทางเลือกหนึ่งของการเก็บเซกเตอร์อะไหล่ ตัวควบคุมสามารถสั่งให้แทนที่บล็อกเสียโดยเซกเตอร์เลื่อน (Sector slipping) ตัวอย่างเช่น สมมติว่าบล็อกทางตรรกะตำแหน่งที่ 17 เสีย และเซกเตอร์อะไหล่แรกที่ใช้ได้อยู่ที่ 202 ดังนั้นเซกเตอร์เลื่อนจะจับตำแหน่งใหม่จาก 17 ไปถึง 202 โดยเคลื่อนตำแหน่งทั้งหมดลงหนึ่งจุด คือ เซกเตอร์ 202 จะถูกใช้เป็นอะไหล่ ดังนั้นเซกเตอร์ 201 จะถูกคัดลอกไป 202 และ 200 จะถูกย้ายไป 201 ไปเรื่อย ๆ จนกระทั่งเซกเตอร์ 18 ถูกคัดลอกลงเซกเตอร์ 19 การเลื่อนเซกเตอร์ด้วยวิธีนี้ ทำให้พื้นที่ของเซกเตอร์ 18 ว่าง ดังนั้นเซกเตอร์ 17 สามารถถูกจับคู่ไปตรงตำแหน่งนั้นแทนได้

การแทนที่ของบล็อกเสียโดยทั่วไปยังไม่เป็นกระบวนการอัตโนมัติเสียทีเดียว เพราะข้อมูลในบล็อกเสียมักจะเสียไป ดังนั้นแฟ้มข้อมูลที่ใช้งานบล็อกนั้นต้องถูกซ่อมแซม (เช่น การกู้คืนจาก Backup เทป) โดยต้องการให้คนเป็นผู้แก้ไข

10.4 การจัดการพื้นที่ที่ใช้ในการสับเปลี่ยน

หน่วยความจำเสมือนใช้พื้นที่ในดิสก์เสมือนส่วนขยายของหน่วยความจำหลัก เพราะว่าการเข้าถึงดิสก์ทำได้ช้ากว่าการเข้าถึงหน่วยความจำ ดังนั้นการใช้พื้นที่ที่ใช้ในการสับเปลี่ยน (Swap space) จะมีผลกระทบมากต่อสมรรถนะของระบบ เป้าหมายหลัก สำหรับการออกแบบและการนำพื้นที่ที่ใช้ ในการสับเปลี่ยนไปใช้ คือ เตรียมอัตรางานที่ได้ต่อหนึ่งหน่วยเวลา (Throughput) ให้ดีที่สุดสำหรับระบบหน่วยความจำเสมือน

10.4.1 การใช้พื้นที่ที่ใช้ในการสับเปลี่ยน (Swap-Space Use)

พื้นที่ที่ใช้ในการสับเปลี่ยน ถูกใช้ได้หลายวิธีโดยหลายระบบปฏิบัติการที่ต่างกัน ขึ้นอยู่กับการใช้อัลกอริทึ่มในการจัดการหน่วยความจำ ตัวอย่างเช่น ระบบที่ใช้การสับเปลี่ยน (Swapping) อาจใช้พื้นที่ที่ใช้ในการสับเปลี่ยน (Swap space) เพื่อเก็บภาพของกระบวนการทั้งหมดไว้ รวมทั้งตอน (Segment) ของรหัสโปรแกรม (Code) และข้อมูลด้วยระบบการแบ่งเป็นหน้า (Paging system) อาจจะง่ายในการเก็บหน้าซึ่งถูกผลักออกจากหน่วยความจำหลัก จำนวนของพื้นที่ที่ใช้ในการสับเปลี่ยน ที่ต้องการของระบบขึ้นอยู่กับจำนวนของหน่วยความจำทางกายภาพ

10.4.2 ตำแหน่งของพื้นที่ที่ใช้ในการสับเปลี่ยน (Swap-Space Location)

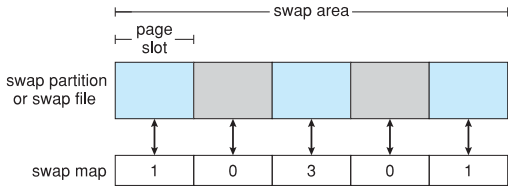
มีพื้นที่อยู่สองพื้นที่ที่เหมาะใช้ในการสับเปลี่ยน คือ พื้นที่ที่ใช้ในการสับเปลี่ยนสามารถถูกตัดออกจากระบบแฟ้มข้อมูลปกติ หรืออยู่ในส่วนของดิสก์ที่แยกออกมา (Separate disk partition) ถ้าการสับเปลี่ยนพื้นที่ทำได้ง่ายกับแฟ้มข้อมูลขนาดใหญ่ภายในระบบแฟ้มข้อมูล โปรแกรมย่อยระบบแฟ้มข้อมูลปกติ (Normal file-system routines) ควรจะถูกใช้เพื่อสร้าง เพื่อระบุชื่อ (ตั้งชื่อ) และเพื่อจัดสรรพื้นที่ให้กับแฟ้มนั้น ๆ วิธีนี้นำไปใช้จริงได้ง่ายแต่ไม่มีประสิทธิภาพ การดำเนินการสำรวจโครงสร้างของไดเรกทอรี่ และโครงสร้างของข้อมูลการจัดสรรดิสก์ต้องใช้เวลา การสูญเสียพื้นที่ย่อยภายนอกทำให้เพิ่มเวลาในการ

สับเปลี่ยนขึ้นมาก เพราะต้องค้นหาหลายครั้งในช่วงของการอ่านหรือเขียนภาพของโพรเซส สามารถปรับปรุงสมรรถนะได้โดยการเก็บตำแหน่งของบล็อกในหน่วยความจำทางกายภาพและโดยการใช้เครื่องมือพิเศษเพื่อจัดสรรบล็อกที่ติด ๆ กัน ทางกายภาพให้กับการสับเปลี่ยนแฟ้มข้อมูล (Swap file) แต่ต้นทุนในการท่องไปในโครงสร้างข้อมูลของระบบแฟ้มข้อมูลยังคงมีอยู่

อีกวิธีหนึ่ง คือ ใช้ส่วนของดิสก์ที่แยกออกมา เพื่อสร้างพื้นที่ ๆ ใช้ในการสับเปลี่ยน จึงไม่มีระบบแฟ้มข้อมูลหรือโครงสร้างของไดเรกทอรี่บนพื้นที่นี้ ตัวจัดการหน่วยเก็บพื้นที่ ๆ จะใช้ในการสับเปลี่ยนจะถูกใช้ในการจัดสรรหรือยึดบล็อกคืนมาสู่ระบบ ตัวจัดการนี้ใช้อัลกอริทึ่มที่ดีที่สุดสำหรับความเร็ว เพื่อให้ได้ประสิทธิภาพการสูญเสียพื้นที่ย่อยภายในอาจจะเพิ่มขึ้น แต่ก็พอรับได้เพราะข้อมูลในพื้นที่ ๆ ใช้ในการสับเปลี่ยนมักจะอยู่ในช่วงสั้นกว่าแฟ้มข้อมูลในระบบแฟ้มข้อมูล และพื้นที่ ๆ ใช้ในการสับเปลี่ยนมักจะถูกเข้าถึงอยู่บ่อยครั้ง ข้อเสียของวิธีนี้คือสร้างพื้นที่ ๆ ใช้ในการสับเปลี่ยนอย่างคงที่เอาไว้มากในระหว่างการแบ่งส่วนของดิสก์ การเพิ่มพื้นที่ ๆ ใช้ในการสับเปลี่ยนให้มากขึ้นทำได้โดยการแบ่งส่วนของดิสก์ใหม่อีกครั้ง (ซึ่งเกี่ยวกับการย้ายหรือการทำลาย และการกู้คืนส่วนของระบบแฟ้มข้อมูลอื่นจากการ Backup) หรือโดยการเพิ่มพื้นที่ที่ใช้ในการสับเปลี่ยนจากพื้นที่อื่น ๆ

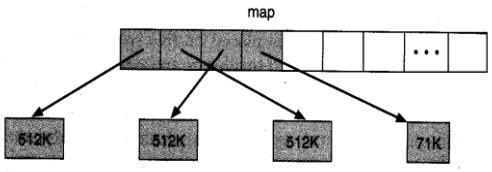
10.4.3 การจัดการพื้นที่ที่ใช้ในการสับเปลี่ยน (Swap-Space Management)

พื้นที่ที่ใช้ในการสับเปลี่ยนถูกจัดสรรให้แก่โพรเซส เมื่อโพรเซสเริ่มต้น พื้นที่จะถูกจัดสรรให้เพื่อเก็บโปรแกรมอย่างเพียงพอ โดยเก็บหน้าของข้อความ (Text page) หรือตอนของข้อความ (Text segment) และตอนของข้อมูล (Data segment) ของโพรเซส ก่อนการจัดสรรพื้นที่ต้องการทั้งหมด ด้วยวิธีนี้เป็นการป้องกันโพรเซสจากการทำงานนอกพื้นที่ ๆ ใช้ในการสับเปลี่ยนในขณะที่กำลังทำงาน (Execute) เมื่อโพรเซสเริ่มต้นทำงาน ข้อความจะถูกแบ่งเป็นหน้าจากระบบแฟ้มข้อมูล หน้าเหล่านี้จะถูกเขียนเมื่อจำเป็นและถูกอ่านย้อนหลังจากตรงนั้น ดังนั้นแฟ้มข้อมูลจะถูกพิจารณาเพียงครั้งเดียว



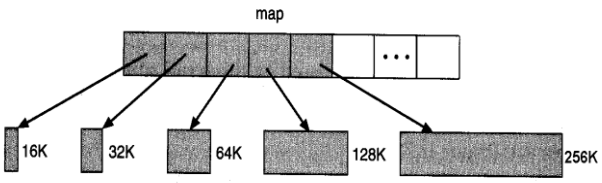
ภาพที่ 10.8 แสดงแผนที่ของการสับเปลี่ยนตอนของข้อความ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.484)

สำหรับในแต่ละหน้าของข้อความหน้าจากตอนของข้อมูลจะถูกอ่านจากระบบแฟ้มข้อมูล หรือถูกสร้างขึ้น (ถ้ายังไม่มี) และถูกเขียนที่พื้นที่ ๆ ใช้ในการสับเปลี่ยน และย้อนกลับไปยังหน้าที่ต้องการ วิธีที่ดีที่สุดวิธีหนึ่ง (เช่น เมื่อผู้ใช้ 2 คน ทำงาน Editor ตัวเดียวกัน) คือ โพรเซสในหน้าของข้อความที่เหมือนกันควรร่วมกันใช้หน้าเหล่านี้ ทั้งในหน่วยความจำทางตรรกะและในพื้นที่ ๆ ใช้ในการสับเปลี่ยนแก่น (Kernel) จะใช้แผนที่ในการสับเปลี่ยน (Swap map) เพื่อตามรอยพื้นที่ที่ใช้ในการสับเปลี่ยนตอนของข้อความจะมีขนาดคงที่ ดังนั้นพื้นที่ที่ใช้ในการสับเปลี่ยนจะถูกจัดสรรให้ก้อนละ 512K ยกเว้นก้อนสุดท้าย จะเก็บส่วนที่เหลือ (อาจไม่ถึง 512K) ดังแสดงในภาพที่ 10.9



ภาพที่ 10.9 แสดงแผนที่ของการสับเปลี่ยนตอนของข้อความ

แผนที่การสับเปลี่ยนตอนของข้อมูลค่อนข้างซับซ้อน เพราะตอนของข้อมูลสามารถเพิ่มขึ้นได้ตลอดเวลา แผนที่ดังกล่าวมีขนาดคงที่แต่เก็บตำแหน่งที่ใช้ในการสับเปลี่ยนที่เป็นบล็อกที่มีขนาดไม่คงที่ ให้ดัชนี i คือ บล็อกที่แผนที่ที่ใช้ในการสับเปลี่ยนชี้อยู่มีขนาด 2^i x 16 K โดยมากที่สุดได้ 2 เมกกะไบต์ โครงสร้างของข้อมูลนี้แสดงได้ดังภาพที่ 10.10



ภาพที่ 10.10 แสดงแผนที่ที่ใช้ในการสับเปลี่ยนตอนของข้อมูล

เมื่อโพรเซสพยายามเพิ่มตอนของข้อมูลก่อนบล็อกสุดท้ายในพื้นที่ที่ใช้ในการสับเปลี่ยน ระบบปฏิบัติการจะจัดบล็อกให้อีกบล็อกหนึ่ง ที่มีขนาดเป็น 2 เท่าของบล็อกก่อนหน้านี้ ผลลัพธ์ของวิธีการนี้ โพรเซสเล็กก็จะใช้บล็อกเล็กตามไปด้วย นั่นเท่ากับเป็นการลดการเสียพื้นที่ บล็อกของโพรเซสขนาดใหญ่ก็จะหาพบได้รวดเร็ว ดังนั้นแผนที่ที่ใช้ในการสับเปลี่ยนจึงมีขนาดเล็กตามไปด้วย ขนาดของบล็อกที่เล็กที่สุดและใหญ่ที่สุดไม่คงที่ และสามารถเปลี่ยนแปลงได้โดยการบูตเครื่องใหม่

10.5 ความน่าเชื่อถือของดิสก์

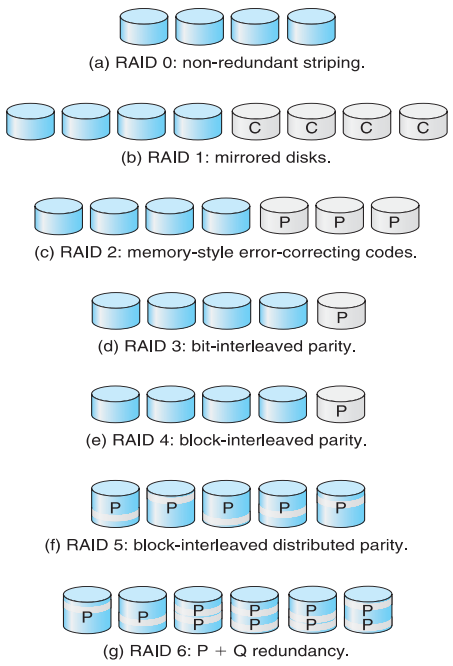
ในระบบคอมพิวเตอร์ดิสก์ยังคงมีอัตราความผิดพลาดสูง และความผิดพลาดเหล่านั้นทำให้ข้อมูลสูญหายและเสียเวลาการกู้คืน อาจต้องใช้เวลาหลายชั่วโมง ในการปรับปรุงเทคนิคในการใช้ดิสก์หลาย ๆ ทางเกี่ยวข้องกับการใช้ดิสก์หลายตัวทำงานประสานกันเพื่อเป็นการปรับปรุงในด้านความเร็ว การแกะดิสก์ (Disk striping) เราใช้กลุ่มของดิสก์เป็นเหมือนหน่วยเก็บข้อมูล 1 หน่วย บล็อกของข้อมูลแต่ละบล็อกถูกแบ่งเป็นบล็อกย่อยหลาย ๆ บล็อก ซึ่งบล็อกย่อยจะถูกเก็บในแต่ละดิสก์ เวลาที่ต้องการเพื่อใช้ในการย้ายบล็อกไปสู่หน่วยความจำจะลดลงอย่างเหลือเชื่อ เพราะดิสก์ทุกตัวโอนย้ายบล็อกย่อยของพวกมันแบบขนาน ถ้าดิสก์มีการหมุนพร้อมกัน (Synchronized) สมรรถนะจะดีขึ้น เพราะดิสก์ทั้งหมดจะพร้อมที่จะโอนย้ายบล็อกย่อยของมันในเวลาเดียวกัน

การจัดการนี้มักเรียกว่า อาร์เรย์ที่เหลือเฟือของดิสก์อิสระ (Redundant array of independent disk : RAID) วิธีนี้เป็นการเพิ่มความน่าเชื่อถือของระบบหน่วยเก็บข้อมูลโดยการเก็บข้อมูลที่เหลือเฟือ การจัดการ RAID ที่ง่ายที่สุด เรียกว่า การสำรองข้อมูล (Mirroring หรือ shadowing) โดยเก็บสำเนาของแต่ละดิสก์ วิธีนี้คุ้มค่าเพราะดิสก์หลายตัวถูกใช้เก็บข้อมูลเดียวกัน 2 ครั้ง แต่ในทางกลับกันวิธีนี้จะเร็วเป็น 2 เท่าเมื่อเป็นการอ่าน เพราะครึ่งหนึ่งของการร้องขอที่จะอ่านจะถูกส่งไปยังแต่ละดิสก์

การจัดการ RAID เรียกว่า ความตรวจสอบบล็อกขาออก (Block interleaved parity) โดยใช้พื้นที่ของดิสก์เพียงเล็กน้อยในการเก็บบล็อกตรวจสอบ (Parity block) ตัวอย่างเช่น สมมติว่ามีดิสก์ 9 ตัวในอาร์เรย์ โดยบล็อกของข้อมูลทุก ๆ 8 ตัว ที่ถูกเก็บในอาร์เรย์จะมีอยู่ 1 บล็อกที่เป็นบล็อกตรวจสอบที่ถูกเก็บไปด้วย ตำแหน่งของบิตแต่ละบิตในบล็อกตรวจสอบนี้ควรจะเก็บค่าสำหรับตำแหน่งของบิตที่ตรงกันในแต่ละบล็อกของข้อมูลทั้ง 8 เหมือนกับการคำนวณบิตตรวจสอบที่ 9 ในหน่วยความจำหลักสำหรับแต่ละ 8 บิต-ไบต์ ถ้าบล็อกของดิสก์หนึ่งเสีย บิตของข้อมูลทั้งหมดต้องถูกลบ แต่ยังสามารถคำนวณใหม่จากบล็อกของข้อมูลบล็อกอื่นบวกกับบล็อกตรวจสอบ ดังนั้นดิสก์เสียตัวเดียวจึงไม่ทำให้ข้อมูลเสียหาย

10.5.1 ระดับของ RAID

Data striping คือการแบ่งข้อมูลออกเป็นส่วน ๆ แล้วนำแต่ละส่วนไปเก็บในฮาร์ดดิสก์แต่ละตัว การทำ Striping นี้จะช่วยให้การอ่าน หรือเขียนข้อมูลใน Disk array มีประสิทธิภาพมากขึ้น เพราะแต่ละไฟล์จะถูกแบ่งเป็นส่วน ๆ กระจายไปเก็บในส่วนที่ต่างกันของฮาร์ดดิสก์หลายตัว โดยฮาร์ดดิสก์เหล่านั้นทำงานไปด้วยกันแบบขนาน (Parallel) จึงทำให้การเข้าถึงข้อมูลนั้นเร็วกว่าฮาร์ดดิสก์แบบตัวเดียวอย่างแน่นอน เราสามารถอธิบาย Raid ประเภทต่าง ๆ ดังนี้ (ภาพที่ 10.11 แสดงระดับของ RAID)



ภาพที่ 10.11 แสดง RAID ระดับต่าง ๆ
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.487)

1. RAID 0 คือการเอาฮาร์ดดิสก์มากกว่า 1 ตัวมาต่อร่วมกันในลักษณะ Non-redundant ซึ่ง RAID 0 นี้มีจุดประสงค์เพื่อที่จะเพิ่มความเร็วในการอ่าน/เขียนข้อมูลฮาร์ดดิสก์โดยตรง ไม่มีการเก็บข้อมูลสำรอง ดังนั้นถ้าฮาร์ดดิสก์ตัวใดตัวหนึ่งเกิดเสียหาย ก็จะส่งผลให้ข้อมูลทั้งหมดไม่สามารถใช้งานได้ทันทีดังแสดงภาพที่ 10.11(a) จะเห็นว่าข้อมูลจะถูกแบ่งไปเก็บที่ฮาร์ดดิสก์ทั้ง 3 ตัว จุดเด่นของ RAID 0 คือความเร็วในการเข้าถึงข้อมูล แต่ข้อเสียก็คือหากฮาร์ดดิสก์ตัวใดตัวหนึ่งเสียหาย จะส่งผลกับข้อมูลทั้งระบบทันที

2. RAID 1 มีอีกชื่อหนึ่งว่า Disk mirroring จะประกอบไปด้วยฮาร์ดดิสก์ 2 ตัวที่เก็บข้อมูลเหมือนกันทุกประการ เสมือนการสำรองข้อมูล หากฮาร์ดดิสก์ตัวใดตัวหนึ่งเกิดเสียหาย ระบบยังสามารถดึงข้อมูลจากฮาร์ดดิสก์อีกตัวหนึ่งมาใช้งานได้ตามปกติดังแสดงภาพที่ 10.11(b) จุดเด่นของ RAID 1 คือความปลอดภัยของข้อมูล ไม่เน้นเรื่องประสิทธิภาพและความเร็วเหมือนอย่าง RAID 0 แม้ว่าประสิทธิภาพในการอ่านข้อมูลของ RAID 1 จะสูงขึ้นก็ตาม

3. RAID 2 ข้อมูลทั้งหมดจะถูกตัดแบ่งเพื่อจัดเก็บลงฮาร์ดดิสก์แต่ละตัวใน Disk array โดยจะมีฮาร์ดดิสก์ตัวหนึ่งเก็บข้อมูลที่ใช้ตรวจสอบและแก้ไขข้อผิดพลาด (Error checking and correcting : ECC) ซึ่งเป็นการลดเปอร์เซ็นต์ที่ข้อมูลจะเสียหายหรือสูญเสียไป เมื่อมีการส่งข้อมูลไปบันทึกใน Disk array จะเห็นได้ว่ามีฮาร์ดดิสก์ที่เอาไว้เก็บค่า ECC โดยเฉพาะ ถ้าเกิดการปรากฏว่าฮาร์ดดิสก์ตัวใดตัวหนึ่งเสียหาย ระบบก็จะสามารถสร้างข้อมูลทั้งหมดในฮาร์ดดิสก์ตัวนั้นขึ้นมาได้ใหม่ โดยอาศัยข้อมูลจากฮาร์ดดิสก์ ตัวอื่น ๆ และจากค่า ECC ที่เก็บเอาไว้ ซึ่งการทำ ECC นี้ส่งผลให้ฮาร์ดดิสก์ ทั้งระบบต้องทำงานค่อนข้างมากทีเดียว และ RAID 2 นั้นจะเห็นได้ว่าต้องใช้ฮาร์ดดิสก์จำนวนมากในการเก็บค่า ECC ซึ่งทำให้ค่อนข้างสิ้นเปลือง ดังแสดงภาพที่ 10.11(c)

4. RAID 3 มีลักษณะที่คล้ายกับ RAID 2 แต่แทนที่จะตัดแบ่งข้อมูลในระดับบิตเหมือน RAID 2 ก็จะตัดเก็บข้อมูลในระดับ Byte แทนและการตรวจสอบและแก้ไขข้อผิดพลาดของข้อมูล จะใช้ Parity แทนที่จะเป็น ECC ทำให้ RAID 3 มีความสามารถในการอ่านและเขียนข้อมูลได้อย่างรวดเร็ว เพราะมีการต่อฮาร์ดดิสก์แต่ละตัวแบบ stripe และใช้ฮาร์ดดิสก์ที่เก็บ Parity เพียงแค่ตัวเดียวเท่านั้นดังแสดงภาพที่ 10.11(d) แต่ถ้านำ RAID 3 ไปใช้ในงานที่มีการส่งผ่านข้อมูลในจำนวนที่น้อย ๆ ซึ่ง RAID 3 ต้องกระจายข้อมูลไปทั่วทั้งฮาร์ดดิสก์จะทำให้เกิดปัญหาที่เรียกว่า คอขวด ขึ้นกับฮาร์ดดิสก์ที่เก็บ Parity ไม่ว่าข้อมูลจะมีขนาดใหญ่ขนาดไหน RAID 3 ต้องเสียเวลาไปสร้างส่วน Parity ทั้งสิ้น ยิ่งข้อมูลมีขนาดเล็ก ๆ แต่ Parity ต้องสร้างขึ้นตลอด ทำให้ข้อมูลถูกจัดเก็บเสร็จก่อนการสร้าง Parity ทั้งระบบต้องมารอให้สร้าง Parity เสร็จก่อน จึงจะทำงานต่อไปได้นั่นเอง RAID 3 เหมาะสำหรับใช้ในงานที่มีการส่งข้อมูลจำนวนมาก ๆ เช่น งานตัดต่อ Video เป็นต้น

5. RAID 4 มีลักษณะโดยรวมเหมือนกับ RAID 3 ทุกประการ ยกเว้นเรื่องการตัดแบ่งข้อมูลที่ทำในระดับ Block แทนที่จะเป็น Bit หรือ Byte ดังแสดงภาพที่ 10.11(e) ซึ่งทำให้การอ่านข้อมูลแบบ Random ทำได้รวดเร็วกว่า อย่างไรก็ตามแต่ยังคงมีปัญหาคอขวดที่กล่าวใน RAID 3

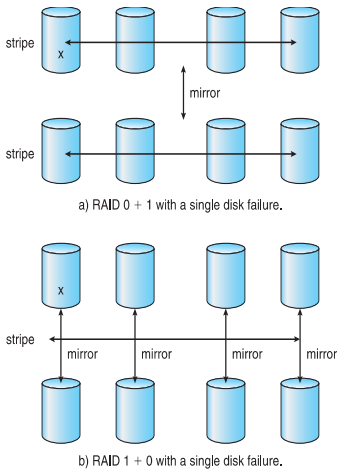
6. RAID 5 มีการตัดแบ่งข้อมูลในระดับ Block เช่นเดียวกับ RAID 4 แต่จะไม่ทำการแยกฮาร์ดดิสก์ตัวใดตัวหนึ่งเพื่อเก็บ Parity ในการเก็บ Parity ของ RAID 5 นั้น จะทำการกระจาย Parity ไปยังฮาร์ดดิสก์ทุกตัวดังแสดงในภาพที่ 10.11(f) โดยปะปนไปกับข้อมูลปกติ จึงช่วยลดปัญหาคอขวด ซึ่งเป็นปัญหาที่สำคัญใน RAID 3 และ RAID 4 คุณสมบัติอีกอันหนึ่งที่น่าสนใจอย่างมากของ RAID 5 คือ

เทคโนโลยี Hot Swap คือ สามารถทำการเปลี่ยนฮาร์ดดิสก์ในกรณีที่เกิดปัญหาได้ ในขณะที่ระบบยังทำงานอยู่ เหมาะสำหรับงาน Server ต่าง ๆ ที่ต้องทำงานต่อเนื่อง

10.5.2 การนำ RAID มาใช้ร่วมกัน

การใช้ RAID ในระดับเดียวนั้นอาจจะไม่สามารถติดตั้งระบบปฏิบัติการต่าง ๆ ได้ครบตามความต้องการ ดังนั้น เพื่อให้ได้เครื่องมือในการทำงานครบถ้วนนั้น การนำข้อดีของ RAID ในแต่ละระดับมารวมกันเพื่อใช้งาน ซึ่งนั่นหมายถึง ถ้าเราสามารถรวมข้อดีของ RAID ในแต่ละลำดับขั้นไว้ด้วยกัน อาจทำให้การทำงานมีประสิทธิภาพมากขึ้นก็เป็นได้ และเป็นเหตุผลของการกำเนิดระดับตัวใหม่ตัวนี้ของ RAID การใช้งานของ RAID ในระดับใหม่นั้นเกิดจากการนำเอาข้อดีของ RAID ในแต่ละระดับมารวมกันซึ่งประสิทธิภาพของการทำงานย่อมเพิ่มขึ้น

นอกจากนี้ลองมาดูการรวมตัวของ RAID ในระดับต่าง ๆ RAID 0 เป็นระดับที่มีประสิทธิภาพการทำงานที่ดีที่สุด และเป็นตัวเดียวที่สามารถนำมาร่วมตัวกับระดับอื่นได้มากที่สุด เพราะไม่ใช่ว่าทุกระดับของ RAID จะสามารถนำมารวมกันได้ โดยทั่วไประดับที่นิยมนำมารวมกันของ RAID ก็จะมี RAID 0+1 และ RAID 1+0 ซึ่งความแตกต่างระหว่าง 0+1 และ 1+0 จะเห็นได้จากชื่อที่เรียกใช้งาน การเลือกใช้ตัวใดนั้นคงต้องเลือกเอาจาก RAID ตัวที่มีข้อผิดพลาดเกิดขึ้นน้อยที่สุดเมื่อนำไปใช้งานจริงการใช้งานของ RAID ทั้งสองตัวนี้ ต้องเตรียมฮาร์ดไดรฟ์เอาไว้ 4 ตัว (แสดงในภาพที่ 10.12 RAID 0 + 1 และ 1 + 0)



ภาพที่ 10.12 แสดง RAID 0 + 1 และ 1 + 0
ที่มา: Abraham, S. Peter, B. G., & Greg, G. Operating system concepts 9th ed. (2013, p.490)

จากนี้จะเริ่มมาดูการทำงานของ RAID 0+1 กันก่อน การรวมกันระหว่าง RAID 0 เพื่อให้ได้ประสิทธิภาพการทำงาน และ RAID 1 เพื่อให้เกิดความผิดพลาดในการปฏิบัติการให้น้อยที่สุดนั้น สำหรับระดับที่มีการรวมนี้ไปแล้ว เมื่อการแยกส่วนจัดเก็บข้อมูลถูกนำมาไว้ในส่วนข้อมูลสำรอง ซึ่งหมายถึงต้องมีฮาร์ดไดรฟ์เพื่อปฏิบัติการนี้ถึง 8 ตัว จึงจะสามารถแยกฮาร์ดไดรฟ์แต่ละตัวนั้นเป็น 2 หมวด ในทุก ๆ 4 ไดรฟ์ และนำ RAID 0 มาประยุกต์ใช้งานร่วมกันไป ซึ่งจะทำให้มีการแยกเก็บข้อมูลไว้ด้วยกัน 2 หมวด เมื่อประยุกต์ RAID 1 เป็นการแยกเก็บข้อมูลแบบ 2 หมวดแล้ว และมีการสำรองข้อมูลไว้ 1 หมวดไว้ในส่วนที่

เหลือ ถ้าฮาร์ดไดรฟ์ตัวที่ถูกแยกส่วนไว้ เกิดความเสียหายหรือสูญหาย และหมวดอื่นที่แยกเก็บข้อมูลไว้ไม่ได้มารองรับข้อมูลในส่วนนี้ การเก็บข้อมูลสำรองเอาไว้จะทำให้ความเสียหายนั้นน้อยลง RAID 1+0 เป็นการประยุกต์ใช้โดยการนำ RAID 1 มาใช้ก่อน จากนั้นจึงนำ RAID 0 มาไว้ในไดรฟ์ ในการที่จะประยุกต์ใช้ RAID 1 นั้นผู้ใช้งานต้องแบ่งไดรฟ์เอาไว้ด้วยกัน 8 ไดรฟ์ทั้งหมด 4 ชุด ซึ่งแต่ละชุดจะมีทั้งหมด 2 ไดรฟ์ โดยในแต่ละชุดนั้นก็คือการสำรองข้อมูลและมีการประยุกต์ข้อมูลเพื่อนำมาใช้งาน จากนั้น จึงประยุกต์ RAID 0 มารวมกันในส่วนนี้จะมีการจำแนกข้อมูลเอาไว้ 4 ชุดด้วยกัน ซึ่งโดยหลัก ๆ แล้วจะมีการจำแนกข้อมูลตามจำนวนของชุดข้อมูลสำรอง

การรวมกันเช่นนี้ จะช่วยลดความผิดพลาดที่อาจเกิดขึ้นได้ดีกว่าการรวมตัวของ RAID 0+1 ซึ่งการทำงานเช่นนี้สำหรับหนึ่งไดรฟ์ภายในชุดสำรองข้อมูลจะมีการปฏิบัติงานได้ดี และส่วนที่ถูกจัดหมวดนั้นยังคงทำหน้าที่ได้ดีเช่นเดิม ซึ่งในทางปฏิบัติแล้วคุณสามารถเก็บข้อมูลได้ถึงครึ่งหนึ่งของไดรฟ์ หากว่าไดรฟ์นั้นเกิดความผิดพลาดในการใช้งาน วิธีการเช่นนี้จะทำให้คุณไม่ต้องเสียข้อมูลไปทั้งหมดเพราะซึ่งวิธีการนี้จะไม่สามารถนำไปใช้ได้กับระดับ RAID 0+1 เพราะมีการแบ่งไดรฟ์ออกเป็นสองส่วนเท่านั้น ความนิยมในตัว RAID 0+1 และ 1+0 นั้นค่อนข้างที่จะนิยมใช้พอกันซึ่งขึ้นอยู่กับความต้องการใช้งานของผู้ใช้มากกว่า เมื่อราคาฮาร์ดไดรฟ์ถูกลงและมีจำนวนเพิ่มขึ้น อาจจะเหลือเการใช้ฮาร์ดไดรฟ์เพียง 4 ตัว

ในการปฏิบัติงานจึงไม่ใช่เรื่องน่าแปลกแต่อย่างใดในปัจจุบัน อย่างไรก็ตามผู้ใช้งานยังคงต้องเกิดความเสียหายในการจัดเก็บถึง 50 เปอร์เซ็นต์ของพื้นที่ใช้งานอยู่ดีเมื่อคุณต้องทำงาน โดยมี การสำรองข้อมูลบริษัทผู้ผลิตระบบปฏิบัติการ และเซิร์ฟเวอร์นั้นมักเสียพื้นที่ส่วนหนึ่งบนเซิร์ฟเวอร์เพื่อทำให้การจัดเก็บข้อมูลเกิดประสิทธิภาพมากที่สุด และเกิดความเสียหายน้อยที่สุด ซึ่งการรวมตัวของ RAID ในระดับต่าง ๆ จะหมายรวมไปถึงการรวมตัวของ RAID ในระดับอื่นด้วย เช่น RAID 0+3,3+0.0+5.5+0,1+5 และ 5+1 ซึ่งระดับต่าง ๆ เหล่านี้เมื่อถูกนำมาใช้งานก็ต้องเป็นการใช้งานร่วมกับฮาร์ดแวร์ที่มีราคาแพงด้วย และใช่ว่าจะมีการนำระดับดังกล่าวมาใช้งานได้ทั้งหมด

10.5.3 ประโยชน์ของ RAID

กล่าวได้ว่าประสิทธิภาพของ RAID จะขึ้นอยู่กับแอพพลิเคชันที่ใช้กับระดับของ RAID ด้วย แต่โดยทั่วไปแล้ว RAID จะถูกใช้เพื่อป้องกันความผิดพลาดในการเก็บข้อมูล และเพิ่มระดับการจัดเก็บด้วยที่สำคัญป้องกันการจัดเก็บข้อมูลในกรณีที่ฮาร์ดไดรฟ์เกิดทำงานผิดพลาด จะเห็นว่าคุณสมบัตินี้สำคัญต่อการบริหารข้อมูลที่สำคัญขององค์กร หรือแม้กระทั่งส่วนบุคคลด้วย นอกจากนั้นจะทำให้ทุกคนไม่ต้องกังวลถึงการสูญเสียพื้นที่หลายกิกะไบต์ให้กับไฟล์ประเภทต่าง ๆ รวมทั้งการกำจัดข้อมูลที่ไม่จำเป็นออกไปของ RAID ทำให้ระบบฐานข้อมูลของคุณนั้นดีขึ้น RAID ระบบเดียวที่ไม่มีคุณสมบัติกำจัดข้อมูลที่ไม่จำเป็นรวมทั้งระบบป้องกันข้อมูลผิดพลาดก็คือ RAID 0

นอกจากนี้ RAID ยังสามารถขยายหน่วยความจำ โดยการผนวกหลาย ๆ ไดรฟ์เข้าด้วยกัน แต่ประสิทธิผลนั้น ขึ้นอยู่กับระดับของ RAID ที่ใช้ด้วย ซึ่งโดยปกติแล้วระดับที่ใช้สำหรับการสำรองการเก็บข้อมูลนั้น ต้องการหน่วยความจำสูงขึ้นเป็นสองเท่า และเหตุผลสุดท้ายที่ควรมาใช้ RAID ก็คือการเพิ่มประสิทธิภาพในการเก็บข้อมูลให้ดีขึ้น เพราะในแต่ละระดับของ RAID ที่เลือกใช้ ความแตกต่างของประสิทธิภาพก็จะแตกต่างกันไป โดยเฉพาะการใช้งานกับแอพพลิเคชันที่ต้องการความเร็วสูงนั้น RAID น่าจะเหมาะสมที่สุด

10.6 การใช้งานหน่วยเก็บข้อมูลชนิดคงที่

โดยนิยามแล้ว ข้อมูลที่อยู่ในหน่วยเก็บข้อมูลชนิดคงที่จะไม่มีทางสูญหายไปไหน การนำหน่วยเก็บข้อมูลไปใช้ เราจำเป็นต้องคัดลอกข้อมูลที่ต้องการไปไว้ยังอุปกรณ์ที่ใช้เก็บข้อมูลหลาย ๆ ที่ (อาจเก็บในดิสก์ด้วย ในเทปด้วย) ด้วยโหมดที่เป็นอิสระจากความล้มเหลว เราต้องการวิธีที่จะทำให้ข้อมูลทันสมัย (Update) ที่รับประกันได้ว่าความผิดพลาดในระหว่างการทำข้อมูลให้ทันสมัยจะไม่ทำให้ข้อมูลเสียหาย และเมื่อเราทำการกู้คืนจากที่ผิดพลาด เราต้องสามารถทำให้ข้อมูลทั้งหมดมีค่าที่ถูกต้อง ถ้าเกิดมีการล้มเหลวอีกในระหว่างการกู้คืนจะต้องทำอย่างไร ดังนั้นดิสก์จะทำงานจนได้ผลลัพธ์หนึ่งในสามข้อนี้ คือ

1. สำเร็จอย่างสมบูรณ์แบบ (Successful completion) ข้อมูลถูกเขียนลงดิสก์ได้อย่างถูกต้อง
2. ล้มเหลวบางส่วน (Partial failure) ความล้มเหลวเกิดขึ้นระหว่างการโอนย้ายข้อมูล ดังนั้นเซกเตอร์บางส่วนจะถูกเขียนด้วยข้อมูลใหม่ และเซกเตอร์ที่ถูกเขียนในระหว่างเกิดการล้มเหลวอาจจะผิดพลาด
3. ล้มเหลวทั้งหมด (Total failure) ความล้มเหลวเกิดขึ้น ก่อนที่ดิสก์จะเริ่มเขียน ดังนั้นค่าของข้อมูลบนดิสก์ก็จะเหมือนเดิมก่อนที่จะเกิดการเขียนใด ๆ

ถ้าความล้มเหลวเกิดในระหว่างการเขียนบล็อก เมื่อระบบตรวจพบแล้วจะทำการกู้บล็อกนั้นคืน ระบบต้องเก็บบล็อกทางกายภาพ 2 บล็อก ไว้ให้บล็อกทางตรรกะแต่ละบล็อก ผลลัพธ์ที่ได้มีดังนี้

- เขียนข้อมูลลงบนบล็อกทางกายภาพบล็อกแรก
- เมื่อเขียนบล็อกแรกเรียบร้อย ก็เขียนข้อมูลเดียวกันลงยังบล็อกทางกายภาพอีกบล็อก
- ประกาศว่าการทำงานเสร็จสิ้นหลังจากการเขียนบล็อกที่สองสำเร็จเท่านั้น

ในระหว่างการกู้ระบบจากความล้มเหลว บล็อกทางกายภาพแต่ละคู่จะถูกทดสอบ ถ้าทั้ง 2 บล็อกเหมือนกันและไม่มีการพบข้อผิดพลาดก็ไม่ต้องทำอะไร ถ้ามีบล็อกหนึ่งพบว่ามีข้อผิดพลาดจะทำการแทนที่เนื้อหาในบล็อกนั้นด้วยข้อมูลของอีกบล็อกหนึ่ง ถ้าทั้งสองบล็อกไม่พบข้อผิดพลาดแต่เนื้อหาต่างกัน จะเอาเนื้อหาของบล็อกที่สองเข้าไปเก็บแทนที่ในบล็อกที่หนึ่ง กระบวนการในการกู้คืนนี้ประกันได้ว่า การเขียนข้อมูลลงในหน่วยเก็บข้อมูลต้องเสร็จสมบูรณ์หรือไม่ผลลัพธ์ที่ได้ต้องไม่เปลี่ยนแปลง

10.7 สรุป

ดิสก์ไดรฟ์เป็นอุปกรณ์ที่เป็นประเภทหน่วยเก็บข้อมูลสำรอง (Secondary-storage I/O) สำหรับเครื่องคอมพิวเตอร์ โดยปกติอุปกรณ์หน่วยเก็บข้อมูลสำรอง จะเป็น Magnetic disk หรือไม่ก็ Magnetic tape ดิสก์ไดรฟ์สมัยใหม่จะถูกสร้างขึ้นด้วย One-dimensional array of logical disk block ขนาดใหญ่ ดิสก์สามารถติดเข้ากับระบบคอมพิวเตอร์ได้โดย 1 ใน 2 ทาง เชื่อมต่ออุปกรณ์เข้ากับคอมพิวเตอร์โดยตรงผ่านทางช่องพอร์ตอินพุตหรือเอาท์พุต ความต้องการข้อมูลสำหรับการอินพุตและเอาท์พุตของข้อมูลในดิกส์ จะถูกกำหนดโดยระบบแฟ้มข้อมูลและระบบความจำเสมือน ความต้องการ แต่ละอย่างจะระบุที่อยู่

บนดิสก์เพื่อใช้เป็นตัวอ้างอิง ซึ่งอยู่ในรูปแบบของ Logical block number disk-scheduling algorithm สามารถที่จะพัฒนา Effective bandwidth, ระยะเวลาตอบสนองเฉลี่ย (Average response time) และความแตกต่างของเวลาในการตอบสนอง (Variance in response time) ให้ดีขึ้นได้

ในระบบคอมพิวเตอร์ความน่าเชื่อถือในการเก็บรักษาข้อมูลในดิกส์ ให้มีความถูกต้องแม่นยำและการกู้คืนระบบมีความสำคัญอย่างยิ่ง มีการปรับปรุงเทคนิคในการใช้ดิสก์หลาย ๆ ทางเกี่ยวข้องกับการใช้ดิสก์หลายตัวทำงานประสานกันเพื่อเป็นการปรับปรุงในด้านความเร็ว มีการใช้กลุ่มของดิสก์เป็นเหมือนหน่วยเก็บข้อมูล 1 หน่วย Data Striping คือการแบ่งข้อมูลออกเป็นส่วน ๆ แล้วนำแต่ละส่วนไปเก็บในดิกส์แต่ละตัว การทำ Striping นี้จะช่วยให้การอ่าน หรือเขียนข้อมูลในดิกส์อาเรย์มีประสิทธิภาพมากขึ้น เพราะแต่ละแฟ้มข้อมูลจะถูกแบ่งเป็นส่วน ๆ กระจายไปเก็บในส่วนที่ต่างกันของดิกส์หลายตัว โดยดิกส์เหล่านั้นทำงานไปด้วยกันแบบขนาน (Parallel) จึงทำให้การเข้าถึงข้อมูลนั้นเร็วกว่าดิกส์แบบตัวเดียว

เทคโนโลยี RAID คือ การนำเอาดิกส์ตั้งแต่ 2 ตัวขึ้นไปมาทำงานร่วมกันเสมือนเป็นดิกส์ ตัวเดียวที่มีประสิทธิภาพสูงขึ้น หรือมีโอกาสที่จะสูญเสียข้อมูลน้อยลงในกรณีที่เกิดความผิดพลาดของฮาร์ดแวร์ กลุ่มของดิกส์ที่เอามาทำงานร่วมกันในเทคโนโลยี RAID จะถูกเรียกว่าดิกส์อาเรย์ โดยระบบปฏิบัติการและซอฟต์แวร์จะเห็นดิกส์ทั้งหมดเป็นตัวเดียว ซึ่งการทำ RAID นี้ จะเป็นการเพิ่มประสิทธิภาพของการเก็บรักษาข้อมูลอย่างมาก มี RAID แบบต่าง ๆ ที่มีความสามารถต่างกันและถูกเอามาใช้ในงานที่แตกต่างกัน แล้วแต่ผู้ใช้งานจะนำไปใช้งาน

แบบฝึกหัดท้ายบทที่ 10

1. จงอธิบายถึงหน้าที่ของระบบปฏิบัติการในการจัดการดิสก์คืออะไร มีรายละเอียดอย่างไรบ้าง
2. จงอธิบาย Device Driver พร้อมวาดภาพประกอบ
3. จงอธิบายความหมายคำว่า Sector, Track และ Cylinder
4. จงอธิบายข้อดีข้อเสียของวิธีการจัดการใช้ดิสก์ทั้งแบบ FCFS, SSTF, SCAN, C-SCAN และ C-LOOK
5. กำหนดให้ดิสก์ของระบบคอมพิวเตอร์ระบบหนึ่งถูกแบ่งออกเป็น 200 ไซลินเดอร์ (0-199) ขณะนี้หัวอ่านของดิสก์อยู่ที่ไซลินเดอร์ที่ 100 ถ้ามีการขอใช้ที่ไซลินเดอร์ต่าง ๆ ตามลำดับดังนี้ 55, 58, 39, 18, 90, 160, 150, 38 และ 184 จงตอบคำถามระบบจะจัดสรรการใช้ดิสก์อย่างไร เมื่อกำหนดให้ใช้การจัดการใช้ดิสก์ด้วยวิธีต่อไปนี้
    - 5.1. การจัดเวลาแบบ First Come First Served: FCFS
    - 5.2. การจัดเวลาแบบ Shortest Seek Time First: SSTF
    - 5.3. การจัดเวลาแบบ SCAN
    - 5.4. การจัดเวลาแบบ C-SCAN (Circular-SCAN)
    - 5.5. การจัดเวลาแบบ C-LOOK
6. กำหนดให้ ณ ปัจจุบันหัวอ่านดิสก์อยู่ที่ตำแหน่ง 80 และในดิสก์มีจำนวนไซลินเดอร์ทั้งหมด 200 ไซลินเดอร์ โดยไซลินเดอร์ที่จะอ่านมีดังนี้ 10 84 72 180 12 160 155 120 50 จงวาดภาพการจัดการเวลาการใช้ดิสก์ ตามโจทย์ต่อไปนี้
    - 6.1. การจัดเวลาแบบ First Come First Served: FCFS

6.2. การจัดเวลาแบบ Shortest Seek Time First: SSTF

6.3. การจัดเวลาแบบ SCAN

6.4. การจัดเวลาแบบ C-SCAN (Circular-SCAN)

6.5. การจัดเวลาแบบ C-LOOK

7. การจัดการพื้นที่ที่ใช้ในการสับเปลี่ยนคืออะไร มีประโยชน์อย่างไร

8. ถ้าองค์กรแห่งหนึ่งมีข้อมูลที่สำคัญในการเก็บเป็นอย่างมาก ถ้าจำเป็นในการเลือกใช้ RAID เพื่อนำมาใช้งานในองค์กรนี้ ควรจะใช้ RAID ระดับใด จงอธิบายเหตุผลในการเลือกใช้มาให้เข้าใจ

9. การนำ RAID ระดับต่าง ๆ มาใช้ร่วมกันมีประโยชน์อย่างไร จงอธิบายโดยให้เหตุผลหลักในการนำ RAID มาใช้ร่วมกัน

บรรณานุกรมประจำบทที่ 10

กลุ่มระบบเครือข่ายคอมพิวเตอร์ กรมตรวจบัญชีสหกรณ์. สืบค้นวันที่ 2 กรกฎาคม 2557 จาก http://netgrp.cad.go.th/ewt_news.php?nid=216&filename=index

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

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

เฉลยแบบฝึกหัด

บทที่ 3

ข้อที่ 7

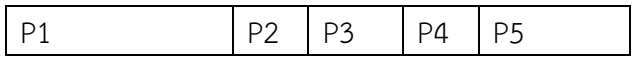
semaphore S ทำให้มั่นใจได้ว่าเมื่อเริ่มทำงานแล้วทั้งสองโพรเซส จะทำงานได้จนเสร็จโดยไม่มีการ interrupt ดังนั้น ลำดับการทำงานของคำสั่งจาก A และ B ที่เป็นไปได้จึงเหลือแค่ :

A1 A2 B1 B2: X = 11
B1 B2 A1 A2: X = 12

บทที่ 6

ข้อที่ 6

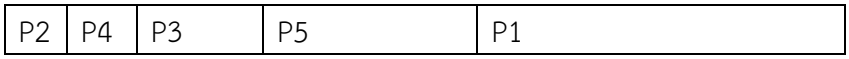
FCFS



เวลา 0 10 11 13 14 19

Turn around time ของ process	P1=10
	P2=11
	P3=13
	P4=14
	P5=19
Waiting time ของ process	P1=0
	P2=10
	P3=11
	P4=13
	P5=14
Average waiting time	= (0+10+11+13+14)/5
	= 9.6 milliseconds

SJF



เวลา 0 1 2 4 9 19

Turn around time ของ process P1=19
P2=1
P3=4
P4=2
P5=9

Waiting time ของ process P1=9
P2=0
P3=2
P4=1
P5=4

Average waiting time = (9+0+2+1+4)/5
= 3.2 milliseconds

Nonpreemptive priority

เวลา 0 1 6 16 18 19

Turn around time ของ process P1=16
P2=1
P3=18
P4=19
P5=6

Waiting time ของ process P1=6
P2=0
P3=16
P4=18
P5=1

Average waiting time = (6+0+16+18+1)/5
= 8.2 milliseconds

RR (quantum=1)

	P1	P2	P3	P4	P5	P1	P3	P5	P1	P5	P1	P5	P1	P5	P1	P1	P1	P1	P1
เวลา	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Turn around time ของ process P1=19
P2=2
P3=7
P4=4
P5=14

Waiting time ของ process P1=9 (0+4+2+1+1+1)
P2=1
P3=2+3=5
P4=3
P5=4+2+1+1+1=9

Average waiting time = (10+1+5+3+9)/5
=5.4 milliseconds

บทที่ 7

ข้อที่ 9

1. เป็น Segment 0 ขนาด 413 จาก Segment Table ใน Segment 0 เริ่มต้นที่ตำแหน่ง 216 ขนาด 600 ในช่วงตำแหน่งอ้างอิง Physical Address ของ Segment 0 คือ 216 + 600 = 816 เพราะฉะนั้น Physical Addresses ของ Logical Addresses 0,413 คือ 216+413 = 629 อยู่ในช่วงตำแหน่งอ้างอิง Physical Address ของ Segment 0

2. เป็น Segment 1 ขนาด 100 จาก Segment Table ใน Segment 1 เริ่มต้นที่ตำแหน่ง 2300 ขนาด 14 อยู่ในช่วงของ Segment 1 คือ 2300 + 14 = 2314 เพราะฉะนั้น Physical Addresses ของ Logical Addresses 1,100 คือ 2300+100 = 2400 จึงเกินตำแหน่งอ้างอิงอยู่ Physical Addresses ของ Segment 1

3. เป็น Segment 2 ขนาด 500 จาก Segment Table ใน Segment 2 เริ่มต้นที่ตำแหน่ง 90 ขนาด 100 อยู่ในช่วงของ Segment 2 คือ 90 + 100 = 190 เพราะฉะนั้น Physical Addresses ของ

Logical Addresses 2,500 คือ 90+500 = 590 จึงเกินตำแหน่งอ้างอิงอยู่ Physical Addresses ของ Segment 2

4. เป็น Segment 3 ขนาด 400 จาก Segment Table ใน Segment 3 เริ่มต้นที่ตำแหน่ง 1327 ขนาด 580 ในช่วงตำแหน่งอ้างอิง Physical Address ของ Segment 3 คือ 1327 + 580 = 1907 เพราะฉะนั้น Physical Addresses ของ Logical Addresses 3,400 คือ 1327+400 = 1727 อยู่ในช่วงตำแหน่งอ้างอิง Physical Address ของ Segment 3

5. เป็น Segment 4 ขนาด 112 จาก Segment Table ใน Segment 4 เริ่มต้นที่ตำแหน่ง 1952ขนาด 96 อยู่ในช่วงของ Segment 4 คือ 1952 + 96 = 2048 เพราะฉะนั้น Physical Addresses ของ Logical Addresses 4,112 คือ 1952+112 = 2064 จึงเกินตำแหน่งอ้างอิงอยู่ Physical Addresses ของ Segment 4

บทที่ 8

ข้อที่ 5

Effective access time = $(1 - p) \times (2000) + p$ (10 milliseconds)

$= (1 - p) \times 2000 + p \times 10{,}000{,}000$

$= 2000 + 9{,}998{,}000 \times p$. นาโนวินาที

บรรณานุกรม

กลุ่มระบบเครือข่ายคอมพิวเตอร์ กรมตรวจบัญชีสหกรณ์. สืบค้นวันที่ 2 กรกฎาคม 2557 จาก http://netgrp.cad.go.th/ewt_news.php?nid=216&filename=index

บุญสืบ โพธิ์ศรีและคณะ. (2550). **เทคโนโลยีสารสนเทศเบื้องต้น.** นนทบุรี : เจริญรุ่งเรืองการพิมพ์.

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

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

วีกีพีเดีย. Retrieved March 2, 2014 from http://th.wikipedia.org/wiki/คอมพิวเตอร์

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. (2014). **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/

ดัชนี (Index)

OPERATING SYSTEM

ระบบปฏิบัติการ