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

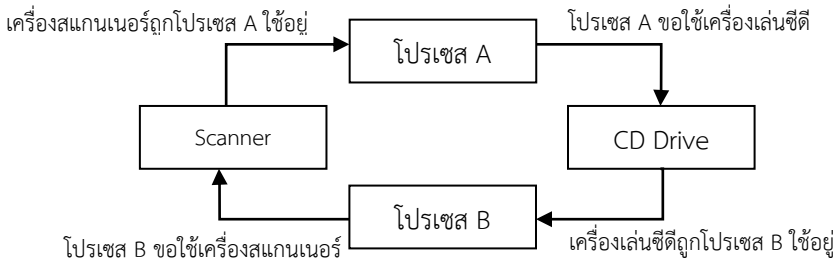
3. การคืน (Release) โพรเซสต้องคืนทรัพยากรที่ใช้เสร็จแล้วกลับสู่ระบบ เพื่อเปิดโอกาสให้โพรเซสอื่นที่ต้องการใช้ทรัพยากรนั้นได้รับการจัดสรรทรัพยากรต่อไป

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

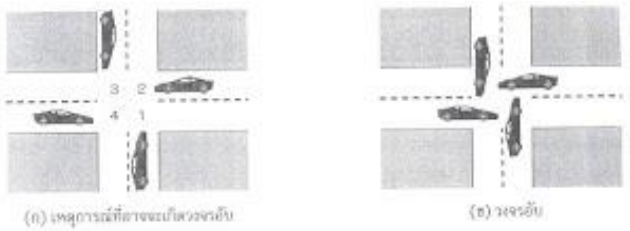


ภาพที่ 5.1 การแสดงโพรเซสสองโพรเซสเกิดวงจรอับ

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

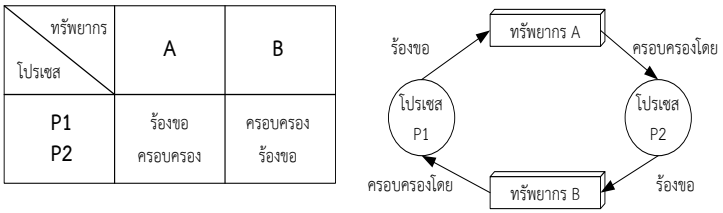
1. การไม่เกิดร่วม (Mutual exclusion) ในกรณีที่เป็นทรัพยากรที่ไม่สามารถใช้ร่วมกันได้ ทำให้ทรัพยากรถูกใช้งานได้เพียงครั้งละ 1 โพรเซส คือ ขณะเวลาหนึ่งจะมีเพียงโพรเซสเดียวที่ใช้งานทรัพยากรได้ ไม่สามารถมีโพรเซสอื่นใช้งานทรัพยากรพร้อมกันได้

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

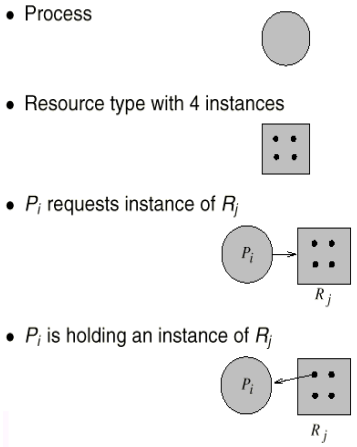
เงื่อนไขสามข้อแรกนั้นจำเป็น แต่ไม่เพียงพอต่อการทำให้เกิดวงจรอับ เงื่อนที่สี่แท้จริงแล้วเป็นผลที่เกิดจากเงื่อนไขทั้งสามข้อแรก นั่นคือเมื่อเงื่อนไขทั้งสามข้อแรกเกิดขึ้น ผลที่เกิดตามมาอาจจะทำให้เกิดวงจรรอคอย ซึ่งแท้จริงก็คือวงจรอับนั่นเอง วงจรรอคอยในภาพที่ 5.3 เป็นวงจรที่ระบบไม่สามารถแก้ได้ เพราะเงื่อนไขทั้งสามข้อแรกบังคับอยู่ ดังนั้นเงื่อนไขทั้งสี่ข้อจะต้องสมบูรณ์จึงจะทำให้เกิดวงจรอับได้



ภาพที่ 5.3 วงจรลูกโซ่ของโพรเซสที่ต่างรอคอยซึ่งกันและกัน

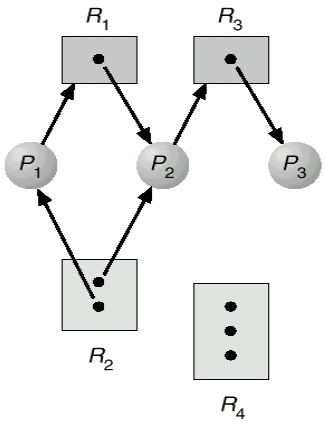
5.5 การกำหนดทรัพยากรด้วยกราฟ

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



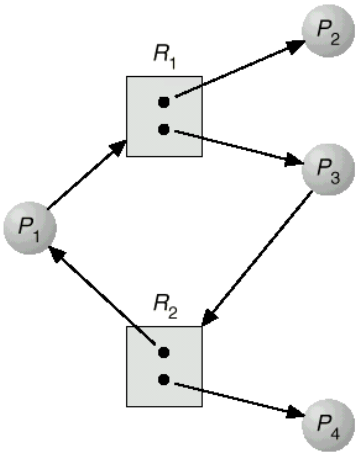
ภาพที่ 5.4 รูปแบบต่าง ๆ ของกราฟ

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

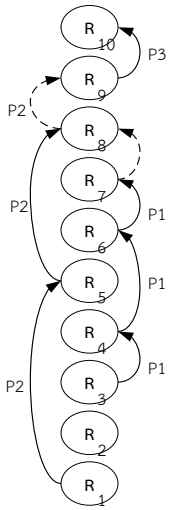
ตัวอย่าง 5.6.3(2) โพรเซส A ร้องขอทรัพยากร R1 ให้พิจารณาว่า ระบบจะจัดสรรทรัพยากรให้โพรเซส A หรือไม่ ในการพิจารณานั้นระบบต้องตรวจสอบก่อนว่า ทรัพยากร R1 ว่างหรือไม่ ถ้าทรัพยากร R1 ว่าง

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	

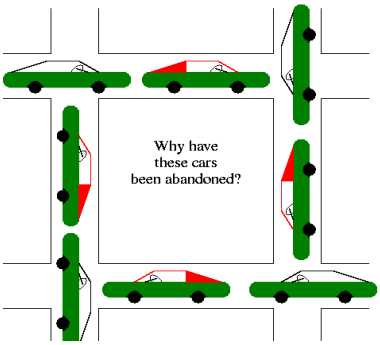
จงตอบคำถามต่อไปนี้โดยใช้ Banker's Algorithm

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.