บทที่ 6
การติดตาย
(Deadlock)

ในระบบประมวลผลแบบหลายโปรแกรม (Multiprogramming) หลายกระบวนการ (Process) ที่ทำการประมวลผลจำเป็นต้องใช้ทรัพยากร (Resource) เป็นจำนวนมากดังนั้น กระบวนการเหล่านั้นอาจแย่งกันใช้ทรัพยากรในระบบซึ่งมีจำนวนจำกัด ในการทำงานแต่ละกระบวนการจะร้องขอที่จะใช้แต่ละทรัพยากร และถ้าทรัพยากรนั้นไม่ว่าง ในขณะที่กระบวนการร้องขอเพื่อใช้ในการประมวลผล กระบวนการนั้นจะเข้าไปอยู่ในสถานะรอ (Wait State) ถ้ามีกระบวนการเป็นจำนวนมากอยู่ในสถานะการรอใช้ทรัพยากร ซึ่งทรัพยากรที่ถูกรอนั้น ถูกใช้โดยการบวนการอื่นอยู่ ดังนั้นอาจเกิดเหตุการณ์ที่กระบวนการอยู่ในสถานะรอคอยทรัพยากรตลอดไป เราเรียกว่าเกิดการติดตาย (Dead Lock) ซึ่งจะเกิดในระบบการประมวลผลแบบการแบ่งเวลา (Timesharing)

รูปแบบการทำงานของระบบ

การทำงานของระบบโดยทั่วไปแล้วทรัพยากรที่ใช้ จะมีจำนวนจำกัดและกระจายทั่วไปในระบบการบวนการต่าง ๆ จะร้องขอใช้ทรัพยากรเหล่านั้น ทรัพยากร ถูกแบ่งออกเป็นหลายประเภท เช่นเวลาที่เข้าใช้ซีพียู (CPU Cycle) เนื้อที่ว่างของหน่วยความจำ (Memory Space)ไฟล์ข้อมูล (Files) และอุปกรณ์ Input/Output (I/O Device) ทรัพยากรแต่ละ ชนิดจะมีมากกว่า 1 ตัว ซึ่งจะมีตัวเลขที่ระบุเพื่อการเรียกใช้

การเรียกใช้ทรัพยากรแต่ละประเภทดำเนินการดังนี้ ถ้ามีกระบวนการร้องขอใช้ทรัพยากรชนิดใดชนิดหนึ่ง 1 ตัว แล้วระบบสามารถจัดสรรทรัพยากรชนิดนั้นซึ่งมีหลายตัวตัวใดตัวหนึ่งก็ได้ให้ใช้ แสดงว่าทรัพยากรเหล่านั้นเป็นชนิดเดียวกันใช้แทนกันได้ แต่ถ้ากระบวนการได้ร้องขอทรัพยากรที่ระบุตัวเลขของทรัพยากรนั้น ๆ ไว้ ระบบจำเป็นต้องจัดทรัพยากรเฉพาะตัวที่นั้นให้ตามที่กระบวนการร้องขอ เราจะถือว่าทรัพยากรเหล่านั้นไม่เหมือนกันหรือเป็นคนละชนิดกัน เช่น เราอาจกำหนดให้ใช้เครื่องพิมพ์ 2 เครื่องในระบบที่เหมือนกันทุกอย่าง ระบบจะถือว่าเป็นทรัพยากรชนิดเดียวกัน ถ้าผู้ใช้ไม่สนใจว่า ผลลัพธ์จะพิมพ์ออกทางเครื่องใดจาก 2 เครื่องนี้ แต่ถ้าเครื่องพิมพ์เครื่องหนึ่งอยู่ชั้นที่ 8 และอีกเครื่องอยู่ชั้น 2 ผู้ใช้อยู่บนชั้นที่ 8 จะเห็นว่าเครื่องพิมพ์ทั้งสองนี้ไม่เหมือนกัน จะต้องกำหนดให้เครื่องพิมพ์ทั้งสองนี้เป็นทรัพยากรต่างชนิดกัน

การทำงานของกระบวนการจะต้องร้องขอใช้ทรัพยากรก่อนที่จะได้ใช้ และเมื่อใช้เสร็จแล้ว จะต้องปลดปล่อยทรัพยากรนั้น กระบวนการหนึ่ง ๆ อาจจะร้องขอใช้ทรัพยากรหลายตัวตามที่ต้องการได้ จำนวนทรัพยากรที่ร้องขอใช้อาจจะมีว่างไม่พอตามจำนวนที่กระบวนการต้องการ กระบวนการจะทำงานได้เมื่อมีทรัพยากรที่ต้องการใช้ครบตามที่ร้องขอ ดังนั้นระบบจะต้องจัดสรรทรัพยากรต่าง ๆ ให้แก่กระบวนการที่ร้องรอได้อย่างพอเพียงและเป็นระบบ โดยมีลำดับขั้นตอนของกระบวนการที่ขอใช้ทรัพยากรดังต่อไปนี้

1. การร้องขอ (Request) ถ้าการร้องขอนั้นไม่ได้รับอนุญาตจากระบบให้ใช้ทรัพยากรที่ต้องการนั้น เนื่องจากทรัพยากรนั้นอาจจะกำลังถูกใช้งานโดยกระบวนการอื่น ดังนั้นกระบวนการที่ร้องขอ จะต้องรอจนกว่าจะได้รับอนุญาตให้ได้ทรัพยากรที่ต้องการ

2. การใช้งาน (Use) เมื่อกระบวนการที่ร้องขอ ขอใช้ทรัพยากรและได้รับอนุญาต กระบวนการนั้นจะสามารถใช้งานทรัพยากรนั้นได้ เช่น ในกรณีที่ทรัพยากรที่ร้องขอคือเครื่องพิมพ์ เมื่อได้รับการจัดสรรจากระบบแล้ว กระบวนการนั้นสามารถพิมพ์ข้อมูลออกทางเครื่องพิมพ์ได้

3. การคืน (Release) เมื่อกระบวนการใช้ทรัพยากรเสร็จแล้ว จะต้องคืนทรัพยากรนั้นแก่ระบบ

การร้องขอใช้ทรัพยากรและการคืนทรัพยากรแก่ระบบ เป็นการควบคุมการกระทำของคำสั่งเรียกระบบ (System Calls) ที่กล่าวไว้ในบทที่ 3 ถ้ามีผู้ใช้ขอใช้ทรัพยากรแต่ละครั้ง ระบบปฏิบัติการจะต้องตรวจสอบเพื่อให้แน่ใจว่า กระบวนการได้ร้องขอใช้ทรัพยากร และได้รับการจัดสรรทรัพยากรให้เรียบร้อยแล้ว โดยที่ระบบจะมีตารางที่บันทึกสถานะของทรัพยากร ว่าขณะนี้ทรัพยากรมีสถานะเป็นอย่างไร ทรัพยากรใดยังว่างอยู่หรือกำลังถูกใช้งาน ถ้ากำลังถูกใช้งานก็จะตรวจดูว่ากระบวนการใดเป็นผู้ใช้ ถ้ามีกระบวนการร้องขอทรัพยากรที่กำลังถูกใช้งาน โดยกระบวนการอื่น กระบวนการที่ร้องขอจะถูกจัดเข้าไป อยู่ในแถวคอยของกระบวนการที่รอคอย (Process Waiting Queue) การใช้ทรัพยากรนั้นอยู่

ทุก ๆ กระบวนการที่อยู่ในแถวคอยของกระบวนการ จะอยู่ในสถานะติดตาย (Deadlock) เมื่อทุก ๆ กระบวนการที่อยู่ในกลุ่มนั้นต่างก็กำลังรอคอยที่จะใช้ทรัพยากรที่กำลังถูกใช้โดยกระบวนการอื่นในกลุ่มนั้น สถานการณ์ที่เราจะให้ความสนใจเป็นอย่างมากในที่นี้ คือ การร้องขอและการคืนทรัพยากร ทรัพยากรอาจจะเป็นทรัพยากรเชิงกายภาพ (เช่น เครื่องพิมพ์ เครื่องขับเทป ที่ว่างในหน่วยความจำ และเวลาเข้าใช้ ซีพียู) หรือทรัพยากรเชิงตรรกะ (เช่น ไฟล์ข้อมูล Semaphore และการควบคุม) นอกจากนี้ยังมีเหตุการณ์อื่น ๆ ที่อาจทำให้เกิดติดตายขึ้นได้ เช่น การสอดคล้องของกระบวนการ และ การสื่อสารระหว่างกระบวนการ

ในการที่จะอธิบายเพื่อให้เห็นภาพสถานะของติดตาย (Deadlock) ที่ชัดเจนมากขึ้น เราจะพิจารณาตัวอย่างของระบบที่ประกอบไปด้วยเครื่องขับเทป 3 ตัว และสมมุติว่ามีกระบวนการ 3 กระบวนการ และแต่ละกระบวนการกำลังใช้เครื่องขับเทปอยู่กระบวนการละเครื่อง ต่อมาถ้าแต่ละกระบวนการร้องขอเครื่องขับเทปเพิ่มอีกกระบวนการละ 1 เครื่อง ระบบไม่สามารถจัดสรรเครื่องขับเทปให้แก่กระบวนใดได้อีกเนื่องจากเครื่องขับเทปถูกใช้หมด ดังนั้นทั้ง 3 กระบวนการจะติดอยู่ในติดตายทันที โดยที่แต่ละกระบวนการกำลังรอเหตุการณ์ ที่เครื่องขับเทปถูกปล่อยคืนสู่ระบบ ซึ่งเป็นเหตุการณ์ที่ขึ้นกับกระบวนการอื่นในระบบเดียวกัน (กำลังรอคอยทรัพยากรอยู่เช่นกัน) จากตัวอย่างเป็นการแสดงภาพของติดตายที่เกิดจากการที่กระบวนการแย่งกันใช้ทรัพยากรประเภทเดียวกัน

ติดตายอาจจะเกิดขึ้นกับทรัพยากรที่ต่างประเภทกัน เช่น ในระบบมีเครื่องพิมพ์ 1 เครื่อง เครื่องขับเทป 1 เครื่อง และถ้าสมมุติว่า มีกระบวนการ P_i กำลังใช้เครื่องขับเทปอยู่ ส่วนกระบวนการ P_j กำลังใช้เครื่องพิมพ์ ต่อมา P_i ร้องขอใช้เครื่องพิมพ์เพิ่ม และ P_j ร้องขอเครื่องขับเทปเพิ่ม ระบบนี้จะเกิดติดตายเพราะเครื่องพิมพ์ และเครื่องขับเทปไม่ว่างให้ใช้ ทั้งสองกระบวนการต้องรอ

ลักษณะของติดตาย

ระบบทั่วไปไม่พึงประสงค์ที่จะให้เกิดติดตาย เพราะจะทำให้ประสิทธิภาพการทำงานต่ำลง เนื่องจากเมื่อเกิดติดตายแล้วจะไม่มีกระบวนการใดได้ทำงานจนเสร็จ และทรัพยากรต่าง ๆ ของระบบต่างก็ถูกใช้ทั้งหมด ทำให้งานอื่นไม่ได้ทำ

เงื่อนไขการเกิดติดตาย (Necessary Conditions)

ติดตายจะเกิดขึ้นได้ก็การทำงานของระบบเกิดเงื่อนไขต่อไปนี้

1. มีทรัพยากรที่ใช้ร่วมกันไม่ได้ (Mutual Exclusion) มีทรัพยากรอย่างน้อย 1 ตัว อยู่ในกลุ่มของทรัพยากรที่ใช้ร่วมกันไม่ได้คือ มีเพียงกระบวนการเดียวเท่านั้นที่สามารถใช้ทรัพยากรตัวนั้น ถ้ามีกระบวนการอื่นร้องขอใช้ทรัพยากรที่กำลังถูกใช้อยู่ กระบวนการนั้นจะต้องรอจนกว่าทรัพยากรนั้นจะถูกคืนให้แก่สู่ระบบ

2. การถือครองและรอคอย (Hold and Wait) มีกระบวนการที่กำลังใช้ทรัพยากรอยู่อย่างน้อย 1 ตัว และขณะเดียวกันก็กำลังรอคอยใช้ทรัพยากรอื่นเพิ่มอีกซึ่งทรัพยากรนั้นกำลังถูกใช้โดยกระบวนการอื่น

3. ไม่แทรกกลางคัน (No Preemption) ถ้ากระบวนการกำลังใช้ทรัพยากรใดอยู่ และยังทำงานไม่เสร็จ แต่มีกระบวนการอื่นมาขอใช้ทรัพยากรนั้น จะไม่มีการแทรกกลางคัน คือ

กระบวนการจะไม่ยอมคืนทรัพยากรให้แก่ระบบ เพื่อจัดสรรให้กระบวนการใหม่ที่ร้องขอมา กระบวนการจะยอมคืนทรัพยากรใช้งานอยู่ให้แก่สู่ระบบ เมื่อกระบวนการนั้นได้ทำงานเสร็จแล้ว

4. การรอคอยเป็นวงรอบ (Circular Wait) มีกระบวนการหลายกระบวนการ(P_0, P_1, …, P_n) ที่กำลังรอคอยทรัพยากรที่ถูกใช้อยู่ ซึ่ง P_0 กำลังรอคอยทรัพยากรที่ถูกใช้โดย P_1 และ P_1 ก็กำลังรอคอยทรัพยากรที่ถูกใช้โดย P_2 และรอคอยต่อกันไปจนถึง P_{n-1} รอคอยใช้ทรัพยากรที่ถูกใช้โดย P_n **และ** P_n รอคอยใช้ทรัพยากรที่ถูกใช้โดย P_0 ซึ่งสามารถแสดงให้เห็นได้ชัดจากกราฟของการจัดสรรทรัพยากร (Resource Allocation Graph)

ถ้าระบบมีเงื่อนไขเกิดขึ้นครบทั้ง 4 นี้จะเกิดติดตายแน่นอน และปรากฏว่าเงื่อนไขทั้ง 4 ข้อ ไม่ได้เป็นอิสระต่อกันอย่างแท้จริง เช่น การรอคอยเป็นวงรอบ จะเกิดจากการที่มีเงื่อนไขการถือครองและรอคอย

กราฟของการจัดสรรทรัพยากร

ติดตายสามารถอธิบายให้เข้าใจและเห็นภาพได้ดียิ่งขึ้นโดยอธิบายในรูปของ กราฟทางเดียว ที่เรียกว่า “กราฟของการจัดสรรทรัพยากรในระบบ” (system resource – allocation graph) โดยที่กราฟจะประกอบด้วยเซตของจุดยอด V และเซตของเส้นขอบ E สำหรับเซตของจุดยอด V ถูกแบ่งออกเป็น 2 รูปแบบ คือ P และ R

P = {P1, P2, …., Pn} เป็นเซตที่ประกอบด้วย กระบวนการทั้งหมดในระบบ

R = {R1, R2, …., Rm} - set ประกอบด้วย Processes type ทั้งหมดในระบบ

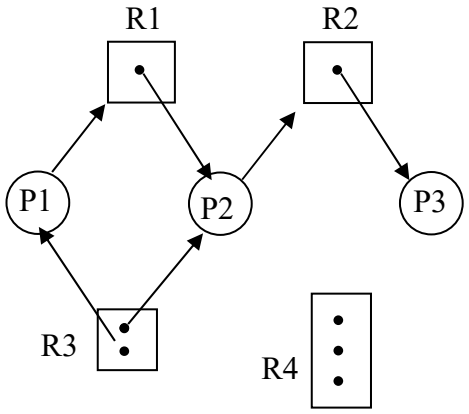
เส้นขอบ E เป็นเป็นเซตของเส้นลูกศรที่ชี้จากกระบวนการไปยังทรัพยากร เช่นกราฟของกระบวนการ P ตัวที่ i กับ ทรัพยากร R ประเภท j เขียนแทนด้วยกราฟ $P_i \rightarrow R_j$ หมายความว่า กระบวนการ P ตัวที่ i ร้องขอใช้ ทรัพยากร R ประเภท j และกำลังรอคออยู่ (Pi ร้องขอใช้ทรัพยากร Ri) ส่วนกราฟของ ทรัพยากร R ประเภท j กับกระบวนการ P ตัวที่ i เขียนแทนด้วยกราฟ $R_j \rightarrow P_i$ หมายความว่า ทรัพยากร R ประเภท j ถูกใช้โดย กระบวนการ P ตัวที่ i (Rj ถูกใช้โดย Pi) จากเส้นขอบ E ของระบบสรุปได้ดังนี้

$P_i \rightarrow R_j$ ถูกเรียกว่า เส้นร้องขอ (request edge)

$R_j \rightarrow P_i$ ถูกเรียกว่า เส้นถือครอง (assignment edge)

การเขียนกราฟของการจัดสรรทรัพยากรในระบบ เราจะใช้วงกลม ○ แทนกระบวนการ Pi ใช้สี่เหลี่ยม □ แทนทรัพยากร Rj และใช้จุด . แทนจำนวนทรัพยากรแต่ละประเภท บรรจุไว้กรอบในสี่เหลี่ยมของ Rj เช่น ⊡ โดยที่เส้นร้องขอจะต้องชี้ไปยังกรอบสี่เหลี่ยม ของทรัพยากร

Rj เท่านั้น แต่เส้นถือครอง จะต้องชี้ไปยังจุดใดจุดหนึ่งในสี่เหลี่ยม เมื่อกระบวนการ Pi ร้องขอทรัพยากรประเภท Rj เราก็จะเขียนเส้นร้องขอลงในกราฟของการจัดสรรทรัพยากร และเมื่อการร้องขอนั้นได้รับอนุมัติจากระบบ เส้นร้องขอก็จะถูกแปลงไปเป็นเส้นถือครองแทน และหลังจากที่กระบวนการปล่อยทรัพยากรคืนสู่ระบบแล้ว เส้นถือครองก็จะถูกลบออกไปจากกราฟ



รูปที่ 6.1 กราฟของการจัดสรรทรัพยากร

จากกราฟ แสดงเซตของระบบดังนี้

P = {P1, P2, P3}

R = {R1, R2, R3, R4}

E = { P1 $\rightarrow$ R1, P2 $\rightarrow$ R2, R1 $\rightarrow$ P2,

,R3$\rightarrow$P1,R3$\rightarrow$ P2, R2 $\rightarrow$ P3}

ประเภทของทรัพยากรในระบบมีดังนี้

R1 มี 1 ตัว R2 มี 1 ตัว

R3 มี 2 ตัว R4 มี 3 ตัว

สถานะของกระบวนการมีดังนี้

กระบวนการ P1 กำลังใช้ทรัพยากร R2 และกำลังรอใช้ทรัพยากร R1

กระบวนการ P2 กำลังใช้ทรัพยากร R1 และ R2 และกำลังรอใช้ทรัพยากร R3

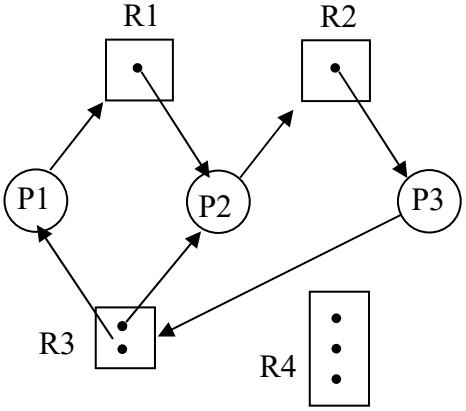
กระบวนการ P3 กำลังใช้ทรัพยากร R3

จากเงื่อนไขของการเกิดติดตายจะต้องมีเงื่อนไขครบทั้ง 4 ข้อแต่อาจพิสูจน์ได้โดยง่ายว่า “ถ้าไม่มีการรอเป็นวงรอบ (Circular Wait) ในกราฟแล้ว ระบบจะไม่เกิดติดตาย แต่ในทางตรงข้าม ถ้ามีวงจรปรากฏในกราฟแล้วแสดงว่าในระบบอาจมีติดตายเกิดขึ้น”

ถ้าเป็นระบบที่ทรัพยากรแต่ละประเภทมีเพียง 1 ตัว เมื่อมีวงจรในกราฟ ก็จะต้องมีติดตายในระบบแน่นอน หรือถ้าระบบมีทรัพยากรประเภทละหลายตัว และวงจรในกราฟลากผ่านเฉพาะทรัพยากรประเภทที่มีตัวเดียว ก็จะสรุปได้ว่า มีติดตายเกิดขึ้น และทุก ๆ กระบวนการในวงจรของกราฟ ก็จะติดอยู่ในติดตายด้วย ซึ่งในกรณีนี้วงจรในกราฟเป็นเงื่อนไขที่จำเป็น และเพียงพอในการเกิดติดตายในระบบ

แต่ถ้าทรัพยากรแต่ละประเภทมีจำนวนมากกว่า 1 ตัว วงจรที่เกิดขึ้นในกราฟ ไม่อาจเป็นตัวบอกได้เสมอไปว่า เกิดติดตายในระบบ ในกรณีนี้วงจรที่เกิดในระบบ เป็นเงื่อนไขที่จำเป็นในการใช้ตรวจสอบติดตาย แต่ไม่เพียงพอที่จะใช้แสดงว่ามีติดตายเกิดขึ้นจริง ๆ ในระบบ

เพื่อเป็นการแสดงภาพตามแนวคิดที่กล่าวมา เราจะกลับไปสู่กราฟการจัดสรรทรัพยากรในรูปก่อนหน้านี้ สมมติว่ากระบวนการ P_3 ได้ร้องขอทรัพยากรประเภท R_2 แต่เนื่องจากไม่มีทรัพยากรตัวใดในประเภท R_2 ว่างเลย ดังนั้นเส้นร้องขอ $P_3 \rightarrow R_2$ จึงถูกใส่เพิ่ม เข้าไปในกราฟซึ่งแสดงในรูปที่ 7.2



รูปที่ 6.2 กราฟของการจัดสรรทรัพยากรที่เกิดติดตาย

จากกราฟนี้ จะเกิดวงจรอย่างน้อย 2 วงขึ้นในระบบ คือ

วงแรก คือ $\mathbf{P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1}$

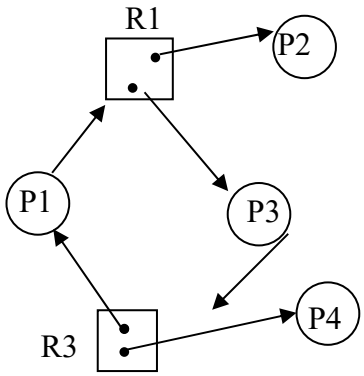
วงที่สอง คือ $\mathbf{P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2}$

เกิดติดตายกับกระบวนการ $\mathbf{P_1}$, $\mathbf{P_2}$ และ $\mathbf{P_3}$ กระบวนการ $\mathbf{P_2}$ กำลังรอคอยทรัพยากร $\mathbf{R_3}$ ซึ่งกำลังถูกใช้โดยกระบวนการ $\mathbf{P_3}$ และกระบวนการ $\mathbf{P_3}$ ก็กำลังร้องขอใช้ทรัพยากร $\mathbf{R_2}$ ซึ่งถูกถือใช้กระบวนการ P_1 และ P_2 หรืออาจกล่าวได้ว่ากระบวนการ P_3 กำลังรอคอยให้ P_1 หรือ P_2 คืนทรัพยากร R_2 แก่ระบบ

ลองพิจารณารูปที่ 6.3 จะเห็นว่ามีวงจรเกิดขึ้นในกราฟดังนี้

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

แต่ปรากฏว่า วงจรนี้จะไม่เกิดติดตาย เนื่องจาก กระบวนการ P_4 จะปล่อยทรัพยากรประเภท R_2 ที่ตนเองถือครองอยู่เมื่อใช้เสร็จ ซึ่งเมื่อทรัพยากร R_2 ถูกปล่อยกลับสู่ระบบแล้ว ระบบก็สามารถที่จะจัดทรัพยากรประเภท R_2 ให้แก่กระบวนการ P_3 ได้ วงจรก็จะขาดทันที



รูปที่ 6.3 กราฟของการจัดสรรทรัพยากรที่เกิดติดตาย

การแก้ปัญหาติดตาย

การแก้ปัญหาติดตายมีอยู่ 3 วิธีหลัก ๆ คือ

1. พยายามไม่ให้เกิดติดตายโดยการกำหนดกฎเกณฑ์บางอย่างในการใช้ทรัพยากร เพื่อให้แน่ใจว่าระบบจะไม่มีทางเกิดติดตายได้ มีวิธีการจัดการ 2 วิธีคือ การป้องกันการเกิดติดตาย (Deadlock Prevention) และการหลีกเลี่ยงการเกิดติดตาย (Deadlock Avoidance)

2. ปล่อยให้ระบบเกิดติดตายขึ้นก่อน แล้วค่อยตรวจหาแล้วจึงตามแก้ไขการเกิดติดตาย (Deadlock Detection and Recovery From Deadlock)

3. วิธีการผสมผสาน (Combined Approach to Deadlock Handle) เป็นการนำหลาย ๆ วิธีมาแก้ปัญหาการเกิดติดตาย หรืออาจใช้วิธีมองข้ามปัญหาทั้งหมด แล้วแสร้งทำว่าติดตายไม่เคยเกิดขึ้นในระบบ วิธีการแก้ปัญหาวิธีนี้เป็นวิธีการหนึ่งที่ถูกใช้ในระบบปฏิบัติการส่วนใหญ่โดยการ Restart เครื่องใหม่

การป้องกันการเกิดติดตาย (Deadlock Prevention)

ตามที่ได้กล่าวมาแล้วว่าระบบจะเกิดติดตายขึ้นได้จะต้องเกิดจากเงื่อนไขในการเกิดติดตาย (Necessary Conditions) ทั้ง 4 ข้อ พร้อมกันในระบบ ดังนั้นเพียงเราป้องกันไม่ให้เกิดเงื่อนไขข้อใดข้อหนึ่งจาก 4 ข้อนั้น ก็จะสามารถป้องกันการเกิดติดตาย (Deadlock Prevention) ได้ ซึ่งสามารถแยกพิจารณาทีละเงื่อนไขได้ ดังนี้

1. มีทรัพยากรที่ใช้ร่วมกันไม่ได้ (Mutual Exclusion) เงื่อนไขในข้อนี้ คือ การที่ระบบไม่อนุญาตให้มีการใช้ทรัพยากรร่วมกัน เช่น เครื่องพิมพ์จะไม่สามารถให้กระบวนการหลาย ๆ กระบวนการใช้พร้อม ๆ กันได้ แต่ถ้าเรายอมให้ในระบบมีการใช้ทรัพยากรร่วมกันได้ ปัญหาติดตายก็จะไม่เกิด เช่น แฟ้มข้อมูลที่อ่านได้อย่างเดียว เป็นตัวอย่างหนึ่งของทรัพยากรที่สามารถใช้ร่วมกันได้ ถ้ามีกระบวนการหลาย ๆ กระบวนการร้องขอที่จะเปิดแฟ้มข้อมูลนี้พร้อม ๆ กัน กระบวนการเหล่านั้นจะได้รับอนุญาตให้ใช้แฟ้มข้อมูลได้พร้อมกันกระบวนการหนึ่ง ๆ ไม่จำเป็นที่จะต้องรอให้กระบวนการอื่นคืนทรัพยากรที่ต้องการสู่ระบบ แต่สามารถใช้ทรัพยากรตัวนั้นร่วมกันได้เลย อย่างไรก็ตาม การที่เราจะป้องกันการเกิดติดตายในระบบ โดยการป้องกันเงื่อนไขนี้ไม่สามารถทำได้เสมอไป เพราะยังมีทรัพยากรบางประเภทที่ไม่มีทางใช้ร่วมกันได้

2. การถือครองและรอคอย (Hold and Wait) การที่จะไม่ให้ระบบเกิด “การถือครองและรอคอย” ขึ้น จะต้องกำหนดว่า ถ้ากระบวนการใดจะร้องขอใช้ทรัพยากร กระบวนการนั้นจะต้องไม่ได้ถือครองทรัพยากรใด ๆ อยู่ ณ เวลานั้นการจัดการทำได้ 2 วิธีคือ

วิธีที่ 1 ก่อนที่จะเริ่มต้นทำงาน ให้กระบวนการร้องขอทรัพยากรที่ต้องการใช้ทั้งหมดตลอดการทำงาน เราอาจดำเนินการตามวิธีนี้ได้ โดยกำหนดให้การร้องขอใช้ทรัพยากรเป็นคำสั่งของการเรียกระบบ (System Call) ที่ต้องทำก่อนการทำงานใด ๆ ของกระบวนการ

วิธีที่ 2 ยอมให้กระบวนการร้องขอทรัพยากรได้ ก็ต่อเมื่อกระบวนการนั้นมิได้ถือครองทรัพยากรใดไว้เลย ตัวอย่างเช่น กระบวนการหนึ่งอาจร้องขอทรัพยากรบางส่วนและใช้ทรัพยากรนั้นไปก่อน และเมื่อกระบวนการนั้นต้องการทรัพยากรเพิ่มอีก กระบวนการนั้นก็จะต้องคืนทรัพยากรที่ถือครองอยู่กลับสู่ระบบเสียก่อน จึงจะร้องขอใหม่ได้

ความแตกต่างของ 2 วิธีนี้ พิจารณาจากตัวอย่างของกระบวนการหนึ่งที่ต้องการ

1. คัดลอกข้อมูลจากเทปลงไปเก็บที่แฟ้มข้อมูลในดิสก์
2. เรียงลำดับข้อมูลของแฟ้มข้อมูลในดิสก์
3. พิมพ์ผลลัพธ์ออกสู่เครื่องพิมพ์

วิธีแรก ถ้าทรัพยากรทั้งหมดต้องถูกร้องขอในตอนเริ่มต้นงานของกระบวนการ แสดงว่ากระบวนการนี้ก็จะถือครองเครื่องพิมพ์ไว้ตลอดเวลาที่กระบวนการทำงานอยู่ ถึงแม้ว่ากระบวนการนี้จะใช้เครื่องพิมพ์เฉพาะในตอนท้ายของการทำงานเท่านั้น

สำหรับวิธีที่ 2 ให้กระบวนการร้องขอทรัพยากรในตอนเริ่มต้น แค่เครื่องขับเทปและแฟ้มข้อมูลในดิสก์ โดยเมื่อได้รับทรัพยากรแล้ว กระบวนการจะคัดลอกข้อมูลจากเทปลงไปสู่ดิสก์ จากนั้นก็จะคืนทั้งเครื่องขับเทปและแฟ้มข้อมูลในดิสก์กลับสู่ระบบ จากนั้นกระบวนการก็จะต้องร้องขอแฟ้มข้อมูลในดิสก์และเครื่องพิมพ์ใหม่อีกครั้งหนึ่ง เมื่อพิมพ์เสร็จเรียบร้อยแล้วกระบวนการก็จะคืนทรัพยากรกลับสู่ระบบ เป็นอันสิ้นสุดการทำงานของกระบวนการ

วิธีการแรกมีข้อเสีย คือ การใช้ทรัพยากรจะมีประสิทธิผลต่ำมาก เพราะกระบวนการจำเป็นต้องร้องขอและถือครองทรัพยากรไว้ทั้งหมดตลอดช่วงเวลาการทำงาน ทั้ง ๆ ที่การใช้ทรัพยากรแต่ละตัวอาจเป็นเพียงช่วงเวลานั้น ๆ ก็ตาม นอกจากนั้นอาจมีปัญหาการรอไม่จบ (Starvation) อีกด้วย โดยถ้ามีบางกระบวนการต้องการใช้ทรัพยากร หลาย ๆ ตัว อาจต้องรอคอยอย่างไม่มีที่สิ้นสุด เพราะทรัพยากรตัวหนึ่งในจำนวนที่ต้องการอาจมีกระบวนการอื่นใช้อยู่ส่วนวิธีการหลังก็มีข้อเสีย คือ ต้องคืนทรัพยากรที่ถือครองอยู่ เพื่อที่จะร้องขอกลับมาใหม่อีกร่วมกับทรัพยากรตัวใหม่ ทำให้เสียเวลาโดยเปล่าประโยชน์

3.ไม่แทรกกลางคัน (No Preemption) ถ้ากระบวนการใด ๆ ที่กำลังถือครองทรัพยากรบางส่วนอยู่ และ ร้องขอทรัพยากรเพิ่ม และระบบยังไม่สามารถจัดสรรให้ได้ในทันที แสดงว่ากระบวนการที่ร้องขอจะต้องรอ เราจะให้ทรัพยากรทั้งหมดที่กระบวนการนี้ถือครองอยู่ถูกแทรกกลางคัน นั่นคือ ทรัพยากรที่กระบวนการนี้ถือครองอยู่ทั้งหมดจะถูกปล่อยคืนสู่ระบบโดยปริยาย กระบวนการที่ถูกแทรกกลางคันนี้จะต้องรอคอยทรัพยากร ทั้งที่ร้องขอไว้ตั้งแต่แรกและที่ถูกแทรกกลางคันไป ก่อนที่ไม่จะสามารถทำงานต่อไปได้

หรืออาจจะกล่าวได้ว่า ถ้ามีกระบวนการหนึ่งได้ร้องขอทรัพยากรบางส่วนจากระบบ ในตอนแรกระบบจะตรวจสอบว่า ทรัพยากรที่ร้องขอนั้นว่างอยู่หรือไม่ ถ้าว่างอยู่ ระบบก็จะจัดสรรทรัพยากรเหล่านั้นให้แก่กระบวนการ แต่ถ้าทรัพยากรที่ถูกร้องขอนั้นไม่ว่าง ระบบจะตรวจดูก่อนว่าทรัพยากรนั้นไม่ว่างเนื่องจากอะไร ถ้าไม่ว่างเนื่องจากกำลังถูกถือครองโดยกระบวนการอื่น ซึ่งกำลังรอคอยทรัพยากรเพิ่มอยู่ ระบบจะทำการแทรกกลางคันทรัพยากรทั้งหมดของกระบวนการนั้นและจัดสรรทรัพยากรที่ได้มาให้แก่กระบวนการที่ร้องขอ แต่ถ้าทรัพยากรที่ไม่ว่างนั้นไม่ได้ถูกถือครองโดยกระบวนการอื่นที่กำลังรออยู่ ระบบก็จะให้กระบวนการที่ร้องขอทรัพยากรนั้นรอ และขณะที่รออยู่นั้น ทรัพยากรทั้งหมดที่กระบวนการนี้ถือครองอยู่อาจถูกแทรกกลางคันได้ เมื่อมี

กระบวนการอื่นร้องขอ และกระบวนการนี้จะสามารถกลับไปทำงานต่อได้ เมื่อได้รับจัดสรรทรัพยากรที่ร้องขอและได้รับทรัพยากรที่อาจถูกแทรกกลางคันไปคืนทั้งหมดเสียก่อน

วิธีการนี้มักใช้กับทรัพยากรที่สามารถเก็บค่าสถานะและติดตั้งค่ากลับคืนมาได้ง่าย เช่นค่าในรีจิสเตอร์ของ CPU และเนื้อที่ในหน่วยความจำหลัก แต่จะไม่สามารถใช้กับทรัพยากรทั่ว ๆ ไปได้ เช่น เครื่องพิมพ์ และหน่วยขับเทป

4. การรอคอยเป็นวงรอบ (Circular Wait) เพื่อให้แน่ใจว่าไม่เกิดเงื่อนไขการรอคอยเป็นวงรอบ เพื่อเป็นการป้องกันการเกิดติดตาย ซึ่งสามารถทำได้โดยการกำหนดลำดับของทรัพยากรทั้งหมดในระบบ และกำหนดให้กระบวนการต้องร้องขอใช้ทรัพยากร เรียงตามลำดับ

กำหนดให้ **R** เป็นเซตของทรัพยากรในระบบ โดยที่ $R = \{ R_1 , R_2 , \dots , R_m \}$ และกำหนดให้ทรัพยากรแต่ละประเภทมีค่าเลขลำดับที่เป็นเลขจำนวนเต็มที่ไม่ซ้ำกัน เขียนแทนด้วยฟังก์ชัน $F(R_i)$ เพื่อให้เราสามารถเปรียบเทียบทรัพยากร 2 ตัวนี้ว่าตัวใดมีลำดับก่อนหรือหลัง ตัวอย่างเช่น ถ้าเซตของทรัพยากร R={ เครื่องขับเทป ,เครื่องขับจานบันทึก, และเครื่องพิมพ์} ดังนั้นค่าเลขลำดับ $F(R_j)$ อาจจะถูกกำหนดได้ ดังนี้คือ

F (เครื่องขับเทป) = 1

F (เครื่องขับดิสก์) = 5

F (เครื่องพิมพ์) = 12

และการร้องขอทรัพยากรในระบบ กำหนดวิธีการไว้ดังนี้ กระบวนการแต่ละตัวสามารถร้องขอทรัพยากรได้ในลำดับที่เพิ่มขึ้นเท่านั้น คือ เริ่มต้นกระบวนการอาจร้องขอทรัพยากรใด ๆ ก็ได้ เช่น ทรัพยากร R_i แต่ต่อจากนี้กระบวนการจะร้องขอทรัพยากร R_j ได้ก็ต่อเมื่อ $F(R_j) > F(R_i)$ ถ้าเป็นการร้องขอทรัพยากรประเภทเดียวกันหลาย ๆ ตัว กระบวนการจะต้องร้องขอทรัพยากรทีละตัว จากตัวอย่าง เลขลำดับทรัพยากรข้างต้น ถ้ากระบวนการหนึ่งต้องการใช้เครื่องขับเทปคือ F(R) = 1 และเครื่องพิมพ์คือ F(R) = 12 กระบวนการนั้นจะต้องร้องขอเครื่องขับเทปก่อน แล้วจึงร้องขอเครื่องพิมพ์ จะร้องขอเครื่องขับเทปก่อน ไม่ได้ระบบจะไม่อนุญาต

ในทางตรงกันข้าม ถ้ากระบวนการต้องการร้องขอทรัพยากรประเภท R_j กระบวนการจะต้องปล่อยทรัพยากร $\mathbf{R_i}$ ซึ่ง $F(R_i) \geq F(R_j)$ คืนสู่ระบบทุกตัวเสียก่อน เช่น ถือครอง R_5 อยู่และถ้าต้องการใช้ R_1 ต้องคืน R_5 ก่อนเพราะว่า $R_5 \geq R_1$

จากกำหนดที่กล่าวมา จะเห็นว่าเงื่อนไขการรอคอยเป็นวงรอบจะไม่สามารถเกิดขึ้นโดยที่สามารถพิสูจน์โดยวิธียกสิ่งตรงข้ามได้ ดังนี้

สมมติให้กระบวนการที่เกิดการรอคอยเป็นวงรอบในระบบ คือ $\{ P_1 , P_2 , \dots , P_n \}$ โดยที่กระบวนการ P_1 รอคอยทรัพยากร R_1 ซึ่งกำลังถูกถือครองโดยกระบวนการ P_{i+1} ตามลำดับเป็นวงจร

เนื่องจากกระบวนการ P_{i+1} ถือครองทรัพยากร R_1 อยู่ ขณะที่ร้องขอทรัพยากร R_{i+1} เราจะได้ว่า $F(R_1) < \ldots < F(R_n) < F(R_0)$ ดังนั้น $F(R_0) < F(R_0)$ ซึ่งเป็นไปไม่ได้ ดันั้นสรุปไดว่าว่าไม่มีการรอคอยเป็นวงรอบในระบบ

พึงสังเกตว่า การกำหนดค่าเลขลำดับของทรัพยากร ควรเรียงตามลำดับการใช้งานตามปกติในระบบ เช่น ปกติเรามักใช้เครื่องขับเทปก่อนเครื่องพิมพ์เสมอ จึงควรกำหนดลำดับให้ F (เครื่องขับเทป) < F(เครื่องพิมพ์)

การหลีกเลี่ยงการเกิดการติดตาย

จากที่กล่าวมาข้างต้นเป็นการป้องกันการเกิดติดตาย (Deadlock Prevention) นั้น เป็นการป้องกันติดตายไม่ให้เกิดขึ้น โดยการสร้างข้อกำหนดในการร้องขอทรัพยากร เพื่อให้แน่ใจว่าเงื่อนไขที่ทำให้เกิดติดตายข้อใดข้อหนึ่ง ไม่เกิดขึ้นอย่างแน่นอน ซึ่งข้อกำหนดเหล่านั้นอาจมีผลกระทบต่อประสิทธิภาพการใช้งานทรัพยากรในระบบ คือ การใช้งานทรัพยากรในระบบ อาจจะมีประสิทธิภาพต่ำลง ซึ่งจะเป็นผลกระทบต่อประสิทธิภาพการทำงานของ CPU คืออัตรางานเสร็จต่อหน่วยเวลา (Throughput) ของระบบการลดลง ดังนั้นจะมีวิธีการที่จะใช้หลีกเลี่ยงการเกิดติดตาย ก็คือ เราต้องมีข้อมูลเกี่ยวกับการร้องขอทรัพยากรของกระบวนการต่าง ๆ ในระบบโดยรวม ตัวอย่างเช่น ในระบบมีเครื่องขับเทป 1 เครื่อง และเครื่องพิมพ์ 1 เครื่องและมีกระบวนการ p1 และ p2 เราอาจทราบว่า กระบวนการ P1 จะร้องขอเครื่องขับเทปเป็นอันดับแรกและร้องขอเครื่องพิมพ์เป็นอันดับต่อไป แล้วก็จะปล่อยทรัพยากรทั้ง 2 ตัวนี้ กลับคืนสู่ระบบ และ กระบวนการ p2 จะร้องขอเครื่องพิมพ์เป็นอันดับแรก แล้วจึงร้องขอเครื่องขับเทปต่อไป จากข้อมูลเหล่านี้ทำให้ระบบทราบลำดับในการร้องขอทรัพยากรของแต่ละกระบวนการ ซึ่งจะช่วยให้ระบบสามารถตัดสินใจได้ว่า การร้องขอในแต่ละครั้งนั้นจะให้กระบวนการรอหรือไม่ เพื่อเป็นการหลีกเลี่ยงปัญหาติดตายที่อาจจะเกิดขึ้นได้

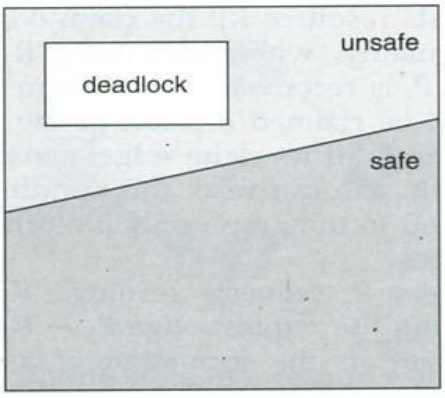
มีขั้นตอนวิธีต่าง ๆ หลายวิธีที่ทำให้รู้จำนวนและประเภทของข้อมูลที่ระบบต้องการใช้แต่มีวิธีหนึ่งที่ง่ายและเป็นประโยชน์มาก คือ วิธีการหลีกเลี่ยงติดตาย (Deadlock Avoidance) วิธีนี้แต่ละกระบวนการ ต้องแจ้งจำนวนทรัพยากรสูงสุดในแต่ละประเภทที่กระบวนการนั้นต้องการใช้ จากข้อมูลนี้ทำให้เราสามารถสร้างขั้นตอนวิธีที่จะรับประกันได้ว่า ระบบจะไม่มีทางเข้าสู่สถานะติดตายได้เลย ขั้นตอนวิธีนี้จะคอยตรวจดู สถานะของการจัดสรรทรัพยากร (Resource - Allocation State) ของระบบอยู่เสมอ ว่าจะไม่มีทางเกิดเงื่อนไขการรอคอยเป็นวงรอบขึ้นในระบบได้ โดยที่สถานะของการจัดสรรทรัพยากรนี้ประกอบด้วยจำนวนทรัพยากรทั้งหมดในระบบ จำนวนทรัพยากรที่จะถูกจัดสรรให้กับกระบวนการ และความต้องการสูงสุดของแต่ละกระบวนการ ระบบจะอยู่ใน

สถานะปลอดภัย (Safe State) ถ้ามีทางเลือกอย่างน้อย 1 ทาง ที่จะจัดสรรทรัพยากร ให้แต่ละกระบวนการได้ จนทำงานเสร็จทั้งหมดโดยไม่เกิดติดตาย

สถานะปลอดภัย (Safe State)

ระบบจะอยู่ในสถานะปลอดภัย (Safe State) ก็ต่อเมื่อมีลำดับการจัดสรรทรัพยากรอย่างปลอดภัยแก่กระบวนการ (Safe Sequence) โดยจะถือว่าลำดับของกระบวนการ $(P_1 , P_2 , \dots , P_n)$ เป็นลำดับที่ปลอดภัยสำหรับสถานะของการจัดสรรทรัพยากรปัจจุบัน ถ้าแต่ละกระบวนการ P_i สามารถได้รับทรัพยากรที่อาจร้องขอจากทรัพยากรที่ยังว่างอยู่ในระบบรวมกับของที่ P_j ถือครองอยู่ ($j < i$) นั่นคือ ถ้ากระบวนการ P_i ต้องการทรัพยากรที่ไม่อาจได้รับในทันที เนื่องจากมีกระบวนการ P_j ถือครองอยู่ จะทำให้ P_i ต้องรอจนกระทั่ง P_j ใช้งานเสร็จ ซึ่งเมื่อ P_j ใช้งานเสร็จแล้ว P_i ก็สามารถได้รับทรัพยากรทั้งหมดที่ต้องการ เพื่อที่จะใช้ทำงานจนเสร็จสมบูรณ์ได้ และ P_i จะปล่อยทรัพยากรที่ใช้เสร็จแล้วทั้งหมดกลับคืนสู่ระบบเมื่อสิ้นสุดการทำงาน เมื่อ P_i ทำงานเสร็จแล้ว กระบวนการ P_{i+1} สามารถได้รับทรัพยากรทั้งหมดที่ต้องการ และจะเป็นไปในทำนองเดียวกันกับกระบวนการอื่น ๆ แต่ถ้าไม่สามารถหาลำดับกระบวนการที่ปลอดภัยในระบบได้แสดงว่าระบบอยู่ในสถานะไม่ปลอดภัย (Unsafe State)

สถานะปลอดภัยเป็นสถานะที่ไม่เกิดติดตาย ส่วนสถานะไม่ปลอดภัยเป็นสถานะที่อาจเกิดติดตายได้ แต่ก็ไม่เสมอไปเพราะ สถานะไม่ปลอดภัยอาจจะไม่เกิดติดตายก็ได้ ดังรูปที่



รูปที่ **6**.4 ติดตายกับสถานะไม่ปลอดภัยและปลอดภัย

เมื่อไรก็ตามที่ระบบอยู่ในสถานะปลอดภัย ระบบจะสามารถหลีกเลี่ยงการเข้าสู่สถานะไม่ปลอดภัยได้ ซึ่งเป็นการหลีกเลี่ยงการเกิดติดตายได้ แต่ถ้าระบบอยู่ในสถานะไม่ปลอดภัยแล้ว ระบบจะไม่อาจหลีกเลี่ยงกระบวนการที่ทำให้เกิดติดตายได้

ตัวอย่างเช่น ระบบหนึ่งมีเครื่องขับเทป 12 ตัว และมีกระบวนการอยู่ในระบบ 3 กระบวนการ คือ P_0, P_1 และ P_2 โดยที่แต่ละกระบวนการ P_0, P_1 และ P_2 ต้องการใช้เครื่องขับเทปจำนวนมากที่สุด 10**,** **4** และ 9 เครื่อง ตามลำดับ ถ้า ที่เวลา T_0 กระบวนการ P_0, P_1 และ P_2 ได้รับเครื่องขับเทปกระบวนการละ 5**,** **2** และ 2 ตัวตามลำดับ ขณะนี้ยังมีเครื่องขับเทปว่างอยู่ 3 ตัวที่ให้กระบวนการใด ๆ สามารถเรียกใช้ได้ดัง ต่อไปนี้

กระบวนการ	ความต้องการสูงสุด	ความ ต้องการปัจจุบัน
P_0	10	5
P_1	4	2
P_2	9	2

ระบบอยู่ในสถานะปลอดภัย เมื่อ ที่ เวลา T_0 ลำดับกระบวนการคือ P_1 , P_0 , P_2 แสดงว่าระบบสามารถจัดสรรทรัพยากรให้กระบวนการ P_1 ได้ทันที ซึ่งทำให้ P_1 ทำงานเสร็จและคืนทรัพยากรทั้งหมดแก่ระบบ (ณ เวลานี้ระบบจะมีเครื่องขับเทปว่างเพิ่มอีก 5 ตัว) ดังนั้นเมื่อกระบวนการ P_0 ต้องการเครื่องขับเทปเพิ่มอีก 5 เครื่องก็สามารถทำได้ กระบวนการ P_0 จึงสามารถทำงานเสร็จได้ และคืนเครื่องขับเทปทั้งหมดสู่ระบบเมื่อใช้เสร็จ (จะทำให้ระบบมีเครื่องขับเทปว่าง 10 ตัว ดังนั้นกระบวนการ P_2 ซึ่งต้องการเครื่องขับเทปอีก 7 ตัวจะสามารถทำงานได้เสร็จ และเมื่อกระบวนการ P_2 ทำงานเสร็จแล้ว จะคืนเครื่องขับเทปทั้งหมดสู่ระบบ (ณ จุดนี้ระบบจะมีเครื่องขับเทปว่าง 12 ตัว)

ในการทำงานระบบอาจจะเปลี่ยนจากสถานะปลอดภัย ไปเป็นสถานะไม่ปลอดภัยได้ เช่น สมมติว่า ณ เวลา T_1 กระบวนการ P_2 ร้องขอเครื่องขับเทปเพิ่มอีก 1 ตัว และได้รับการจัดสรรจะทำให้สถานะของระบบกลายเป็นสถานะไม่ปลอดภัยทันที เนื่องจาก ณ จุดนี้จะมีกระบวนการ P_1 เพียงกระบวนการเดียวที่สามารถได้รับทรัพยากรทั้งหมดที่ต้องการ (4 ตัว) โดยการร้องขอเครื่องขับเทปที่ว่างอยู่ขณะนี้ 2 ตัว และเมื่อใช้งานเสร็จก็จะคืนเครื่องขับเทปกลับสู่ระบบ ทำให้ระบบมีเครื่องขับเทปว่างอยู่ 4 ตัว ซึ่งไม่เพียงพอสำหรับทั้งกระบวนการ P_0 และ P_2 โดยที่ P_0 อาจร้องขอได้อีก 5 ตัว ส่วน P_2 อาจร้องขอได้อีก 6 ตัว ทำให้เราไม่สามารถหาลำดับของกระบวนการที่ปลอดภัยได้ จะทำให้เกิดติดตายได้ เราจะสามารถหลีกเลี่ยงสถานะไม่ปลอดภัยได้ถ้าเราปล่อยให้กระบวนการ **P_2** รอจนกระทั่งกระบวนการ P_0 หรือ P_1 ทำงานเสร็จ และคืนเครื่องขับเทปกลับสู่ระบบ จากนั้นค่อยจัดสรรให้กระบวนการ P_2 ตามที่ขอ ก็จะไม่เกิดติดตาย

จากแนวคิดของสถานะปลอดภัย สามารถสร้างขั้นตอนวิธีการหลีกเลี่ยงติดตายได้ โดยเมื่อใดก็ตามที่กระบวนการร้องขอทรัพยากรเพิ่มและทรัพยากรยังมีว่างพอ ระบบต้องตัดสินใจว่าจะ

ให้ทรัพยากรตามที่ร้องขอทันทีหรือไม่ให้ โดยพิจารณาจากว่าถ้าจัดสรรให้ตามที่ร้องขอแล้วระบบจะยังคงอยู่ในสถานะปลอดภัยหรือไม่ ซึ่งจะประกันได้ว่า จะไม่เกิดติดตายขึ้นในระบบ

อัลกอริธึมของแบงเกอร์ (Banker's Algorithm)

การทำงานของอัลกอริธึม เมื่อมีกระบวนการเข้ามาทำในระบบ กระบวนการนั้นจะต้องแจ้งจำนวนทรัพยากรแต่ละประเภททั้งหมดที่ต้องการใช้ โดยที่จำนวนที่ต้องการนี้จะต้องไม่มากกว่าจำนวนทรัพยากรนั้น ๆ ที่มีอยู่จริงในระบบและเมื่อกระบวนการร้องขอทรัพยากร ระบบจะต้องพิจารณาว่าเมื่อจัดสรรทรัพยากรให้แต่ละกระบวนการแล้ว จะทำให้ระบบอยู่ในสถานะปลอดภัยหรือไม่ ถ้าอยู่ในสถานะปลอดภัย ระบบก็จะจัดสรรทรัพยากรให้ตามที่กระบวนการขอ แต่ถ้าไม่อยู่ในสถานะปลอดภัย ระบบก็จะไม่จัดสรรทรัพยากรให้กระบวนการที่ร้องขอก็จะต้องรอจนกว่ากระบวนการอื่นได้คืนทรัพยากรบางส่วนให้แก่ระบบจนเพียงพอ ระบบจึงจะจัดสรรทรัพยากรให้กระบวนการนั้นทำงานต่อไป ดังนั้นระบบต้องเก็บโครงสร้างข้อมูลหลายตัวเพื่อใช้ในอัลกอริธึมของแบงเกอร์นี้ โครงสร้างข้อมูลนี้จะเป็นตัวบอกสถานะของการจัดสรรทรัพยากรในระบบให้กับแต่ละกระบวนการ โดยกำหนดให้

n เป็นจำนวนกระบวนการในระบบ

m เป็นจำนวนของประเภททรัพยากร

โครงสร้างข้อมูลที่ใช้อัลกอริธึมของแบงเกอร์ มีดังนี้

ทรัพยากรที่ว่าง หรือ Available : เป็นเลขจำนวนเต็ม m เก็บค่าจำนวนทรัพยากรแต่ละประเภท ที่ว่างเช่น Available[j] = k หมายถึง ทรัพยากรประเภท R_j มีจำนวนทรัพยากรว่างอยู่ **k** ตัว

ทรัพยากรที่มีทั้งหมด หรือ Max : เป็นเมตริกซ์ ของเลขจำนวนเต็ม ขนาด n x m เก็บค่าจำนวนสูงสุดของทรัพยากรแต่ละประเภทที่กระบวนการแต่ละตัวร้องขอเช่น Max[i,j] = k หมายถึง กระบวนการ P_i ต้องการทรัพยากรประเภท R_j มากที่สุด k ตัว

ทรัพยากรที่จัดสรรแล้ว หรือ Allocation : เป็นเมตริกซ์ ของเลขจำนวนเต็ม ขนาด n x m เก็บค่าจำนวนทรัพยากรแต่ละประเภทที่กระบวนการแต่ละตัว กำลังใช้อยู่ เช่น Allocation[i,j] = k หมายถึง กระบวนการ P_i กำลังใช้ทรัพยากรประเภท R_j อยู่ k ตัว

ทรัพยากรที่ต้องการใช้ หรือ Need : เป็น เมตริกซ์ ของเลขจำนวนเต็ม ขนาด n x m เก็บค่าจำนวนทรัพยากรแต่ละประเภท ที่กระบวนการแต่ละตัวอาจร้องขอเพิ่มอีกได้ เช่น Need[i,j] = k หมายถึง กระบวนการ P_i อาจร้องขอทรัพยากรประเภท R_j ได้อีก k ตัว จะเห็นว่า Need[i,j] = Max[i,j] - Allocation[i,j]

อัลกอริธึมตรวจสอบสถานะปลอดภัย (Safety Algorithm)

ขั้นตอนวิธีในการตรวจสอบ ระบบจะอยู่ในสถานะปลอดภัยหรือไม่ เป็นดังนี้

1. กำหนดให้ Work และFinish มีค่าเป็น m และ n ตามลำดับ และกำหนดค่าเริ่มต้นดังนี้

```
Work : = Available;
For i   : = l To n Do
        Finish[i] : = FALSE;
```

2. ให้ i = 1

```
WHILE i ≤ n Do BEGIN
    IF Finish[i] = FALSE AND Need[i] ≤ Work
        THEN BEGIN
                Work : = Work + Allocation[i];
                Finish[i] : = TRUE;
                i  : = i + 1 ; END
    ELSE i : = i + 1;
END
```

3. IF some Finish[i] = FALSE THEN “unsafe” ELSE “safe”

อัลกอริธึมนี้ต้องใช้สูตร $m \times n^2$ เพื่อหาว่าอยู่ในสถานะปลอดภัยหรือไม่

อัลกอริธึมร้องขอทรัพยากร (Resource - Request Algorithm)

กำหนดให้ $\mathbf{Request_i}$ หมายถึงกระบวนการ $\mathbf{P_i}$ ร้องขอทรัพยากรประเภท $\mathbf{R_j}$ ถ้าให้ $Request_i = k$ เป็นค่าที่กระบวนการ $\mathbf{P_i}$ แสดงคำร้องทรัพยากรประเภท $\mathbf{R_j}$ เป็นจำนวน **k** ตัว

เมื่อกระบวนการ $\mathbf{P_i}$ ร้องขอทรัพยากร ระบบจะจัดการตามอัลกอริธึมร้องขอทรัพยากร ดังนี้

1. ถ้า $Request_i > Need_i$ ระบบจะแจ้งว่าผิดพลาด “กระบวนการขอทรัพยากรมากกว่าที่ระบุ” แล้วกระบวนการจะออกจากระบบแต่ถ้า $Request_i \leq Need_i$ แล้วจะไปทำงานที่ข้อ 2

2. ถ้า $Request_i > Avaiable$ แล้วให้ P_i รอจนกว่าทรัพยากรที่ร้องขอจะว่าง จากนั้นจึงไปทำงานในข้อ 3 แต่ถ้า $Request_i \leq Avaiable$ แล้ว จะไปทำงานในข้อ 3

3. ระบบได้ทำการจัดสรรทรัพยากรให้ตามที่กระบวนการ P_i ร้องขอมา โดยระบบจะมีสถานะเปลี่ยนไปตามอัลกอริธึม ดังนี้

Available : = Available - $Request_i$;

$Allocation_i$: = $Allocation_i$ + $Request_i$;

$Need_i$: = $Need_i$ - $Request_i$**;**

แล้วตรวจสอบดูว่า สถานะของการจัดสรรทรัพยากรขณะนี้เป็นสถานะปลอดภัยหรือไม่ ถ้าเป็นระบบ ก็จะจัดสรรทรัพยากรให้ตามที่สมมติทันที แต่ถ้าระบบอยู่ในสถานะไม่ปลอดภัยแล้ว ระบบก็จะให้กระบวนการ $\mathbf{P_i}$ รอ และถอยกลับไปอยู่ในสถานะเดิม

ตัวอย่าง ระบบหนึ่งมีกระบวนการอยู่ 5 ตัว คือ P_0 ,P_1 ,P_2 ,P_3 และP_4 มีทรัพยากรอยู่ 3 ประเภท คือ A,B และC โดยที่แต่ละประเภทมีจำนวนทรัพยากร 10**,**5 และ 7 ตามลำดับดังนี้

	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	

และเนื่องจากเมทริกซ์ Need เกิดจาก Max – Allocation ดังนั้นจะได้ว่า

Process	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

และเราพบว่าระบบอยู่ในสถานะปลอดภัย เนื่องจากกระบวนการอาจทำงานได้ตามลำดับของกระบวนการ P_1 , P_3 , P_4 , P_2 , P_0 ซึ่งเป็นไปตามเงื่อนไขของสถานะปลอดภัย

สมมติว่ากระบวนการ P_1 ร้องขอทรัพยากรประเภท **A** เพิ่มอย่างละ 1 และ **C** เพิ่มอย่างละ 2 ตัว ดังนั้น $Request_1$ = (1,0,2) ระบบจะตัดสินใจว่าจะอนุญาตให้ทรัพยากรแก่กระบวนการตามที่ร้องขอหรือไม่ตามขั้นตอนดังนี้

1 $Request_1 \leq Need_1$ เนื่องจาก $(1,0,2) \leq (1,2,2)$

2. $Request_1 \leq Avaiable$ เนื่องจาก $(1,0,2) \leq (3,3,2)$

ถ้าทั้ง 2 ขั้นเป็นจริง ระบบก็จะจัดสรรทรัพยากรให้กระบวนการตามที่ร้องขอ ทำให้ระบบมีสถานะใหม่ดังนี้

	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	

แล้วจะตรวจสอบว่าสถานะใหม่ว่าเป็นสถานะปลอดภัยหรือไม่ โดยใช้อัลกอริธึมตรวจสอบสถานะปลอดภัย (Safety Algorithm) จะพบว่ากระบวนการอาจทำงานได้ตามลำดับของกระบวนการ P_1, P_2, P_3, P_0, P_2 ซึ่งเป็นไปตามเงื่อนไขของสถานะปลอดภัย เมื่อเป็นเช่นนั้น ระบบจะสามารถอนุญาตการร้องขอทรัพยากรทั้งหมดของกระบวนการ P ได้

ในบางครั้งระบบอาจไม่อนุญาตการร้องขอของกระบวนการได้ เช่น จากตัวอย่างเดิมขั้นต้น ถ้า P_4 ร้องขอทรัพยากร (3,3,0) เพิ่ม ระบบอาจไม่อนุญาตให้ได้ เพราะมีทรัพยากรไม่พอที่จะจัดสรรให้ หรือถ้า **P** ร้องขอทรัพยากร (0,2,0) เพิ่ม ระบบก็จะไม่อนุญาตเช่นเดียวกัน แม้ว่าจะมีทรัพยากรพอ เพราะว่าเมื่อระบบลองสมมติว่าได้จัดสรรทรัพยากรให้ตามที่ขอและพบว่าสถานะใหม่เป็นสถานะไม่ปลอดภัย

การตรวจหาการติดตาย

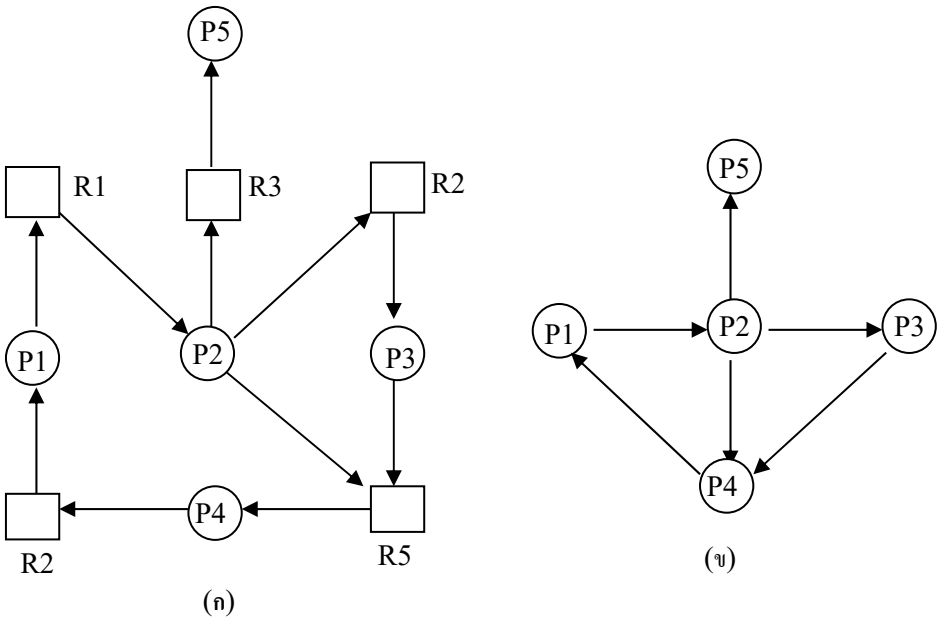
ถ้าในระบบปฏิบัติการไม่มีการป้องกันหรือหลีกเลี่ยงการติดตายแล้ว ในที่สุดระบบก็อาจจะตกอยู่ในสถานะติดตายได้ ดังนั้นระบบจึงต้องมีวิธีอื่นทดแทน คือ

- ขั้นตอนวิธีที่จะตรวจสอบหาติดตายในระบบว่า เกิดขึ้นแล้วหรือยัง
- ขั้นตอนวิธีในการแก้ไขติดตาย

ระบบที่มีทรัพยากรแต่ละประเภทเพียงตัวเดียว

ใช้กราฟการจัดสรรทรัพยากรมาทำให้เป็น กราฟการรอคอยทรัพยากร (Wait-for-Graph) การแปลงสภาพทำโดยเอาสี่เหลี่ยมที่แทนทรัพยากรออก และยุบรวมลูกศรเข้าด้วยกัน ดังนี้

ถ้าลูกศรจากPiไป P_jในกราฟการรอคอยทรัพยากร แสดงว่า P_i กำลังรอทรัพยากรซึ่ง P_j ถือครองอยู่ หรืออาจกล่าวได้ว่าจะมีลูกศร $P_i \rightarrow P_j$ ในกราฟการรอคอยทรัพยากรที่แปลงมาก็ต่อเมื่อมีลูกศร $P_i \rightarrow R_q$ และ $R_q \rightarrow P_j$ในกราฟการจัดสรรทรัพยากรต้นแบบ ตัวอย่างดังรูปที่ 6.5



รูปที่ 6.5 (ก) เป็นต้นแบบกราฟการจัดสรรทรัพยากร
(ข) แปลงเป็นกราฟการรอคอยทรัพยากร

ถ้าในระบบมีวงจรในกราฟการรอคอยทรัพยากร ระบบก็จะเกิดติดตาย และถ้าเกิดติดตายแล้ว ก็แสดงว่ามีวงจรในกราฟการรอคอยทรัพยากร ระบบต้องเก็บข้อมูลของกราฟการรอคอยทรัพยากรไว้ และใช้อัลกอริธึมการตรวจสอบหาวงจรในกราฟ เพื่อตรวจสอบหาติดตายในระบบ ต้องคอยตรวจหาติดตายทุก ๆ ช่วงเวลาที่กำหนด

ระบบที่มีทรัพยากรแต่ละประเภทหลายตัว

อัลกอริธึมการตรวจสอบหาติดตายนี้คล้ายกับอัลกอริธึมของแบงเกอร์ (Banker's Algorithm) จำเป็นต้องใช้โครงสร้างของข้อมูลทำนองเดียวกันดังต่อไปนี้

Available : เป็นเลขจำนวนเต็ม **m** แสดงจำนวนทรัพยากรแต่ละชนิดที่ยังว่างอยู่ ไม่ถูกใช้โดยกระบวนการใด

Allocation : เป็นเมทริกซ์ **n x m** ใช้เก็บค่าจำนวนทรัพยากรแต่ละชนิดที่กระบวนการแต่ละตัวถือครองอยู่

Request : เป็นเมทริกซ์ **n x m** ใช้เก็บค่าจำนวนทรัพยากรแต่ละชนิดที่กระบวนการแต่ละตัวร้องขอ

เครื่องหมายต่าง ๆ ที่ใช้ในอัลกอริธึมนี้จะเหมือนกับเครื่องหมายที่ใช้ในอัลกอริธึมของแบงเกอร์ (Banker's Algorithm) ที่ได้กล่าวมาแล้วข้างต้น อัลกอริธึมในการตรวจหาติดตายนี้ เริ่มจากการหาลำดับการอนุญาตให้กระบวนการใช้ทรัพยากรตามที่เป็นไปได้ ในการทำจนเสร็จ

อัลกอริธึมการตรวจหาติดตาย

1 .Work : = Available;

2. FOR i : = 1 TO n DO

```
IF Allocation[i] ≠0
      THEN Finish[i] : = FALSE
      ELSE Finish[i] : = TRUE;
```

3. i : =1;

```
WHILE i ≤ n Do BEGIN
IF Finish[i] = FALSE AND Request[i] ≤ Work
      THEN BEGIN
                  Work : = Work + Allocation[i];
                  Finish[i] : = TRUE;
                  i : = i + 1; END
       ELSE i : = i + 1;
END;
```

4. FOR i = 1 TO n DO

```
IF Finish[i] = FALSE THEN process Pi is in a deadlocked.
```

5. IF Finish[i] = TRUE ทั้งหมด แสดงว่าขณะนี้ระบบไม่เกิดติดตาย

เราต้องคืนทรัพยากรที่กระบวนการ P_i ถือครองอยู่ให้กับระบบ เมื่อ Request[i] $\leq$ Work เนื่องจากขณะนี้ไม่มีกระบวนการ $\mathbf{P_i}$ ใดอยู่ในติดตาย ดังนั้นเราจึงสามารถให้ $\mathbf{P_i}$ ทำงานได้จนเสร็จ และเมื่อเสร็จแล้วก็จะคืนทรัพยากรที่ถือครองให้แก่ระบบ แต่ถ้าไม่เป็นตามนี้อาจเกิดติดตายได้ในอนาคต ซึ่งเราก็สามารถตรวจดูได้ ในการตรวจหาติดตายในครั้งต่อไปตามอัลกอริธึมเดียวกันนี้

ตัวอย่าง ให้ระบบที่มีการบวนการ 5 กระบวนการ P_0 ,P_1,P_2,P_3 และ P_4 และมีทรัพยากร 3 ชนิด แต่ละชนิดมีหลายตัว คือ ชนิด A มี 7 ตัว ชนิด B มี 2 ตัวและชนิด C มี 6 ตัว ณ เวลา T_0 ระบบอยู่ในสถานะดังนี้

process	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	

จากอัลกอริธึมสมารถสรุปได้ว่า ระบบไม่ได้อยู่ในสถานะติดตายคือไม่เกิดติดตายโดย

มีลำดับการทำงานของกระบวนการคือ P_0 ,P_2 ,P_3 ,P_1, P_4 ซึ่งจะให้ Finish[i] = TRUE ทั้งหมด

ถ้ากระบวนการ P_2 ร้องขอทรัพยากรประเภท C เพิ่มอีก 1 ตัว สถานะของระบบจะเป็นดังนี้

process	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

จะพบว่าในระบบเกิดติดตายเกิดขึ้น เพราะถึงแม้ว่า P_0 ทำงานเสร็จ แล้วคืนทรัพยากรชนิด B สู่ระบบแต่ทรัพยากรที่ว่างอยู่มีไม่พอตามคำร้องขอของกระบวนการอื่น ๆ กระบวนการเหล่านั้นจึงไม่อาจทำงานต่อได้ ดังนั้นทุกกระบวนการ **P_1, P_2, P_3** และ**P_4** จึงตกอยู่ในสภาพติดตาย

การใช้อัลกอริธึมตรวจหาติดตาย (Detection – Algorithm Usage)

ระบบจะเรียกใช้ อัลกอริธึมตรวจกาติดตาย (Detection – Algorithm) เมื่อมีปัจจัยต่อไปนี้

1. ความถี่ในการเกิดติดตาย

2. เมื่อเกิดติดตายมีกระบวนการมีจำนวนกระบวนการที่ติดอยู่

ถ้าในระบบเกิดติดตายบ่อย ต้องมีการเลือกอัลกอริธึมที่ใช้ตรวจหาติดตายบ่อย ๆ ด้วย เพราะเมื่อเกิดติดตาย ทรัพยากรต่าง ๆ ในระบบจะไม่สามารถถูกใช้ได้จนกว่าจะแก้ไขติดตายนั้น

และถ้าระบบปล่อยติดตายไว้ โดยยังไม่แก้ไขจะทำให้ติดตายขยายขึ้นเรื่อย ๆ จนกระทั่งระบบ อื่น ๆ อาจจะไม่สามารถทำงานได้

ติดตายจะเกิดขึ้นเนื่องจากมีบางกระบวนการร้องขอใช้ทรัพยากรแล้วระบบไม่สามารถจัดสรรให้ได้ทันที ดังนั้นเราจึงใช้อัลกอริธึมตรวจหาติดตาย โดยตรวจทุกครั้งที่มีเหตุการณ์นี้เกิดขึ้น ซึ่งจะเกิดติดตายหรือไม่เกิดติดตายก็ต้องตรวจหาติดตาย วิธีนี้นอกจากเราจะตรวจพบได้รวดเร็วทันเหตุการณ์แล้ว ยังทำให้ระบุชัดด้วยว่า กระบวนการใดทำให้เกิดติดตายนี้ได้ (ความจริงแล้วติดตายเป็นวงจรของกระบวนการซึ่งผูกติดกันเป็นวงกลม ทุก ๆ กระบวนการที่ติดอยู่ย่อมเป็นสาเหตุให้กิดติดตายได้เท่าเทียมกัน) ถ้าการร้องขอแต่ละครั้งสามารถขอทรัพยากรได้หลายชนิดพร้อม ๆ กัน กระบวนการหนึ่งอาจร้องขอทรัพยากรแล้วทำให้เกิดติดตายขึ้นพร้อม ๆ กันหลายวงจรก็ได้

ถ้าระบบมีการ การตรวจหาติดตายทุก ๆ การร้องขอใช้ทรัพยากร จะเกิดการการตรวจหาติดตายบ่อย ๆ มาก ทำให้เสียค่าใช้จ่าย หรือเวลา เพื่อที่จะประหยัดค่าใช้จ่ายและเวลา เราอาจตรวจหาติดตาย ทุก ๆ ช่วงเวลาแทนทุก ๆ การร้องขอใช้ทรัพยากร เช่น ทุก ๆ 1 ชั่วโมง หรือเมื่อประสิทธิผลของการใช้หน่วยประมวลผลกลางต่ำกว่าหรือลดลงเหลือ 40% โดยปรกติติดตายจะทำให้ระบบทำงานได้น้อยลง และเป็นผลให้ประสิทธิผลของการใช้หน่วยประมวลผลกลางต่ำลงด้วย ถ้าระบบตรวจหาติดตายและพบว่าเกิดติดตายขึ้นในระบบมากกว่า 1 วงจร เราไม่สามารถรู้ว่ากระบวนการใดเป็นตัวทำให้เกิดติดตาย

การแก้ไขการติดตาย

เมื่อตรวจพบว่าในระบบเกิดติดตายขึ้น ระบบอาจมีวิธีจัดการได้ 2 วิธี คือ

วิธีแรก แจ้งผู้ควบคุมระบบ ให้ทราบว่าขณะนี้ในระบบเกิดติดตายขึ้น โดยปล่อยให้ผู้ควบคุมระบบจัดการแก้ไขติดตายเอง

วิธีที่ **2** ระบบดำเนินการแก้ไขติดตายเองอัตโนมัติ ซึ่งอาจทำได้ 2 วิธี คือ

- การยกเลิกกระบวนการ (Process Termination) ยกเลิกกระบวนการที่อยู่ในติดตายบางกระบวนการเพื่อให้หายจากการเกิดติดตาย

- การแทรกกลางคัน (Resource Preemption) ระบบให้มีการแทรกกลางคันของทรัพยากรบางตัวที่ติดอยู่ในติดตาย ได้เพื่อให้หายจากการเกิดติดตาย

การยกเลิกกระบวนการ (Process Termination)

การแก้ไขติดตายโดยการยกเลิกกระบวนการในติดตายมีอยู่ 2 วิธี

1. ยกเลิกทุกกระบวนการที่ติดอยู่ในติดตาย วิธีนี้จะแก้ไขติดตายได้แน่นอน แต่จะมีผลกระทบกับกระบวนการที่เข้าประมวลผลนานแล้วและใกล้จะเสร็จ จะเสียค่าใช้จ่ายสูง เพราะว่า

กระบวนการที่เข้ามาทำงานนานแล้ว แต่กลับถูกยกเลิกการทำงานโดยผลลัพธ์ที่ได้จากการทำงานจะถูกยกเลิกด้วยและอาจจะต้องมาเริ่มต้นทำงานใหม่

2. ยกเลิกกระบวนการในติดตายทีละกระบวนการ จนหายจากการเกิดติดตาย ระบบกลับสู่สภาวะการทำงานปรกติ วิธีนี้อาจมีค่าใช้จ่ายสูงเช่นกัน เพราะว่าหลังจากที่ยกเลิกกระบวนการหนึ่งแล้วระบบจะต้องทำการตรวจหาติดตายอีก เพื่อตรวจสอบว่าระบบยังอยู่ในสถานะติดตายหรือไม่ ระบบจะต้องทำการตรวจหาติดตาย และทำการยกเลิก จนกระทั่งหายจากการเกิดติดตาย

การยกเลิกกระบวนการไม่ใช่เรื่องง่าย ๆ เช่น ถ้ากระบวนการกำลังแก้ไขแฟ้มข้อมูลอยู่ การยกเลิกกระบวนการอาจก่อให้เกิดความผิดพลาดของข้อมูลในแฟ้มข้อมูลได้เช่น เดียวกันกับกระบวนการกำลังพิมพ์ข้อมูลออกสู่เครื่องพิมพ์ เมื่อกระบวนการนี้ถูกยกเลิก ระบบจะต้องกำหนดสถานะของเครื่องพิมพ์ใหม่ให้ถูกต้องเสียก่อนที่จะพิมพ์งานต่อไป

การยกเลิกกระบวนการทีละตัว จะต้องเลือกว่ากระบวนการใดเหมาะสมที่จะถูกยกเลิกเพื่อที่จะทำให้ระบบหายจากติดตายและเข้าการทำงานตามปรกติ กระบวนการที่จะถูกยกเลิกจะต้องเป็นกระบวนการที่ทำให้เกิดค่าใช้จ่ายน้อยที่สุด ดังนั้นการยกเลิกกระบวนการจึงพิจารณาจากปัจจัยต่อไปนี้

1. ลำดับความสำคัญของกระบวนการ (Priority) กระบวนการที่มีลำดับความสำคัญต่ำกว่าจะถูกยกเลิก

2. เวลาทำงานของกระบวนการ กระบวนการนั้นทำงานมานานเท่าไรแล้วและเหลือเวลาอีกนานเท่าไรจึงจะเสร็จ กระบวนการที่เข้ามาในระบบไม่นานจะถูกยกเลิก

3. กระบวนการนั้นได้ใช้ทรัพยากรประเภทใดไปจำนวนเท่าไร

4. กระบวนการต้องการทรัพยากรอีกเท่าไร จึงจะทำงานเสร็จ

5. จำนวนกระบวนการที่จะถูกยกเลิก

6. ประเภทของกระบวนการเป็นแบบใด แบบโต้ตอบหรือแบบกลุ่ม

การแทรกกลางคัน (Resource Preemption)

การแก้ปัญหาติดตายโดยการแทรกกลางคันนั้น จะทำโดยให้บางกระบวนการที่อยู่ในติดตายคืนทรัพยากรบางส่วนให้แก่ระบบ (ทั้งเต็มใจคืนและไม่เต็มใจคืน) เพื่อนำทรัพยากรนั้นไปจัดสรรให้กับกระบวนการอื่นที่ต้องการ จะกระทำการแทรกกลางคันไปแบบนี้ไปเรื่อย ๆ จนกว่าระบบจะหายจากติดตายและกลับเข้าสู่การทำงานตามปรกติ

ถ้าเลือกใช้วิธีแทรกกลางคันเราการแก้ปัญหาติดตายจะต้องพิจารณาสิ่งที่เกี่ยวข้องต่อไปนี้

1. การเลือกเป้าหมาย (Selection a victim) เลือกกระบวนการใดในติดตาย ที่จะถูกแทรกกลางคัน ที่เลือกแล้วจะเสียค่าใช้จ่ายน้อยที่สุด ปัจจัยในการพิจารณาค่าใช้จ่ายได้แก่ จำนวนทรัพยากรที่กระบวนการนั้นถือครองอยู่ และระยะเวลาที่กระบวนการนั้นได้ทำงานมาแล้ว

2. การถอยกลับ (Rollback) กระบวนการที่ถูกแทรกกลางคันนั้นไม่สามารถทำงานต่อไปได้ เพราะขาดทรัพยากรที่กระบวนการนั้นต้องการ ดังนั้นเราจะต้องให้กระบวนการนั้นถอยกลับไปอยู่ที่จุดที่ปลอดภัยและให้เริ่มทำงานใหม่อีกครั้งจากจุดนี้ เป็นเรื่องยากที่จะรู้ว่าปลอดภัยอยู่ตรงไหน ดังนั้นจึงให้กระบวนการนั้นถอยกลับไปยังจุดเริ่มต้นเท่ากับการยกเลิกกระบวนการ แล้วกลับไปเริ่มทำงานใหม่ แต่มีวิธีที่มีประสิทธิภาพกว่านี้ คือให้ถอยกลับไปที่ ที่จำเป็นในการแก้ไขติดตายเท่านั้น แต่จะมีปัญหาเกี่ยวกับเรื่องการเก็บข้อมูลเกี่ยวกับสถานะต่าง ๆ ของกระบวนการทั้งหมดที่กำลังทำงานอยู่

3. การรอแบบไม่จบ (Starvation) ถ้ากระบวนการหนึ่งถูกแทรกกลางคัน เราจะรับประกันได้อย่างไรว่ากระบวนการนั้น จะเกิดการรอแบบไม่จบ เช่นถ้ากระบวนการนั้นรอใช้ทรัพยากรที่กำลังใช้อยู่เสมอๆจากกระบวนการอื่นที่ได้รับการแทรกกลางคัน

ในระบบที่มีการเลือกเป้าหมาย โดยพิจารณาจากค่าใช้จ่าย อาจมีบางกระบวนการถูกเลือกเป็นเป้าหมายให้แทรกกลางคันเสมอ ๆ ทำให้กระบวนการนั้นไม่สามารถทำงานจนเสร็จได้ ดังนั้นการเลือกกระบวนการที่เป็นเป้าหมาย จะต้องแน่ใจว่า กระบวนการนั้นจะถูกเลือกเป็นเป้าหมายอย่างมีขอบเขตของจำนวนครั้งในการถูกเลือก โดยการรวมจำนวนครั้งในการถอยกลับของกระบวนการ

การจัดการปัญหาการติดตายโดยวิธีผสมผสาน

จากวิธีการแก้ปัญหาติดตายหลาย ๆ วิธีที่กล่าวมาแล้วไม่มีวิธีใดวิธีหนึ่งที่ดีที่สุดสำหรับจัดการปัญหาติดตายของระบบได้ จึงอาจใช้หลายวิธีผสมผสานกัน โดยเลือกวิธีที่เหมาะสมกับแต่ละลักษณะของติดตายของแต่ละระบบย่อยนั้น ๆ สามารถทำได้โดยจัดทรัพยากรในระบบให้เป็นชั้นเป็นพวก ตามลำดับความสำคัญแล้วจัดแบ่งกลุ่มทรัพยากรในแต่ละกลุ่ม เราอาจเลือกใช้วิธีที่จัดการติดตายที่แตกต่างกันไปตามความเหมาะสม

ระบบที่ใช้วิธีผสมผสานจะไม่เกิดติดตายสามารถแสดงให้เห็นได้ ดังนี้

1. ติดตายจะเกิดข้ามกลุ่มของทรัพยากรไม่ได้ เพราะเราใช้วิธีการจัดเรียงลำดับทรัพยากร

2. ไม่เกิดติดตายในแต่ละกลุ่ม เพราะเราเลือกวิธีจัดการติดตายตามวิธีต่าง ๆ ที่เหมาะสม ดังนั้นระบบโดยรวมจะไม่เกิดติดตาย

ตัวอย่าง

สมมติว่าระบบประกอบไปด้วยทรัพยากร 4 ประเภท คือ

1. ทรัพยากรภายใน (Internal resources) คือ ทรัพยากรภายในของระบบใช้ เช่น PCB (Process Control Block) และรีจีสเตอร์ (Register)

2. หน่วยความจำหลัก (Main Memory) ซึ่งผู้ใช้และระบบต้องใช้

3. อุปกรณ์ต่าง ๆ ในระบบ (Job Resources) เช่น อุปกรณ์ทางกายภาพ อุปกรณ์ทางตรรกะที่กระวนการต่าง ๆ ต้องใช้

4. หน่วยความจำสำรอง (Swappable Space) คือพื้นที่ในฮาร์ดดิสค์ (Backing Store) สำหรับจัดเก็บแต่ละงาน

การปัญหาติดตายในระบบนี้ โดยแบ่งกลุ่มของทรัพยากรเป็น 4 กลุ่มข้างต้นแต่ละกลุ่มใช้วิธีจัดการที่แตกต่างกันไปตามความเหมาะสมดังนี้

1. ทรัพยากรภายใน (Internal Resources) ป้องกันการเกิดติดตายโดย การจัดลำดับของทรัพยากรเนื่องจาก กระบวนการที่ร้องขอทรัพยากรเหล่านี้ ล้วนเป็นกระบวนการภายในของระบบจึงให้ใช้ตามลำดับ

2. หน่วยความจำหลัก (Main Memory) ป้องกันการเกิดติดตายโดยการให้มีการแทรกกลางคันได้ เมื่อมีกระบวนการต้องการใช้หน่วยความจำเพิ่ม และหน่วยความจำเต็มแล้ว สมารถย้ายงานออกจากหน่วยความจำหลักไปเก็บไว้ในหน่วยความจำสำรอง (Backing Store) ได้ง่าย

3. อุปกรณ์ต่าง ๆ ในระบบ (Job Resources) ป้องกันการเกิดติดตายโดย ในระบบใช้วิธีการหลีกเลี่ยง เพราะสามารถรู้ข้อมูลเกี่ยวกับความต้องการทรัพยากรสูงสุดของแต่ละงาน

4. หน่วยความจำสำรอง (Swappable Space) ป้องกันการเกิดติดตายโดย โดยการจัดสรรล่วงหน้า เพราะแต่ละงานจะถูกกำหนดจำนวนหน่วยความจำสำรองไว้แล้ว

ปฏิบัติการที่ 6

1. ให้นักศึกษาแต่ละกลุ่มทดลองเรียกใช้งานในระบบคอมพิวเตอร์หลาย ๆ งานจนเครื่องคอมพิวเตอร์เกิดการติดตาย ทำการแก้ปัญหาการติดตายด้วยวิธีต่าง ๆ ได้อย่างไรบ้าง
2. แต่ลกลุ่มแรกเปลี่ยนเรียนรู้กัน

คำถามท้ายบท

1. ติดตายคืออะไร สาเหตุที่ทำให้เกิดติดตายมีอะไรบ้าง
2. วิธีการหลัก ๆ ในการแก้แญหาติดตาย มีกี่วิธี อะไรบ้าง
3. จงอธิบายวิธีการป้องกันไม่ให้เกิดติดตาย
4. จงอธิบายอัลกอริธึมของการตรวจหาและแก้ไขติดตาย
5. ระบบหนึ่งมีกระบวนการอยู่ 3 ตัว คือ P_0 ,P_1 และP_2 มีทรัพยากรอยู่ 3 ประเภท คือ A,B และC โดยที่แต่ละประเภทมีจำนวนทรัพยากร 9 ,5 และ 4 ตามลำดับ

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

จงแสดงว่าการจัดสรรทรัพยากรนี้เกิดติดตายหรือไม่

6. จงอธิบายวิธีการแก้ไขติดตายโดยการยกเลิกกระบวนการ(Process Termination) ในติดตาย
7. จงอธิบายวิธีการแก้ไขติดตายโดยการแทรกกลางคัน (Resource Preemption)
8. จงอธิบายวิธีการจัดการปัญหาติดตายโดยวิธีผสมผสาน