รายงานการศึกษาบทความวิชาการ

เรื่อง

Deadlock

นำเสนอโดย

นายกันตินันท์ กิมกิติกุลวิไล 640910268

นางสาวอมรรัตน์ หาญเนืองนิตย์ 640910640

นายณชพล อุบล 640911019

รายงานเล่มนี้เป็นส่วนหนึ่งของรายวิชา 618344 วิศวกรรมระบบปฏิบัติการเบื้องต้น

สาขาวิศวกรรมอิเล็กทรอนิกส์และระบบคอมพิวเตอร์ ภาควิชาวิศวกรรมไฟฟ้า

คณะวิศวกรรมศาสตร์เทคโนโลยีอุตสาหกรรม มหาวิทยาลัยศิลปากร

ปีการศึกษา 2566

คำนำ

รายงานเล่มนี้เป็นส่วนหนึ่งของรายวิชา 618344 วิศวกรรมระบบปฏิบัติการเบื้องต้น รายงานการศึกษาบทความวิชาการ โดยศึกษาบทความและงานวิจัยและเขียนรายงานในหัวข้อ Deadlock ที่เป็นหนึ่งในปัญหาที่เกิดขึ้นกับระบบปฏิบัติการได้ จึงได้จัดทำรายงานเล่มนี้เพื่ออธิบายที่มาสาเหตุของการเกิด Deadlock รวมถึงบอกวิธีการป้องกันไม่ให้เกิดปัญหานี้

ทั้งนี้ทางคณะผู้จัดทำหวังเป็นอย่างยิ่งว่ารายงานฉบับนี้จะเกิดประโยชน์ให้กับผู้ที่สนใจ หากมีข้อเสนอแนะหรือข้อผิดพบาดประการใด คณะผู้จัดทำขออภัยมา ณ ที่นี้

คณะผู้จัดทำ

นักศึกษาชั้นปีที่ 3

สารบัญ

Deadlock

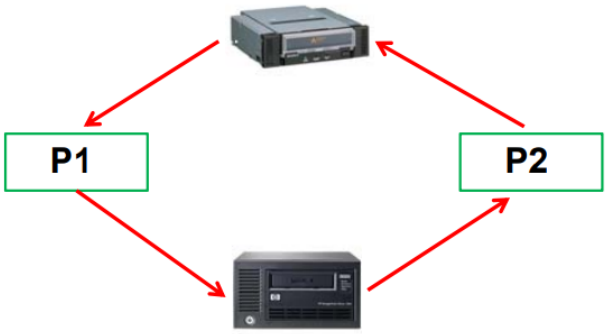
The Deadlock Problem :

process ร้องขอใช้ resource จากระบบ แต่ในขณะนั้น resource ที่ต้องการยังไม่ว่าง

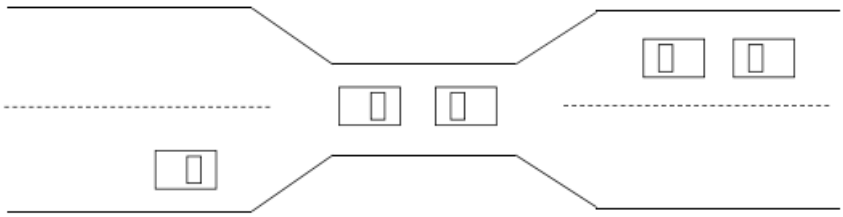
(อาจมี process อื่นๆ ครอบครองอยู่) process นั้นจะต้องรอไปเรื่อยๆ

Ex. - ในระบบมี tape drives สองตัว

- P1 และ P2 ใช้ tape drive อยู่อย่างละตัว และเรียกใช้เพิ่มอีก1ตัว



Bridge Crossing Example :



- สะพานรถวิ่งทางเดียว หากรถแต่ละด้านวิ่งขึ้นสะพานพร้อมกันก็จะไม่มีใครได้ขยับ
- Starvation อาจเกิดขึ้นได้ หากแต่ละด้านต่างรอให้อีกด้านนึงไปก่อน

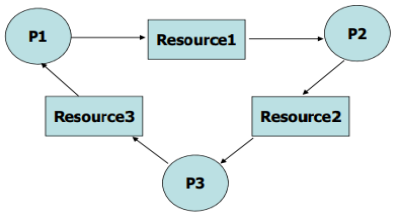
-ยกตัวอย่างการเกิด deadlock เช่น ถนน 2 เลนที่ต้องขึ้นสะพานเลนเดียว ถ้าทั้ง 2 ฝั่งไม่รอจะทำให้ไม่มีฝั่งไหนไปได้ วิธีแก้ ให้ด้านนึงรอรถอีกด้านไปก่อน ก็จะไม่เกิดอาการ deadlock

Example กรณีการเกิด Deadlock

1) Deadlock จากการร้องขอไฟล์ กระบวนการ A ร้องขอไฟล์ X และ B ร้องขอไฟล์ Y แต่ขณะเดียวกันถ้า A พยายามเข้าถึงไฟล์ Y และ B พยายามเข้าถึงไฟล์ X การร้องขอจะล้มเหลวและเกิดอาการ deadlock
2) Deadlock ในฐานข้อมูล ตาราง A ถูกล็อกโปรเซส X และ ตาราง B ถูกล็อกโปรเซส Y ถ้า X ต้องการเขียนข้อมูลในตาราง B และ Y ต้องการเขียนในตาราง A อาจทำให้เกิด Deadlock จาก X รอ A และ Y รอ B
3) Deadlock จากการจองใช้อุปกรณ์ที่ไม่สามารถใช้พร้อมกันได้ A และ B เลือกใช้อุปกรณ์ที่ไม่สามารถใช้พร้อมกันได้ ถ้าไม่มีการควบคุมที่เหมาะสม อาจจะทำให้เกิดสถานะ A รอ B และ B รอ A
4) Deadlock จากการจองใช้อุปกรณ์หลายๆชนิด เกิดจากกระบวนการ A ร้องขออุปกรณ์ 1 และ 2 กระบวนการ B ร้องขอ 2 และ 3 เนื่องจาก A B นั้นใช้งานอุปกรณ์ได้ทีละตัว A และ B อาจรอใช้อุปกรณ์ที่ 2 พร้อมกัน ทำให้มีโอกาศเกิด Deadlock ขึ้นได้
5) Deadlock ใน Spooling ยกตัวอย่างเครื่องพิมที่มีความจำ 2 หน่วย กระบวนการ A เขียนข้อมุลลงหน่วยความจำที่ 1 กระบวนการ B เขียนข้อมูลลงหน่วยความจำที่ 2 สภาวะ Deadlock จะเกิดขึ้น ถ้า A ต้องการใช้ ความจำที่ 2 และ B ต้องการใช้ความจำที่ 1 นะเวลาเดียวกัน ทำให้ A รอ B และ B รอ A
6) Deadlock จากการใช้งาน Disk ร่วมกัน A ทำการอ่านข้อมูลใน disk และ Bพยายามเขียนข้อมูลใน disk ที่ A กำลังใช้งานมีและล็อกอยู่ อาจทำให้เกิด อาการ deadlockได้

เงื่อนไขที่ทำให้เกิด Deadlock :

- Mutual exclusion เหตุการณ์ที่ process หนึ่ง สามารถถือครองทรัพยากรตัวหนึ่งได้ ในขณะใดขณะหนึ่ง ซึ่งในขณะนั้น process ตัวอื่นจะไม่สามารถใช้ทรัยากรตัวนั้นได้
- Hold and wait คือ เหตุการณ์ที่ process หนึ่ง ถือครองทรัพยากรไว้แล้ว และทำการร้องขอทรัพยากรอีกตัวที่กำลังถูกใช้อยู่ ส่งผลให้ process นั้น เกิดการรอ
- No Preemptive คือ เหตุการณ์ที่ทรัพยากรจะถูกปลดปล่อยไปให้ process อื่นใช้งานต่อ เมื่อ process เดิมที่ถือครองใช้งานเสร็จแล้ว
- Circular wait คือ เหตุการณ์ที่ process หลายๆตัว ต่างรอทรัพยากรจาก process อีกตัว กันเป็นวงกลม ยกตัวอย่างเช่น P1 รอ P2 / P2 รอ P3 / และ P3 รอ P1 ดังรูปตัวอย่างบนสไลด์



การจัดการกับ Deadlock :

- **Deadlock prevention :** ป้องกันไม่ให้มีเงื่อนไขที่จะทำให้เกิด deadlock
- Deadlock avoidance : ไม่ให้ process ได้ใช้ resource ที่เสี่ยงจะทำให้เกิด deadlock
- Deadlock detection : ยอมให้ระบบเกิด deadlock ได้ แต่ต้องมีวิธีแก้ไข

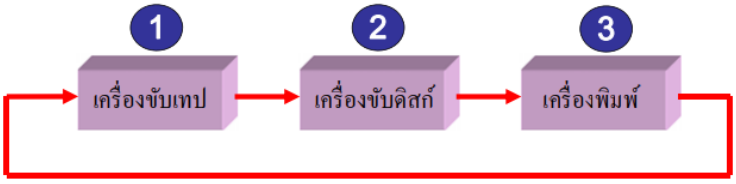
Deadlock Prevention :

- Mutual exclusion ไม่เกิดขึ้น โดยการยอมให้ process สามารถเข้าถึงทรัพยากรตัวเดียวกัน ได้หลายๆ process

- **Hold and Wait :**
    - ให้ process ได้ครอบครอง resources ที่ต้องการทั้งหมดตลอดช่วงเวลาทำงาน
        - ใช้ resource ได้ไม่เต็มประสิทธิภาพ อาจเกิด starvation
    - หรือยอมให้ process ได้ไม่เต็มประสิทธิภาพ resource เมื่อ process ไม่ได้ครอบครอง resource ใดๆ
        - สิ้นเปลืองเวลาโดยเปล่าประโยชน์
- **No Preemption :**
    - ทรัพยากรที่ต้องการว่างอยู่
        - ระบบจะจัดสรรทรัพยากรนั้นให้กับโปรเซสที่ร้องขอ
    - ทรัพยากรนั้นไม่ว่าง
        - เนื่องจากถูกถือครองโดยโปรเซสอื่น ซึ่งกำลังรอคอยทรัพยากรเพิ่มอยู่ ระบบจะ preemptive ทรัพยากรทั้งหมดของโปรเซสที่ร้องขอ
        - เนื่องจากถูกถือครองโดยโปรเซสอื่น แต่ไม่ได้กำลังรอคอยทรัพยากรอยู่ ระบบจะให้โปรเซสที่ร้องขอนั้นรอ
        - โปรเซสจะเริ่มทำงานได้อีกครั้งเมื่อได้รับทรัพยากรเก่าและอันใหม่พร้อมกัน

- **Circular Wait** : จัดอันดับให้กับ resource แต่ละชนิดให้ process เรียกใช้ resource ตามอันดับนั้นๆจากน้อยไปมาก

Ex. การร้องขอใช้เครื่องขับเทป,เครื่องขับดิกส์และเครื่องพิมพ์ตามลำดับ



ป้องกันการเกิด Deadlock :

โดยกำหนดหมายเลขให้กับทรัพยากรของระบบ

1. กำหนดให้ R = {R1, R2, … Rm] ให้ R = เซตของทรัพยากรในระบบ
2. กำหนดให้ทรัพยากรแต่ละประเภทมีเลขลำดับไม่ซ้ำกัน = F(Ri)

F(เครื่องขับเทป) = 1

F(เครื่องขับดิสก์) = 5

F(เครื่องพิมพ์) = 12

Ex **Circular Wait : ***ร้องขอทรัพยากรใดๆก็ต่อเมื่อ F(Rj) > F(Ri)

โปรเซสต้องการร้องขอให้เครื่องขับเทป (F(R) = 1)และเครื่องพิมพ์ (F(R) =12) จะเห็นได้ว่า

$$(F(R) = 1) < (F(R) = 12)$$

โปรเซสต้องขอเครื่องขับเทปก่อน จึงค่อยขอเครื่องพิมพ์

*คืนทรัพยากรกลับสู่ระบบก่อนร้องขอใหม่ F(Ri)>=F(Rj)

โปรเซสต้องการขอทรัพยากร Rj จะต้องปล่อย Rj เช่น F(R)=5 อยู่ โปรเซสต้องการ F(R)=1 ต้องคืน R5 ก่อน

$$R5 \geq R1$$

Deadlock Avoidance:

- Process แจ้งจำนวน resource ที่ต้องการ “maximum requirements (MR)”

- วิธีเลี่ยง deadlock คือ ไม่ให้ process ขอ resource ถ้าจะเกิด deadlock
- Algorithm ที่หลีกเลี่ยง deadlock คือ “Banker’s algorithm”

Banker’s algorithm

คิดโดย Edsger W.Dijkstra การทำงานจะอยู่บนพื้นฐานการทำงานของธนาคาร โดยก่อนจะกู้เงินสิ่งที่ธนาคารต้องทราบคือ

1. ต้องทราบว่ามีเงินเท่าไหร่ที่ให้ลูกค้ากู้ได้(ทราบระบบมีทรัพยากรทั้งหมดกี่ตัว)
2. ธนาคารต้องทราบจำนวนคนที่จะกู้และลูกค้าแต่ละคนกู้ได้สูงสุดเท่าไหร่(ต้องทราบงานนั้นใช้ทรัพยากรสูงสุดกี่ตัว)

Banker’s algorithm มี 2 สถานะ คือ

1. สถานะไม่ปลอดภัย (Unsafe State)
2. สภานะปลอดภัย (Safe State)

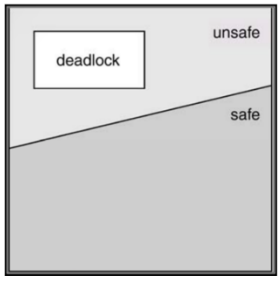
Unsafe State: •จะไม่มี process ใดทำงานสำเร็จเนื่องจากขอใช้ทรัพยากรจำนวนสูงสุด แต่ระบบไม่สามารถให้ได้ จึงเกิด deadlock

Safe State: •จัดอันดับการใช้ resource ให้ process แล้วไม่เกิด deadlock

•อันดับ<P_1 ,P_2, ...,P_n >ถือว่า safe ถ้า P_i ทำงานได้โดยใช้ resource ที่เหลืออยู่ให้ +resource ที่ถูกใช้โดย P_j ,เมื่อ j<i

-ถ้า resource ที่ P_i ต้องการ ถูกถือโดย P_j ทำให้ P_i ต้องรอ P_j **ทำงานเสร็จ**

-เมื่อ P_i ทำงานเสร็จ จะปล่อยทรัพยากรคืนสู่ระบบ P_{i+1} สามารถเรียกใช้ resource ที่ต้องการ



Ex: Safe State ให้ระบบหนึ่งมีเครื่องขับเทป 12 เครื่อง โปรเซส 3 เครื่อง คือ P_0 , P_1 , P_2 โดยแต่ละตัวต้องการใช้เครื่องขับเทปสูงสุด และได้รับเครื่องขับเทปดังตารางด้านล่าง แสดงว่า ณ เวลานั้นมีเครื่องขับเทปว่าง 3 เครื่อง

Process	Maximum Needs	Current Needs
P_0	10	5
P_1	4	2
P_2	9	2

ณ เวลา T_0 ลำดับ Process <P_1 ,P_0 ,P_2>

P_1 ไม่ต้องการเพิ่ม ทำงานเสร็จคืนให้ระบบ มีเครื่องขับเทปว่าง คือ 3+2=5 เครื่อง

P_0 ต้องการเพิ่มอีก 5 เครื่อง ทำงานเสร็จคืนให้ระบบ มีเครื่องขับเทปว่าง คือ 5+5=10 เครื่อง

P_2 ต้องการเพิ่มอีก 7 เครื่อง ทำงานเสร็จคืนให้ระบบ มีเครื่องขับเทปว่าง คือ 3+9=12 เครื่อง

Safety Algorithm

1. กำหนด vector Work และ Finish ขนาด m และ n ตามลำดับ

   Work = Available

   Finish[i] = false เมื่อ i = 1,2, 3, ..., n
2. หา Pi ที่: (a) Finish[i] = false

   (b) Need[i] <= Work

   หากไม่มี Pi ตามเงื่อนไขนี้ข้ามไปทำ step 4
3. คืน Resource กลับสู่ระบบ

   Work = Work + Allocation[i]

   Finish[i] = true

   กลับไปทำstep2
4. ถ้า Finish [i] == true สำหรับทุกๆ i,

   แสดงว่าระบบอยู่ใน safe state

ข้อเสียของ Banker's Algorithm

- จะต้องใช้กับระบบที่มีจำนวนงานที่จะประมวลผลค่อนข้างตายตัว
- ทรัพยากรต้องมีจำนวนคงที่แน่นอน
- ใช้ทรัพยากรไม่เต็มประสิทธิภาพเท่าที่ควร
- ต้องใช้อัลกอริทึมในการตรวจสอบการร้องขอเสมอ อาจเกิด Overhead
- มีปัญหาในการระบุจำนวนทรัพยากรสูงสุดที่แต่ละงานต้องการ

Deadlock Detection

- ยอมให้ระบบเข้าสู่ deadlock state
- มีวิธีในการตรวจว่าเกิด deadlock

- Directed Resource Graph

- วิธีการกู้คืนในระบบให้ออกจาก deadlock
  - Process Termination
  - Resource Preemptin

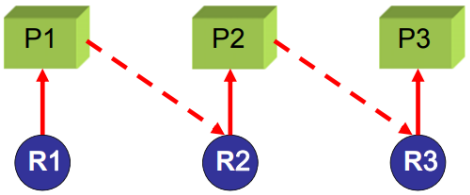
Directed Resource Graph

เป็นอัลกอริทึมสำหรับการตรวจสอบการเกิด Deadlock สามารถอธิบายได้โดยการใช้กราฟ ซึ่งหากลดรูปของกราฟได้สำเร็จ ก็จะถือว่าระบบนั้นไม่เกิด Deadlock เกิดขึ้น

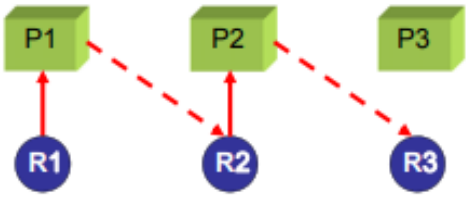
ขั้นตอนการลดรูปของกราฟ

1. หาโปรเซสซึ่งกำลังใช้งานทรัพยากรอยู่ และโปรเซสไม่ได้รอทรัพยากรอื่นอีก ให้ตัดเส้นเชื่อมโยงระหว่างโปรเซสและทรัยากรออก
2. หาโปรเซสที่คอยทรัพยากรอื่นอยู่ แต่ต้องไม่เป็นทรัพยากรที่โปรเซสอื่นครอบครอง ให้ตัดเส้นเชื่อมโยงระหว่างโปรเซสและทรัยากรที่เกี่ยวข้องกับโปรเซสนี้ออกให้หมด
3. วนกลับไปขั้นที่ 1 อีกครั้ง จนเส้นเชื่อมถูกกำจัดออกไปจนหมด จึงจะถือว่าลดรูปกราฟสำเร็จ ไม่เกิด deadlock

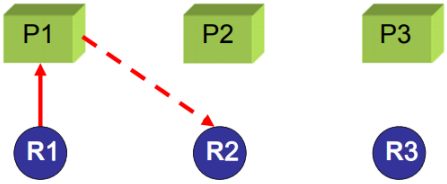
Ex:



ลดรูปขั้นตอนที่ 1



ลดรูปขั้นตอนที่ 2



วนกลับมาขั้นที่ 1 อีกครั้งหากไม่เป็นตามเงื่อนไข ให้ข้ามไปทำขั้นที่ 2

วนกลับมาขั้นตอนที่ 2 อีกครั้ง

ลดรูปสำเร็จ ไม่เกิด daedlock

การกู้คืนdeadlock มี2วิธี

- Process Termination
- Resource Preemption

Process Termination

1.จะหยุดการทำงานทุกProcesses ที่เกิด deadlocked วิธีนี้จะแก้ไขได้รวดเร็วแต่จะเสียProcessจำนวนมาก

2.จะหยุดทีละ Process จนกว่าวงจร Deadlock จะหายไป ถ้าจะยกเลิกหลาย Process ต้องตรวจหา Deadlock หลายครั้ง

Resource Preemption

จะเลือกprocessที่ต้องปล่อยresourceกลับคืนให้ระบบแล้วจะ Rollback(ถอยไปตำแหน่งที่จะแก้ deadlock) ปรับ processนั้นๆ ให้กลับสู่สภาวะก่อนจะเกิด deadlock แต่อาจทำให้เกิดStarvation(การอดตายในระบบ) บาง processอาจจะถูกเลือกให้ปล่อยresourceคืนสู่ระบบเสมอ ซึ่งอาจจะทำให้เกิด starvation ได้

เอกสารอ้างอิง

https://web.facebook.com/?locale=th_TH&_rdc=1&_rdr

https://www.ict.up.ac.th/worrakits/OS.files/PDF/%E0%B8%9A%E0%B8%97%E0%B8%97%E0%B8%B5%E0%B9%88%209%20Deadlock%202_2552.pdf