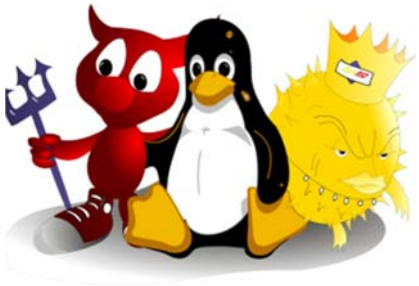
Operating System

Worrakit Sanpote



http://www.ict.pyo.nu.ac.th/nuttapons/OS.html

Part 9.

Deadlock

Deadlock

The Deadlock Problem

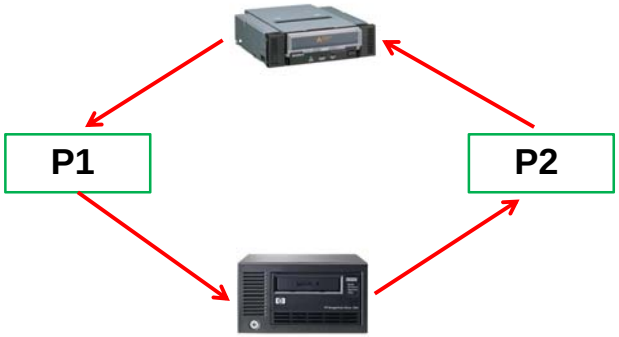
- process ร้องขอใช้ resource จากระบบ แต่ในขณะนั้น resource ที่ต้องการยังไม่ว่าง (อาจมี process อื่น ครอบครองอยู่) process นั้นจะต้องรอไปเรื่อยๆ

  ตัวอย่าง

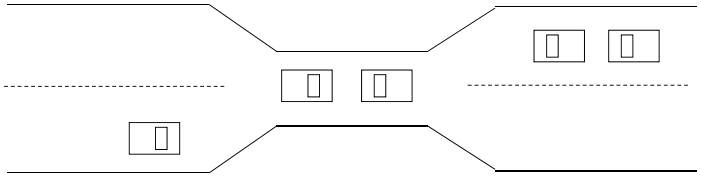
  - ในระบบที่มี tape drives สองตัว
  - P1 และ P2 ใช้ tape drive อยู่อย่างละตัวแล้วเรียกใช้อีกตัวหนึ่ง

Deadlock

The Deadlock Problem



Bridge Crossing Example



- สะพานรถวิ่งทางเดียว
- หากรถแต่ละด้านวิ่งขึ้นสะพานพร้อมกันก็จะไม่มีใครได้ขยับ
- Starvation อาจเกิดขึ้นได้ หากแต่ละด้านต่างรอให้อีกด้านหนึ่งขึ้นสะพานก่อน

ตัวอย่างกรณีของการเกิด Deadlock

1. Deadlock จากการร้องขอไฟล์
2. Deadlock ในฐานข้อมูล
3. Deadlock จากการจองใช้อุปกรณ์ที่ไม่สามารถใช้พร้อมกันได้
4. Deadlock จากการจองใช้อุปกรณ์หลายๆ ชนิด
5. Deadlock ใน Spooling
6. Deadlock จากการใช้งาน disk ร่วมกัน

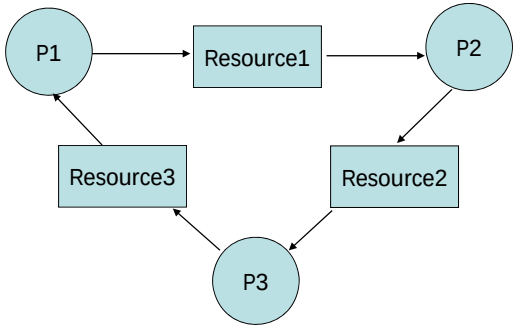
เงื่อนไขที่ทำให้เกิด Deadlock

- Mutual exclusion: มี resource ที่ใช้ร่วมกันไม่ได้อยู่อย่างน้อย 1 ตัว จะมีเพียง 1 process เท่านั้นที่ใช้ resource ตัวนั้นได้

- Hold and wait: process ครอบครอง resource อันหนึ่งอยู่แล้ว และรอเรียกใช้ resource ใหม่ซึ่งถูกครอบครองโดย process อื่นอยู่
- No preemption: ไม่มีอะไรบังคับให้ process เลิกใช้ resource ได้นอกจาก process ที่ครอบครอง resource อยู่จะเลิกใช้เอง

Deadlock

- Circular Wait: จากเงื่อนไข Mutual exclusion, Hold and wait และ No preemption หากเขียนเป็นรูปก็จะเหมือนกับลูกโซ่ที่ย้อนกลับมาที่เดิม



Deadlock

การจัดการกับ Deadlock

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

Deadlock

Deadlock Prevention

- Mutual Exclusion : ยอมให้ระบบมีการใช้ทรัพยากรร่วมกันได้ เช่น แฟ้มข้อมูลประเภท read only เป็นต้น

Deadlock

- Hold and Wait :
  - ให้ process ได้ครอบครอง resources ที่ต้องการทั้งหมดตลอดช่วงเวลาการทำงาน
    - ใช้ resource ได้ไม่เต็มประสิทธิภาพ อาจเกิด starvation
  - หรือยอมให้ process เรียกใช้ resource เมื่อ process ไม่ได้ครอบครอง resource ใดๆ
    - สิ้นเปลืองเวลาโดยเปล่าประโยชน์

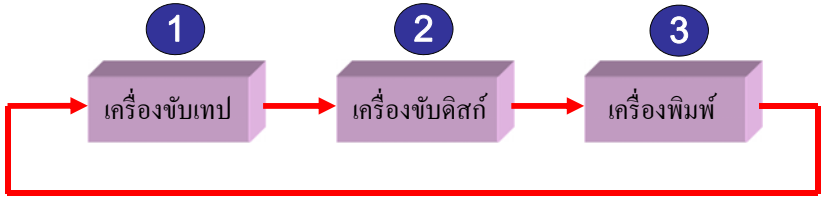
Deadlock

- No Preemption :
  - ทรัพยากรที่ต้องการว่างอยู่
    - ระบบจะจัดสรรทรัพยากรนั้นให้กับโปรเซสที่ร้องขอ
  - ทรัพยากรนั้นไม่ว่าง
    - เนื่องจากถูกถือครองโดยโปรเซสอื่น ซึ่งกำลังรอคอยทรัพยากรเพิ่มอยู่ ระบบจะ preemptive ทรัพยากรทั้งหมดของโปรเซสนั้น และจัดสรรทรัพยากรให้กับโปรเซสที่ร้องขอ
    - เนื่องจากถูกถือครองโดยโปรเซสอื่น แต่ไม่ได้กำลังรอคอยทรัพยากรอยู่ ระบบจะให้โปรเซสที่ร้องขอนั้นรอ
    - โปรเซสจะเริ่มทำงานได้อีกครั้งเมื่อได้รับทรัพยากรเก่าและอันใหม่พร้อมกัน

Deadlock

- Circular Wait : จัดอันดับให้กับ resource แต่ละชนิด และให้ process เรียกใช้ resource ตามอันดับนั้นๆจากน้อยไปมาก

Ex การร้องขอใช้เครื่องขับเทป, เครื่องขับดิกส์และเครื่องพิมพ์ตามลำดับ



Deadlock

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

Deadlock

- Ex Circular Wait :
  - ร้องขอทรัพยากรใดๆ ได้ก็ต่อเมื่อ F(Rj) > F(Ri)
    - โปรเซสต้องการร้องขอใช้เครื่องขับเทป (F(R) = 1) และเครื่องพิมพ์ (F(R) = 12) จะเห็นได้ว่า

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

โปรเซสนั้นจะต้องร้องขอเครื่องขับเทปก่อน แล้วจึงค่อยร้องขอเครื่องพิมพ์

- คืนทรัพยากรกลับสู่ระบบก่อนร้องขอใหม่ F(Ri) ≥ F(Rj)
  - โปรเซสต้องการร้องขอทรัพยากร Rj จะต้องปล่อยทรัพยากร Ri เช่น ถือ F(R) = 5 อยู่ โปรเซสต้องการ F(R) = 1 ต้องคืน R5 ก่อน

R5 ≥ R1

Deadlock Avoidance

- แต่ละ process แจ้งจำนวน resource ทั้งหมดที่ต้องการใช้ เรียกว่า "**maximum requirements (MR)**"
- วิธีหลีกเลี่ยง deadlock
  - ไม่ให้ process ขอ resource ถ้าจะทำให้เกิด deadlock
- Algorithm เพื่อหลีกเลี่ยง deadlock
  - Banker's algorithm

Banker's algorithm

คิดค้นโดย Edsger W. Dijkstra นักวิทยาศาสตร์คอมพิวเตอร์ชาวเนเธอแลนด์ การทำงานของ Banker's algorithm จะอยู่บนพื้นฐานของการทำงานของธนาคาร ซึ่งเปิดให้ลูกค้ามากู้เงิน โดยก่อนที่จะกู้เงินได้นั้น สิ่งที่ธนาคารจำเป็นต้องทราบ คือ

1. ธนาคารต้องทราบว่ามีเงินอยู่ทั้งหมดเท่าไหร่ที่จะสามารถให้ลูกค้ากู้เงินได้ (ต้องทราบว่าระบบมีทรัพยากรทั้งหมดกี่ตัว)
2. ธนาคารต้องทราบว่าลูกค้าที่ต้องการกู้เงินมีทั้งหมดกี่คนและลูกค้าแต่ละคนสามารถกู้เงินได้สูงสุดเท่าไหร่ (ต้องทราบว่างานนั้นใช้ทรัพยากรสูงสุดกี่ตัว)

Banker's algorithm จะแบ่งสถานะออกเป็น 2 สถานะ คือ

- สถานะไม่ปลอดภัย (Unsafe State)
- สถานะปลอดภัย (Safe State)

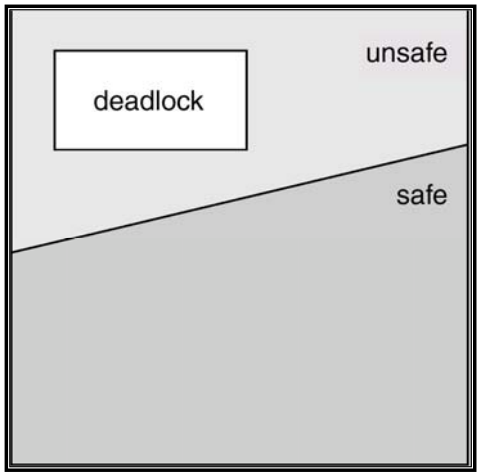
Unsafe State

- ระบบอยู่ใน Unsafe state จะไม่มี process ใดทำงานได้เสร็จ เนื่องจาก process ต่างๆ ร้องขอใช้ทรัพยากรตามจำนวนสูงสุดที่สามารถจะใช้ได้ แต่ระบบไม่สามารถจัดสรรทรัพยากรให้กับงานต่างๆ ได้ จึงเกิด deadlock ขึ้น

Safe State

- ระบบอยู่ใน safe state เมื่อจัดอันดับการใช้ resource ให้กับ process แล้วไม่เกิด deadlock
- อันดับ $<P_1, P_2, \ldots, P_n>$ ถือว่า safe ถ้า P_i สามารถทำงานได้โดยใช้ resources ที่เหลืออยู่ให้ + resource ที่ถูกใช้โดย P_j , เมื่อ $j<i$
  - ถ้า resource ที่ P_i ต้องการ ถูกถือครองโดย P_j ทำให้ P_i ต้องรอจนกระทั่ง P_j ทำงานเสร็จ
  - เมื่อ P_i ทำงานเสร็จ จะปล่อยทรัพยากรที่ใช้เสร็จคืนสู่ระบบ P_{i+1} สามารถเรียกใช้ resource ที่ต้องการ

Safe, Unsafe , Deadlock State



Ex. Safe State

ให้ระบบหนึ่งมีเครื่องขับเทป 12 เครื่อง มีโปรเซสอยู่ 3 โปรเซส คือ P_0 , P_1 , P_2 โดยที่แต่ละโปรเซสมีความต้องการใช้เครื่องขับเทปสูงสุด และได้รับเครื่องขับเทปดังตารางด้านล่าง (แสดงว่า ณ เวลานั้นมีเครื่องขับเทปว่างอยู่ 3 เครื่อง)

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

Deadlock

ณ เวลา T_0 ลำดับโปรเซส $<P_1, P_0, P_2>$ ⟹ Safe State

P_1 → ไม่ต้องการเพิ่ม ทำงานเสร็จคืนทรัพยากรให้กับระบบ

มีเครื่องขับเทปว่าง คือ 3 + 2 = 5 เครื่อง

P_0 → ต้องการเพิ่มเครื่องขับเทปอีก 5 เครื่อง ทำงานเสร็จคืนทรัพยากรให้กับระบบ

มีเครื่องขับเทปว่าง คือ 5 + 5 = 10 เครื่อง

P_2 → ต้องการเพิ่มเครื่องขับเทปอีก 7 เครื่อง ทำงานเสร็จคืนทรัพยากรให้กับระบบ

มีเครื่องขับเทปว่าง คือ 3 + 9 = 12 เครื่อง

Deadlock

Data Structures สำหรับ Banker's Algorithm

- ให้ n เป็นจำนวน process และ m เป็นจำนวนชนิดของ resource
- Available: Vector ขนาด m ถ้า Available [j] = k, แสดงว่ามี resource ชนิด R_j ว่างอยู่ k ตัว
- Max: matrix ขนาด *n x m* ถ้า Max [*i,j*] = *k*, แสดงว่า P_i อาจเรียกใช้ resource ชนิด R_j มากที่สุดเป็นจำนวน k ตัว

Deadlock

- Allocation: matrix ขนาด *n* x *m* ถ้า Allocation[*i,j*] = *k* แสดงว่า P_i กำลังใช้ resource R_j จำนวน k ตัว
- Need: matrix ขนาด *n* x *m* ถ้า Need[*i,j*] = *k*, แสดงว่า P_i ต้องการ resource R_j เพิ่มขึ้นอีก k ตัวเพื่อที่จะทำงานให้เสร็จ
- Need [i,j] = Max[i,j] – Allocation [i,j]

Deadlock

Resource-Request Algorithm สำหรับ Process P_i

1. ถ้า $Request_i > Need_i$ ระบบจะแสดง error เพราะโปรเซสเรียกทรัพยากรมากกว่า MR แต่ถ้า $Request_i \leq Need_i$ ไปที่ข้อ 2.
2. ถ้า $Request_i > Available$ แล้ว P_i ต้องรอจนกว่าทรัพยากรที่ร้องขอจะว่าง แต่ถ้า $Request_i \leq Available$ ไปที่ข้อ 3.

3. พยายามจัดทรัพยากรให้ Pi โดยแก้ไข state ดังนี้:

$$Available = Available - Request_i$$
$$Allocation_i = Allocation_i + Request_i$$
$$Need_i = Need_i - Request_i$$

- ถ้า safe ⇒ Pi ได้ใช้ resources
- ถ้า unsafe ⇒ Pi ต้องรอและปรับ state กลับไปที่เดิม

Safety Algorithm

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

   Work = Available
   Finish[i] = false สำหรับ i = 1,2, 3, …, n

2. หา Pi ที่:

   (a) Finish[i] = false
   (b) Need[i] ≤ Work
   หากไม่มี Pi ตามเงื่อนไขนี้ข้ามไปทำ step 4

Safety Algorithm (ต่อ)

3. คืน Resource กลับสู่ระบบ

   Work = Work + Allocation[i]
   Finish[i] = true
   กลับไปทำ step 2

4. ถ้า Finish [i] == true สำหรับทุกๆ i,
   แสดงว่าระบบอยู่ใน safe state

ตัวอย่าง Banker's Algorithm

5 processes P0 ถึง P4; resource 3 ชนิด
A (10 ตัว), B (5 ตัว), และ C (7 ตัว)

	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

ตัวอย่าง Banker's Algorithm (ต่อ)

	Allocation	Max	Need	Available
	A B C	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3	3 3 2
P1	2 0 0	3 2 2	1 2 2	
P2	3 0 2	9 0 2	6 0 0	
P3	2 1 1	2 2 2	0 1 1	
P4	0 0 2	4 3 3	4 3 1	

- ระบบอยู่ใน safe state ถ้าเราจัดอันดับการใช้งาน resource ตามนี้

< P1, P3, P4, P0, P2 >

Ex. P1 Request (1,0,2)

- ตรวจสอบว่า Request ≤ Available

นั่นคือ (1,0,2) ≤ (3,3,2) ⇒ true

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

P1 Request (1,0,2) (ต่อ)

- ตรวจสอบด้วย safety algorithm จะได้ว่าอันดับ ?

ข้อเสียของ Banker's algorithm

- จะต้องใช้กับระบบทีมีจำนวนงานที่จะประมวลผลค่อนข้างตายตัว
- ทรัพยากรของระบบต้องมีจำนวนคงที่แน่นอน
- ใช้ทรัพยากรไม่เต็มประสิทธิภาพเท่าที่ควร
- ระบบต้องใช้อัลกอริทึมในการตรวจสอบการร้องขออยู่เสมอ ส่งผลให้เกิด Overhead
- มีปัญหาในการระบุจำนวนทรัพยากรสูงสุดที่แต่ละงานนั้นต้องการ

Deadlock Detection

- ยอมให้ระบบเข้าสู่ deadlock state
- มีวิธีในการตรวจว่าเกิด deadlock
  - Directed Resource Graph
- วิธีการกู้คืนระบบ ให้ออกจาก deadlock
  - Process Termination
  - Resource Preemption

Directed Resource Graph

เป็นอัลกอริทึมสำหรับการตรวจสอบการเกิด Deadlock สามารถอธิบายได้โดยการใช้กราฟ จะพิจารณาจากกราฟและลดรูปของกราฟนั้น ซึ่งหากสามารถลดรูปของกราฟได้สำเร็จ ก็จะถือว่าระบบนั้นไม่มี Deadlock เกิดขึ้น

ขั้นตอนการลดรูปของกราฟ

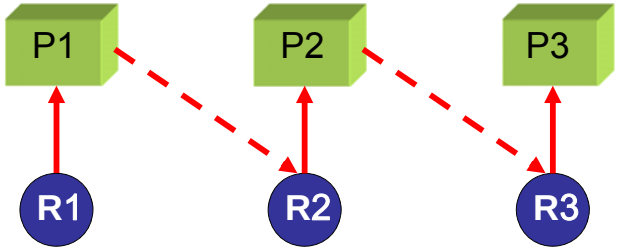
ขั้นตอนการลดรูปของกราฟแบ่งออกเป็น 3 ขั้นตอน คือ

1. หาโปรเซสซึ่งกำลังใช้งานทรัพยากรอยู่ และโปรเซสนั้นไม่ได้รอคอยทรัพยากรอื่นๆ อีก ให้ตัดเส้นเชื่อมโยงระหว่างโปรเซสและทรัพยากรนั้นออก

2. หาโปรเซสที่รอคอยทรัพยากรอื่นอยู่ โดยที่ทรัพยากรที่โปรเซสนี้กำลังรอคอย ต้องเป็นทรัพยากรที่ไม่ถูกโปรเซสอื่นครอบครองอยู่ ให้ตัดเส้นเชื่อมโยงระหว่างโปรเซสและทรัพยากรที่เกี่ยวข้องกับโปรเซสนี้ออกไปให้หมด
3. วนกลับไปทำขั้นตอนที่ 1 ใหม่อีกครั้ง จนกระทั่งเส้นที่เชื่อมโยงระหว่างทรัพยากรและโปรเซสถูกกำจัดออกไปจนหมด จึงจะถือว่าลดรูปกราฟได้สำเร็จ ซึ่งหมายความว่า "ระบบไม่มีการเกิด Deadlock"

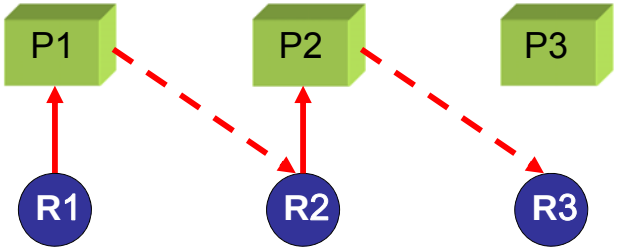
Deadlock

Ex จากรูปด้านล่างจงพิจารณาว่าระบบเกิด **Deadlock** หรือไม่



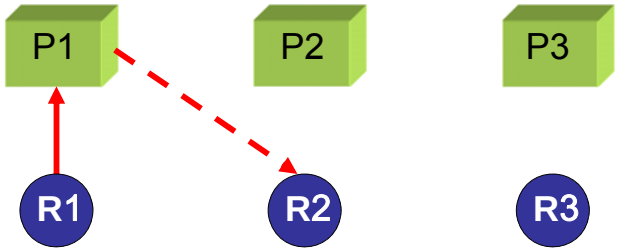
Deadlock

ลดรูปขั้นตอนที่ 1



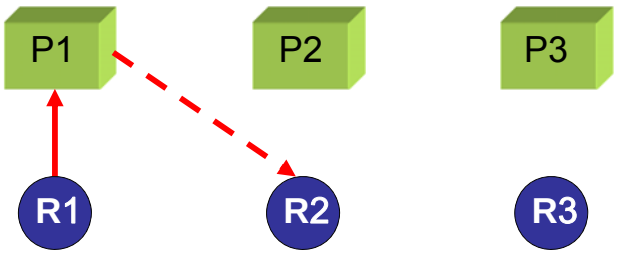
Deadlock

ลดรูปขั้นตอนที่ 2



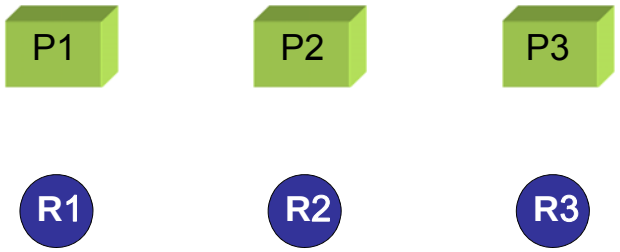
Deadlock

วนกลับมาลดรูปที่ขั้นตอนที่ 1 อีกครั้ง หากไม่เป็นไปตามเงื่อนไขการลดรูปกราฟให้ข้ามไปทำขั้นตอนที่ 2



Deadlock

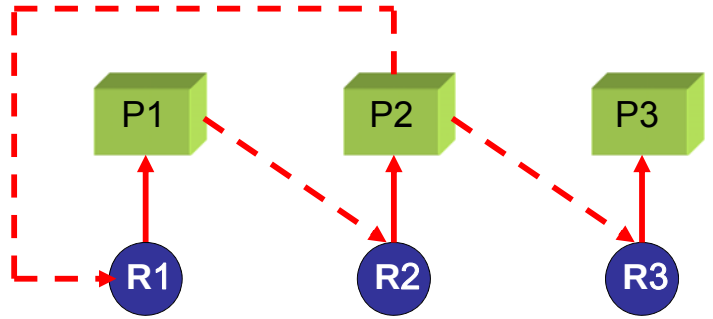
วนกลับมาลดรูปที่ขั้นตอนที่ 2 อีกครั้ง



สามารถลดรูปกราฟได้สำเร็จ ดังนั้นระบบนี้ไม่เกิด **Deadlock**

Deadlock

Ex จากรูปด้านล่างจงพิจารณาว่าระบบเกิด **Deadlock** หรือไม่



Deadlock

Deadlock

การกู้คืนจาก **Deadlock**

- Process Termination
- Resource Preemption

Process Termination

- หยุดการทำงานของทุก processes ที่เกิด deadlocked
  - รวดเร็ว
  - จำนวน process ที่ต้องเสียงานไปมีมาก
- หยุดการทำงานทีละ process จนกระทั่ง deadlock หมดไป
  - ต้องทำงานเพื่อตรวจสอบ deadlock มากขึ้น

Resource Preemption

- เลือก process ที่จะต้องปล่อย resource กลับคืนให้กับระบบ
- Rollback - ปรับ process นั้นๆ ให้กลับคืนสู่สภาวะก่อนที่จะเกิด deadlock
- Starvation - บาง process อาจจะถูกเลือกให้ปล่อย resource คืนสู่ระบบเสมอ ซึ่งอาจเกิด starvation ได้

Exercise 1.

จงหาลำดับของโปรเซสในการร้องขอทรัพยากรของระบบโดยไม่เกิด Deadlock

(กำหนดให้ทรัพยากรในระบบทั้งหมดมีชนิด A = 7, B = 2 และ C = 6)

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

Exercise 2.

ถ้าให้ P2 ขอใช้ทรัพยากรในระบบชนิด C เพิ่มขึ้นอีก 1 ตัว ?

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

Question