สรุป Os คะแนนเต็ม 50 เก็บ 35 คะแนนดิบ

อธิบาย 7 ข้อ คะแนนรวมทั้งหมด 16 คะแนน

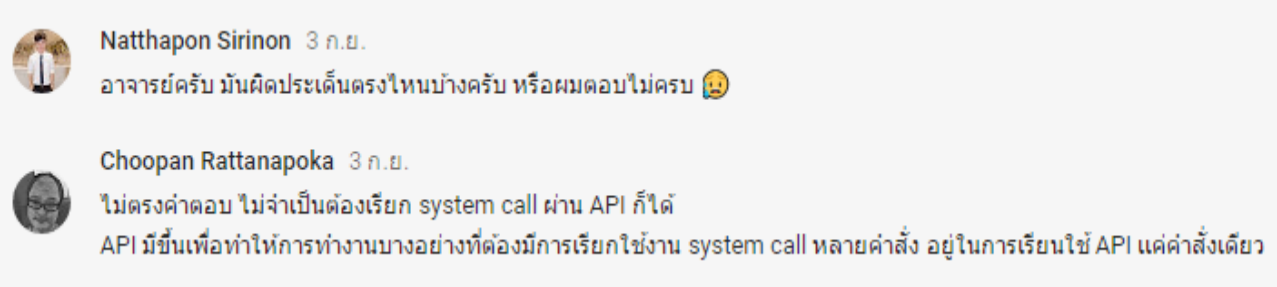
*****ออกแน่นอน

***อาจจะออก 80 %

Os คือ ระบบปฏิบัติการที่ทำหน้าที่เป็นตัวกลางเชื่อมต่อระหว่างผู้ใช้กับคอมพิวเตอร์และฮาร์ดแวร์คอมพิวเตอร์

System call เป็นส่วนติดต่อสำหรับการเขียนโปรแกรมเพื่อติดต่อไปยังบริการที่ Os ให้บริการ

***API ข้อนี้ เทพมาเอง



*****การส่ง Parameters ให้กับ System call มี 3 วิธี

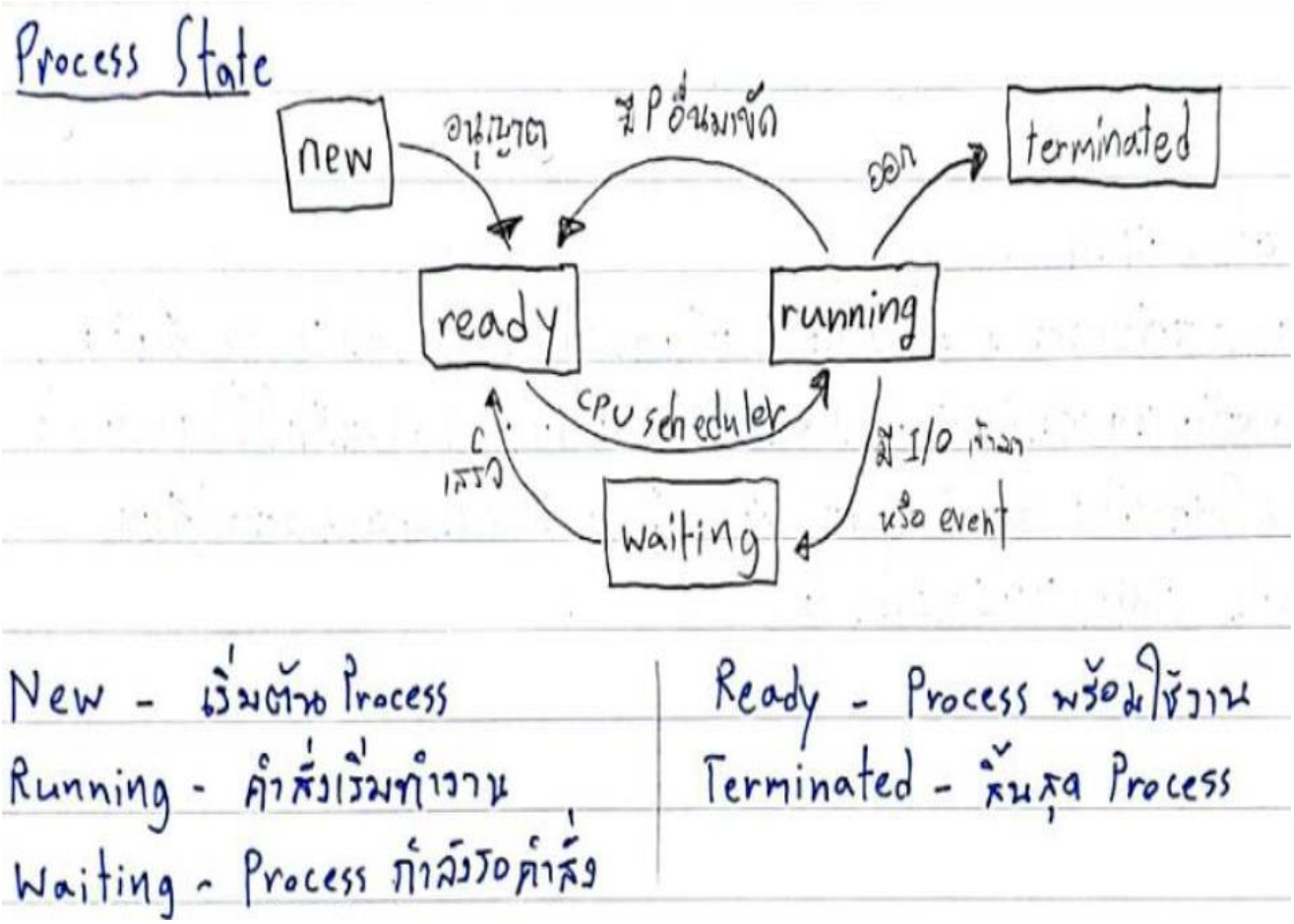
1. ส่ง Address ของ Parameters ไปเก็บไว้ใน Register แล้ว System call ไปอ่านมาใช้งาน มีข้อเสียคือถ้า Parameters เยอะ Register จะไม่พอ
2. จอง Memory ไว้ก่อน แล้วเก็บ Address ของ Parameters ไปไว้ใน Memory ที่จอง แล้วส่ง Address ของ Parameters ที่อยู่ใน Memory ไปยัง Register
3. เก็บในรูปแบบของ Stack แล้วให้ System call ไป pop มาใช้ โดย Address ของ Stack ส่งไปยัง Register

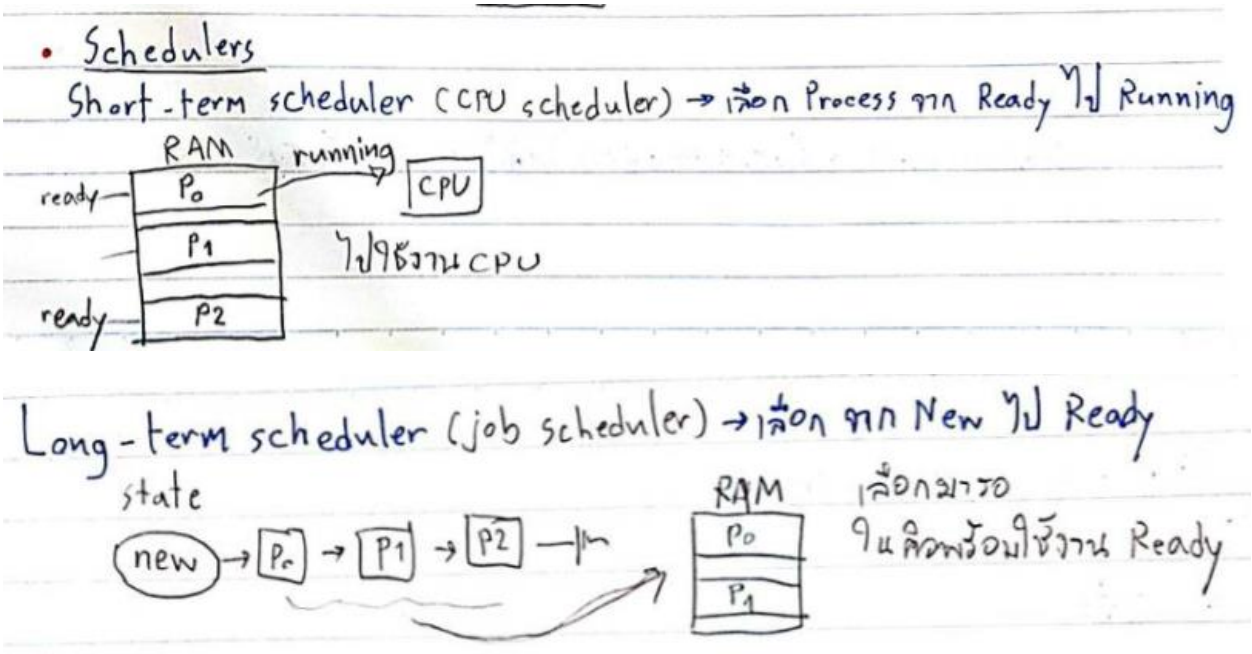
kernel - ส่วนบริหารระบบ ติดต่อระหว่าง Hardware กับ Software

***Monolitic Kernel จะมีรูปแบบเป็นก้อนเดียว กล่าวคือ "**ไม่มีการแบ่งส่วนการทำงาน**" แตกต่างจาก Microkernel อย่างชัดเจนคือ Microkernel จะ"**มีการแบ่งส่วนการทำงาน**" โดยย้ายการทำงานจาก kernel ไปอยู่ที่ user space จึงเป็นผลให้ทั้ง 2 kernel มีข้อดีและข้อเสียในตัว Monolitic Kernel นั้น จะมีการทำงานที่รวดเร็ว แต่จะยากต่อการย้าย port Os ไปยัง architectures ใหม่ๆ ส่วน Microkernel จะง่ายต่อการย้าย port Os ไปยัง architectures ใหม่ๆ และมีความเสถียรปลอดภัยมากขึ้น แต่ก็ต้องแลกกับการทำงานที่ช้าลง เนื่องจากมีการทำงานผ่าน System call บ่อยครั้ง

*****Process คือ โปรแกรมที่อยู่ระหว่างการทำงาน และจะทำงานเป็นลำดับ

Process State



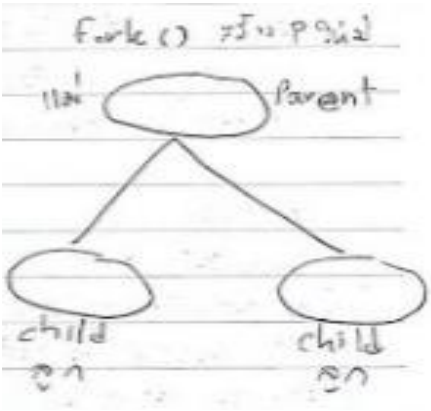


***CPU scheduler และ Job scheduler ต่างกันคือ

CPU seheduler คือการเลือกงานจากสถานะ ready ที่พร้อมประมวลผล เพื่อเข้าใช้ cpu ในการประมวลผล Job schedular คือการเลือกงานขึ้นมาไว้ใน ram เพื่อรอการประมวลผลต่อไป ข้อแตกต่างชัดเจนเลยก็คือลักษณะการทำงานที่กล่าวไว้ข้างต้น และความถี่การทำงานยังแตกต่างกันอีก โดย CPU seheduler จะทำงานถี่มาก เพราะต้องทำงานได้เร็วมากในการประมวลผล ส่วน Job schedular ไม่ต้องทำงานถี่มาก สามารถทำงานช้าได้บ้าง เพราะเลือกเข้าไปแล้ว ก็ต้องรออยู่ดี

***Context Switch คือการที่ CPU สับเปลี่ยน process อื่นขึ้นมาทำงาน โดยระบบจะเก็บสถานะของ process ที่ทำงานอยู่ไว้ แล้วไปโหลดสถานะของ process ใหม่ ขึ้นมาทำงาน ซึ่งเวลาที่ใช้ในการสับเปลี่ยนนี้ ระบบจะไม่สามารถทำงานอย่างอื่นได้ ก็จะสูญเสียเวลาตรงนี้ไป ช้าเร็วก็จะขึ้นอยู่กับ hardware ที่ support ว่าสามารถรองรับการ save และ load ได้ในขั้นตอนเดียวไหม ก็จะช่วยให้รวดเร็วในการสับเปลี่ยน และข้อดีของการ context switch นี้จะช่วยแก้ปัญหาคอขวดของระบบได้

foke() เป็น System call สำหรับสร้าง process ใหม่



abort() คำสั่งหยุดการทำงาน process ลูก โดย process แม่ สามารถใช้คำสั่งนี้ได้ เมื่อ

- process ลูก ใช้ทรัพยากรมากกว่าที่มีให้
- งานที่ให้ process ลูกทำ ไม่มีความจำเป็นต่อไป

***process จะถูกเรียกว่า zombie เมื่อ process แม่ ไม่ได้ใช้คำสั่ง wait() จึงเป็นผลให้ process ลูกไม่สามารถคืนสถานะการทำงานให้กับ process แม่ ได้ “ลูกทำงานเสร็จแต่ตายไม่ได้” จึงถูกเรียกว่า zombie

***process จะถูกเรียกว่า orphan เมื่อ process แม่ ได้ทำงานเสร็จก่อน process ลูก โดยไม่ได้รอให้ process ลูก ทำงานเสร็จก่อน จึงถูกเรียกว่า orphan

ความแตกต่างของ Ordinary pipe และ Named pipe

Ordinary pipe เป็นการติดต่อสื่อสารแบบทิศทางเดียว โดยใช้ติดต่อกันเฉพาะ process ที่มีความสัมพันธ์กัน เช่น process แม่ จะสร้าง pipe เพื่อติดต่อกับ process ลูก แต่ Named pipe เป็นการติดต่อสื่อสารแบบสองทิศทาง โดย process ต่างๆ สามารถติดต่อสื่อสารกันได้ โดยไม่ต้องมีความสัมพันธ์กัน

Thread คือ หน่วยการทำงานย่อยที่อยู่ใน Process สามารถเรียกใช้ thread หลายๆ ตัวได้พร้อมๆ กัน โดย thread แต่ละตัว ของ process เดียวกันจะทำงานแตกต่างกันแต่มีความเกี่ยวข้องกันและต้องทำงานอยู่ใต้สภาพแวดล้อมเดียวกัน

Concurrency คือระบบการทำงานของ CPU ที่มี 1 Core แต่ใช้หลักการแบ่งกันทำงาน โดยสลับการทำงานอย่างรวดเร็ว เหมือนเป็นการทำงานพร้อมกัน
Parallelism คือระบบการทำงานของ CPU ที่มีหลาย Core โดยจะสามารถทำงานได้พร้อมกัน

***Data parallelism และ Task parallelism มีความแตกต่างกัน คือ Data parallelism คือการทำงานโปรแกรมแบบขนาน กล่าวคือ การทำงานจากโค้ดคำสั่งเดียวกัน แต่ข้อมูลที่นำมาประมวลผล คนละชุดกัน จึงทำให้รวดเร็วมากขึ้นในการประมวลผล เช่น Big Data เป็นต้น แต่ Task parallelism คือ การกระจายงานออกไป และงานที่กระจายนี้ก็ทำคนละแบบไปเลย เช่น การพยากรณ์อากาศของวันถัดไป ก็จะแยกการคำนวณของแต่ละภาคไปในแต่ละเครื่องคอมพิวเตอร์นั้นๆ เป็นต้น

***Race Condition คือสภาวะที่ procrss 2 process ที่ต้องการใชตัวแปรเดียวกัน ในเวลาเดียวกัน จึงทำให้เกิดการแย่งตัวแปรกัน และทำให้ผลลัพธ์อาจจะเกิดการผิดพลาดขึ้น
Critical Section การทำงานของ code ที่สามารถทำงานได้แค่ 1 process ห้ามการทำงาน > 1 process จะสามารถป้องกันการเกิด Race Condition ได้
Preemptive **สามารถ**สับเปลี่ยนการทำงานของ process ได้ ในขณะที่ทำงานอยู่ใน kernel
Non-preemptive ห้ามสับเปลี่ยนการทำงานของ process ได้ ในขณะที่ทำงานอยู่ใน kernel

Deadlock คือการที่ process ต้องการใช้ทรัพยากร แต่ทรัพยากรนั้นไม่ว่าง เนื่องจาก process อื่นกำลังใช้งานอยู่ ทำให้ process นั้นๆ ต้องรอไปจนไม่มีสิ้นสุด

คำนวณข้อ 8 9 แบ่งเป็นข้อละ 3 ข้อย่อย รวมทั้งหมด 6 ข้อย่อย คะแนนรวมทั้งหมด 12 คะแนน

ข้อ 8-9 คำนวณ Amdahl's Law 2 ข้อ เดาว่าให้หาจำนวนเท่าข้อนึง, เวลาข้อนึง โจทย์บอก parallel เป็น %

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

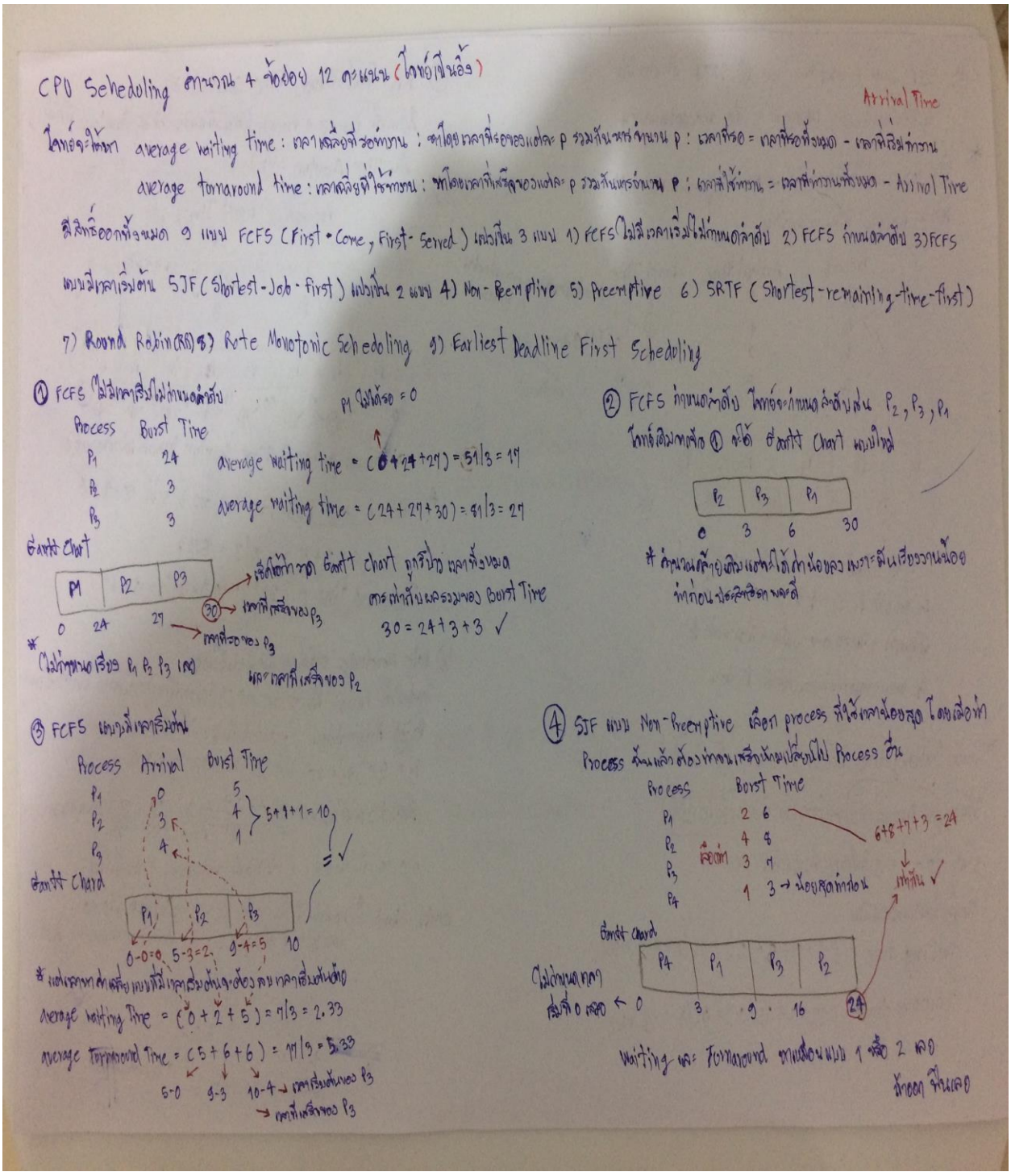
S = serial Ex parallel 60%. = serial 40%.

N = จำนวน core

เวลา x speedup = เวลาที่ใช้ (เพิ่มขึ้น) speedup - เร็วขึ้นกี่เท่า

เวลา / speedup = เวลาที่ใช้ (ลดลง)

คำนวณข้อ 10 แบ่งเป็นข้อละ 4 ข้อย่อย คะแนนรวมทั้งหมด 12 คะแนน



⑤ SJF แบบ preemptive และ ⑥ SRTF ตัวเดียวกัน

ทำวิธีเดียวกันเลย แต่อาจมี 2 ชื่อ น่าจะออกแน่นอน

* เลือก Process ที่น้อยที่สุดมาทำ แต่ขณะทำอยู่สามารถหยุด !!

แล้วนำ Process ที่สั้นกว่ามาทำได้

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

น้อยสุดก็จริง แต่เวลาเริ่มต้น P_1 มาก่อน

$8+4+9+5=26$ เท่ากัน ✓

P_1	P_2	P_4	P_1	P_3

0 ① 5 ⑩ 17 26

ทำ P_1 ได้ 1 วิแล้ว P_2 เริ่มทำงานได้ และเวลาในการทำ P_2 น้อยกว่า P_1 $P_2 = 4$ วิ $P_1 = 8-1<7$ วิ เพราะทำไปแล้ว 1 วิตรง 0→1 เหลือ 7 วิ มากกว่า P_2 P_1 เลยถูกหยุดการทำงาน แล้วทำ P_2 ก่อน

* การหา waiting time Process ที่ทำงาน 2 ครั้ง เช่น P_1 ให้เอาเวลาที่รอครั้งที่ 2 ลบด้วยเวลาที่ถูกหยุดการทำงาน จะได้ $10-1=9$ วิ คือเวลาทั้งหมดที่ P_1 รอ ที่เหลือคำนวณเหมือนเดิม

waiting time $= (9+0+15+2) = 26/4 = 6.5$ msec

Turnaround time $= (17+4+24+7) = 52/4 = 13$ msec

⑦ RR ไม่ต้องกำหนด Time Quantum มา

คือการที่ Process 1 Process ทำงานได้ไม่เกิน Time Quantum ที่ให้มา

เช่น Time Quantum = ④

Process	Burst Time
P_1	24
P_2	3
P_3	3

$24+3+3=30$ = ✓

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 ③ 7 ③ 10 ④ 14 ④ 18 ④ 22 ④ 26 ④ ㉚

ทำได้แค่ 4 วิ ตามที่ Time Quantum กำหนดไว้ แล้วต้องหยุด

* waiting time คิดคล้ายข้อ 6,7 ของ P_1 จะได้ $10-4=6$

$= (6+4+7) = 17/3 = 5.67$

Turnaround Time $= (30+7+10) = 47/3 = 15.67$

⑧ Rate Monotonic อาจไม่ออกเพื่อได้ตารอสอบ

โจทย์จะให้ Period สั้น t มา โดย t คือเวลาที่ใช้ทำงานคล้าย Burst time

Period สั้นกว่าทำก่อน, ยาวกว่าทำทีหลัง เช่นมี 2 Process

$P_1 = 50, t_1 = 20$ และ $P_2 = 100, t_2 = 35$

เช็คว่าทำงานได้ไหมโดยหาจาก $\left(\frac{t_1}{P_1}+\frac{t_2}{P_2}\right) = \left(\frac{20}{50}+\frac{35}{100}\right) = 0.4+0.35$

$= 0.75$ ถ้าน้อยกว่า 1 ทำงานได้ แต่มากกว่า 1 ทำงานไม่ได้

Gantt Chart * จากตัวที่ P สั้นกว่าคือ P_1 และทำงานอยู่ t วิคือ 20 และทุกๆ P วิคือทุกๆ 50 P_1 จะต้องกลับมาทำงานอีก ดังนี้

P_1	P_2	P_1	P_2

0 20 50 70 ~5~ 75

t_1 $t_2 = 35$ แต่ทำงานได้ 30 ขาดอีก 5 → ทำต่อจนเสร็จ

⑨ Earliest Deadline First

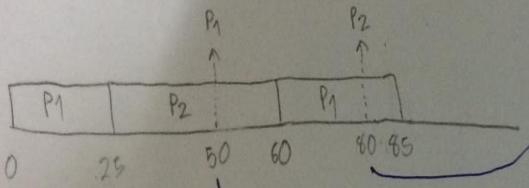
คล้อย ⑧ แต่พิจารณา Deadline

Deadline ที่คือ t_1 เลยที่ต้องเริ่มทำงานขึ้นเอง

ถ้าหากมีอันอื่นมาให้ทำงานเสร็จแล้ว สามารถทำต่อขนเสร็จได้

เช่น $P_1 = 50, t_1 = 25$ และ $P_1 = 80, t_1 = 35$

$\left(\frac{25}{50} + \frac{35}{80}\right) = 0.5 + 0.43 = 0.93$ ทำงานได้



ตรงๆ P_2 ต้องกลับมาทำงานอยู่ที่ 80 แต่ P_1 ที่ทำเสร็จแล้วตั้งแต่ 5 เลยทำให้เสร็จ

ตรงๆ P_1 ต้องกลับมาทำงาน แต่ P_2 Deadline ใกล้เสร็จแล้ว จึงสามารถทำต่อไปอีก 10 เพื่อเสร็จการทำงาน

ออก 4 ข้อ

แต่ว่า 1 FCFS แบบกำหนดเวลาเริ่มต้นข้อ ③

2 SJF แบบ Non-Preemptive ข้อ ④ หรือ EDF ข้อ ⑨

3 SRTF หรือ SJF แบบ Preemptive ข้อ ⑤, ⑥ ออกอันไหนก็ทำเหมือนกัน ต่างแต่ชื่อ

4 RR ข้อ ⑦

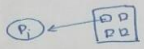
คำนวณข้อ 11 แบ่งเป็นข้อละ 4 ข้อย่อย คะแนนรวมทั้งหมด 10 คะแนน

Dead lock

เกิดเมื่อ !

1. Mutual exclusion : 1 process เท่านั้นที่สามารถใช้ทรัพยากรได้
2. Hold and wait : process หนึ่งใช้ทรัพยากรอยู่ อีก process มาร้องขอ
3. No preemption : process ใช้ทรัพยากรไม่ยอมปล่อย
4. Circula wait : การรอทรัพยากรของ process ทั้งไปเรื่อยๆ

Resource - Allocation Graph

- Request edge → ร้องขอ 
- assignment edge → ถือครองอยู่ 

คำนวณ Banker's Algorithm

สิ่งที่ต้องรู้*

- Available = บอกว่าทรัพยากร R_n มีทรัพยากรว่างอยู่กี่ Instance
- Max = Process แต่ละตัวต้องใช้ทรัพยากรสูงสุดเท่าไหร่
- Allocation = Process ได้รับการจัดสรรมาแล้วกี่ตัว (ถือครองทรัพยากรเท่าไหร่แล้ว)
- Need = ต้องการทรัพยากร

*ลองทำโจทย์ !

จากโจทย์ มี 5 process : P_0 ถึง P_4

มี 3 ประเภททรัพยากร : A (10 instances), B (5 instances), C (7 instances)

Snapshot ที่เวลา T_0 :

	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			

ถ้าไม่ให้มาให้เอาค่า Allocation ของ P ทุกตัวมาลบกัน แล้วนำไปลบค่าทรัพยากรสูงสุดที่กำหนดให้

1. คำนวณ Need matrix
2. หาคำตอบของการทำงานของ process เพื่อดูว่า safe state ไหม
3. ถ้า P_1 ร้องขอทรัพยากร (1,0,2) จะให้ไหม
4. ถ้า P_4 ร้องขอทรัพยากร (3,3,0) จะให้ไหม
5. ถ้า P_0 ร้องขอทรัพยากร (0,2,0) จะให้ไหม

หน้าถัดไป

1. คำนวณ Need matrix

สูตรคือ Need = Max − Allocation // ความต้องการสูงสุด − ทรัพยากรที่ได้มาใช้แล้ว

Need

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

// $\overset{ABC}{753} - \overset{ABC}{010} = \overset{ABC}{743}$

2. หาลำดับของการทำงานของ process เพื่อดูว่า safe state ไหม

* ดูตัวที่ Need น้อยกว่า work (Available) ตัวไหนน้อยให้ตัวนั้นทำงาน และ + บวก Allocation เข้าไปใน work เมื่อจากทำงานเสร็จแล้วก็คืนทรัพยากร

Work = [3, 3, 2] → ค่า Available

P0 ว่า Need $\overset{ABC}{743}$ มากกว่า Work ดังนั้นยังไม่ทำ

ให้ P1 ทำงาน ∴ work = [5, 3, 2]

P1 ว่า Need < work ทำได้ Allocation + Available

ให้ P3 ทำงาน ∴ work = [7, 4, 3]

ให้ P0 ทำงาน ∴ work = [7, 5, 3]

ให้ P2 ทำงาน ∴ work = [10, 5, 5]

ให้ P4 ทำงาน ∴ work = [10, 5, 7]

∴ safe state = < P1, P3, P0, P2, P4 >

3. ถ้า P1 ร้องขอทรัพยากร (1, 0, 2) จะให้หรือไม่ให้

	Allocation A	B	C	Need A	B	C	available 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			

* เมื่อสมมุติให้ P1 ร้องของทรัพยากรเพิ่มได้แล้วเลขที่จะเปลี่ยนจะมีอยู่ 3 ตัวคือ ค่าเลขที่ร้องของเพิ่มไปบวกให้กับ Allocation แต่ค่าไปลบกับค่า available และคำนวนค่า Need ใหม่

ดังนั้น work = [2, 3, 0]

ให้ P1 ทำงาน ∴ work = [5, 3, 2]

ให้ P3 ทำงาน ∴ work = [7, 4, 3]

ให้ P0 ทำงาน ∴ work = [7, 5, 3]

ให้ P2 ทำงาน ∴ work = [10, 5, 5]

ให้ P4 ทำงาน ∴ work = [10, 5, 7]

∴ safe state = < P1, P3, P0, P2, P4 >

และให้ P1 สามารถร้องขอได้

4. ถ้า P4 ร้องขอทรัพยากร (3, 3, 0) จะให้หรือไม่

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

work = [0, 0, 2]

* สังเกตุว่า Need ทุกตัวมากกว่า work ทั้งหมดเลย ดังนั้น ∴ ระบบอยู่ในสภาวะ unsafe ถ้าเราอนุญาตให้ P4 ร้องขอได้

∴ ไม่ให้ P4 ร้องขอทรัพยากร (3, 3, 0)