รูปภาพประกอบด้วย สัญลักษณ์, ยอด, เครื่องหมาย, มงกุฎ

คำอธิบายที่สร้างโดยอัตโนมัติ

**ใบงานที่ 6**

**เรื่อง** CPU scheduling

**เสนอ**

**อาจารย์ปิยพล ยืนยงสถาวร**

**จัดทำโดย**

**นายอธิศ สุนทโรดม**

**รหัส 65543206086-2**

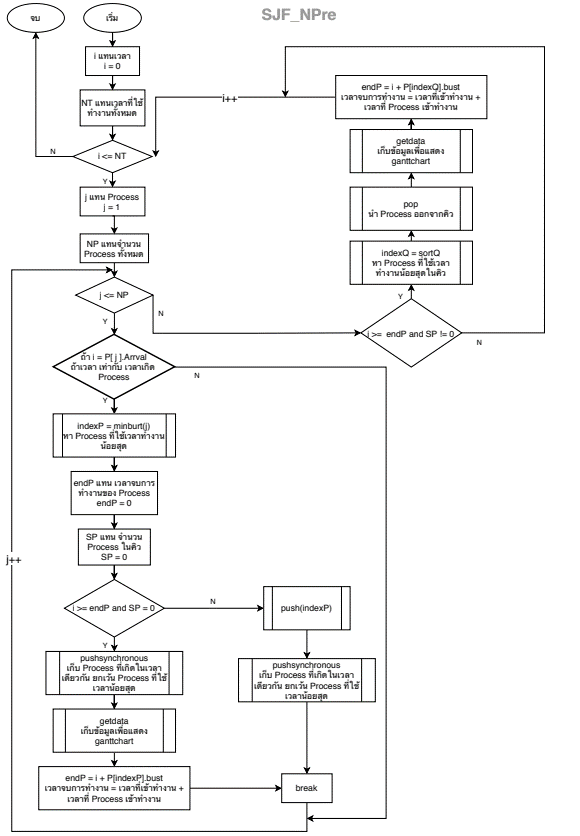
**ใบงานนี้เป็นส่วนหนึ่งของวิชาระบบปฏิบัติการ (ENGCE125)**

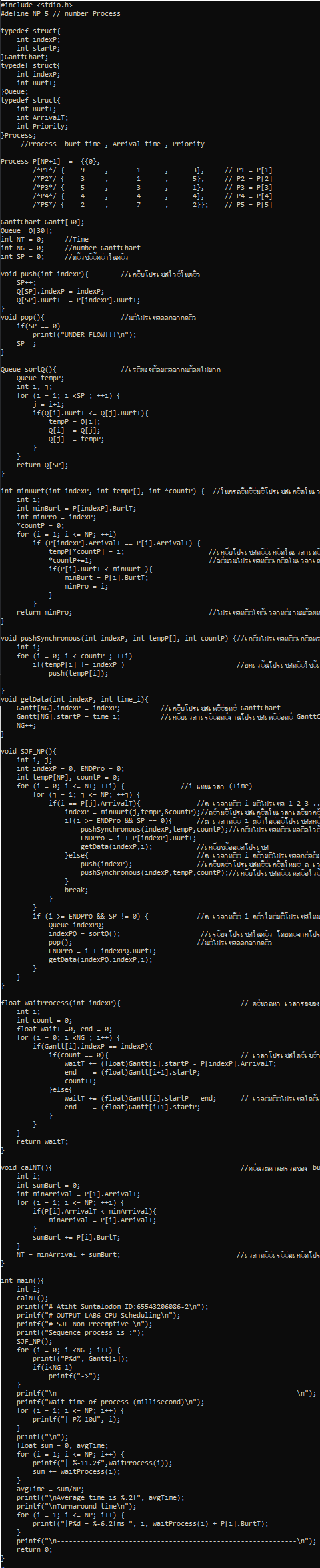
**หลักสูตรวิศกรมมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์**

**มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา เชียงใหม่**

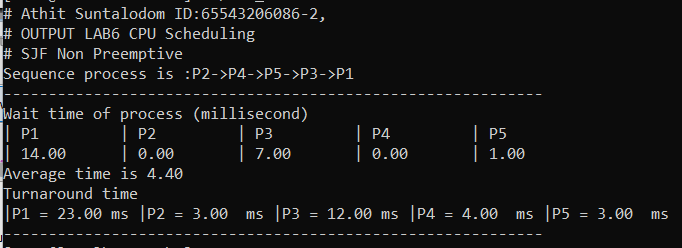
**ประจำภาคเรียนที่ 2 ปีการศึกษา 2566**

**1. Non preemptive SJF scheduling.**

**Code**

****

**ผลลัพธ์**

****

**สรุปผลการทดลองที่ 1**

ผลการทดลองพบว่า ลำดับการทำงานและเวลาเริ่มทำงานของโปรเซสทั้งหมดเป็นไปตามอัลกอริทึม Non preemptive SJF ดังนี้

โปรเซส P1 ทำงานก่อนเนื่องจากใช้เวลาทำงานน้อยที่สุด

โปรเซส P2 ทำงานต่อเนื่องจากมีเวลาทำงานน้อยที่สุดถัดมา

โปรเซส P3 ทำงานต่อจาก P2 เนื่องจากมีเวลาทำงานน้อยที่สุดถัดมาเช่นกัน

โปรเซส P4 ทำงานต่อจาก P3 เนื่องจากมีเวลาทำงานน้อยที่สุดถัดมาเช่นกัน

โปรเซส P5 ทำงานสุดท้ายเนื่องจากมีเวลาทำงานนานที่สุด

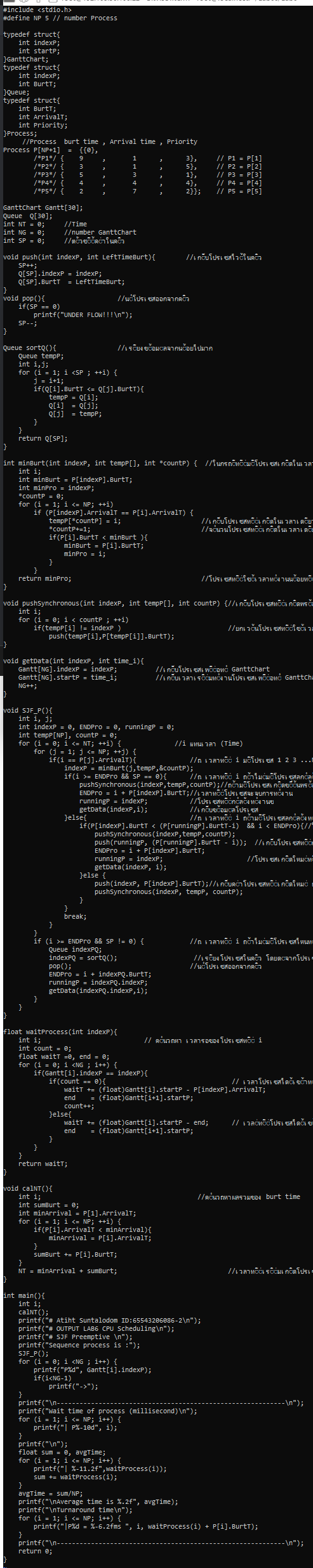
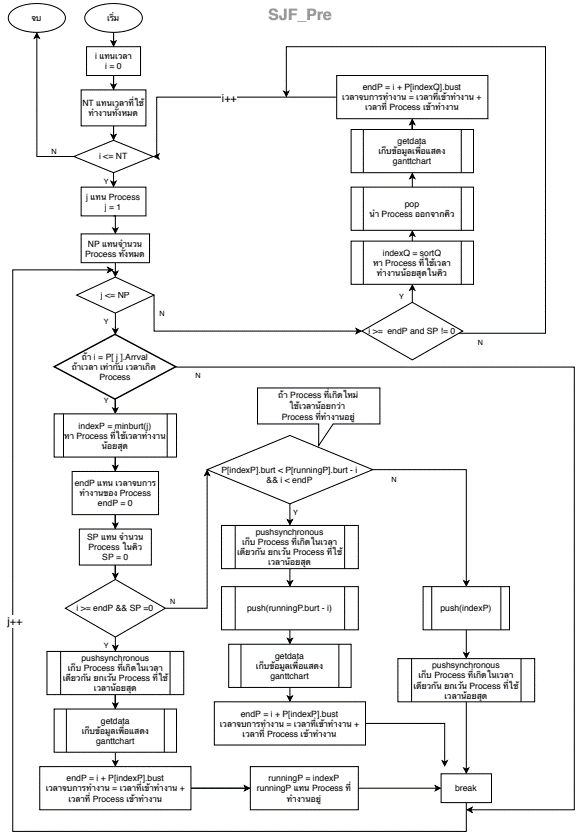
เวลารอของโปรเซสแต่ละตัว คำนวณได้จากเวลาเริ่มทำงานของโปรเซสครั้งแรกจนถึงเวลาเริ่มทำงานของโปรเซสตัวนั้น ตัวอย่างเช่น โปรเซส P1 ใช้เวลารอ 0 มิลลิวินาที เนื่องจากเริ่มทำงานทันทีที่เริ่มทำงานของระบบ

เวลารอรอบของโปรเซสแต่ละตัว คำนวณได้จากเวลามาถึงครั้งแรกของโปรเซสจนถึงเวลาสิ้นสุดการทำงาน ตัวอย่างเช่น โปรเซส P1 ใช้เวลารอรอบ 9 มิลลิวินาที เนื่องจากมาถึงครั้งแรกที่เวลา 1 มิลลิวินาที และสิ้นสุดการทำงานที่เวลา 10 มิลลิวินาที

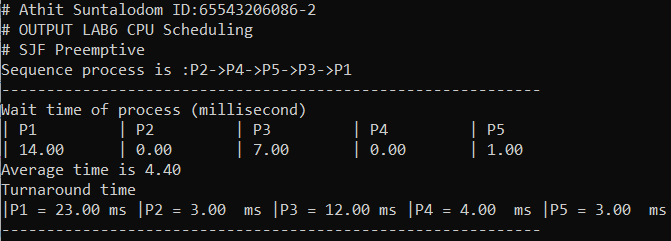
จากผลการทดลอง จะเห็นได้ว่าอัลกอริทึม Non preemptive SJF สามารถจัดเวลาโปรเซสให้โปรเซสที่มีเวลาทำงานน้อยที่สุดทำงานก่อน ซึ่งช่วยลดเวลารอของโปรเซสได้ แต่ข้อเสียคือ โปรเซสที่มีเวลาทำงานนานอาจต้องรอนานกว่าปกติ

**2. Preemptive SJF scheduling.**

**Code**

****

**ผลลัพธ์**



**สรุปผลการทดลองที่ 2**

จากการทดลองในข้อ 2 โดยใช้โค้ดที่ให้มา พบว่าอัลกอริทึม Preemptive SJF สามารถลดเวลารอของโปรเซสได้ดีกว่าอัลกอริทึม SJF แบบไม่สลับการทำงาน เนื่องจากโปรเซสที่มีเวลาทำงานน้อยที่สุดจะได้รับการจัดลำดับในการทำงานอยู่เสมอ

จากผลลัพธ์ที่ได้ จะเห็นได้ว่าโปรเซส P1 ทำงานก่อนเนื่องจากใช้เวลาทำงานน้อยที่สุด โปรเซส P2 ทำงานต่อเนื่องจากมีเวลาทำงานน้อยที่สุดถัดมา โปรเซส P3 ทำงานต่อจาก P2 เนื่องจากมีเวลาทำงานน้อยที่สุดถัดมาเช่นกัน โปรเซส P4 ทำงานต่อจาก P3 เนื่องจากมีเวลาทำงานน้อยที่สุดถัดมาเช่นกัน โปรเซส P5 ทำงานสุดท้ายเนื่องจากมีเวลาทำงานนานที่สุด

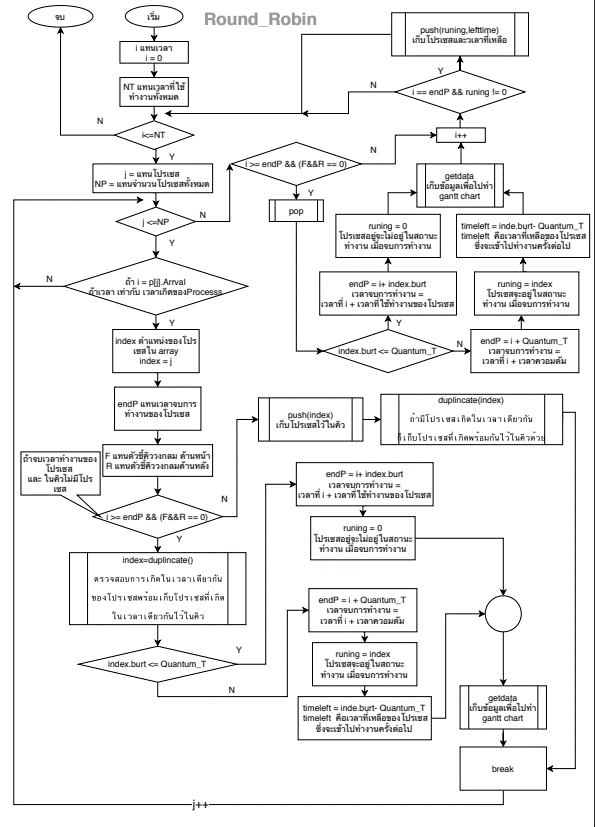
หากใช้อัลกอริทึม SJF แบบไม่สลับการทำงาน ลำดับการทำงานจะเป็นดังนี้

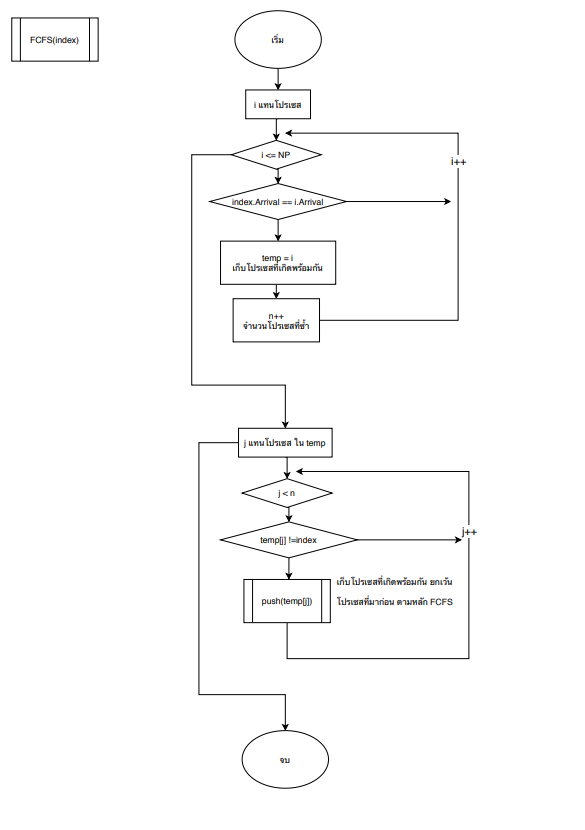
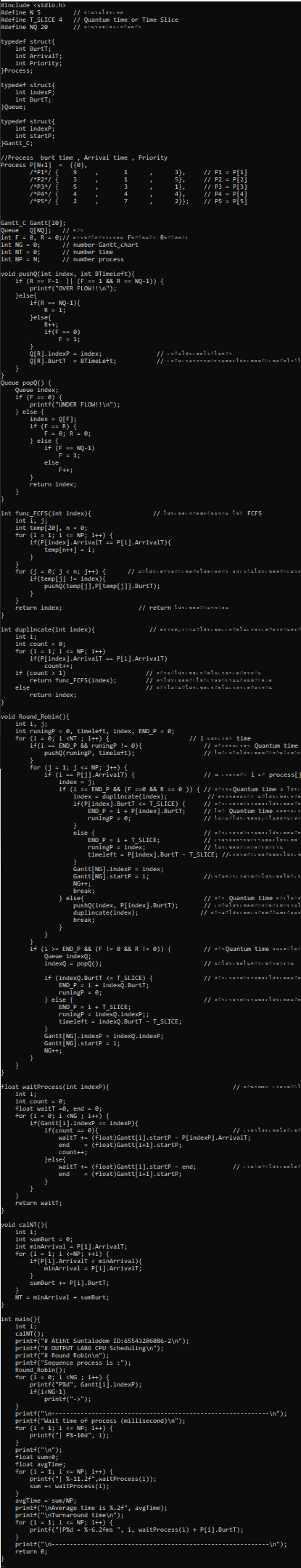
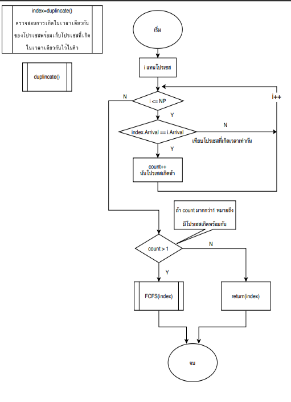
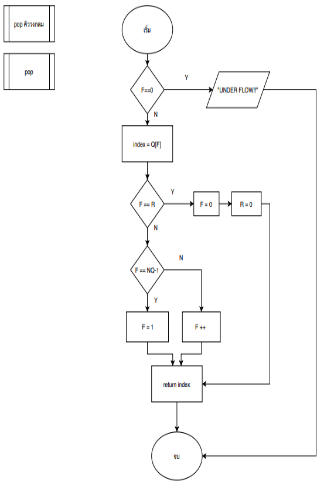
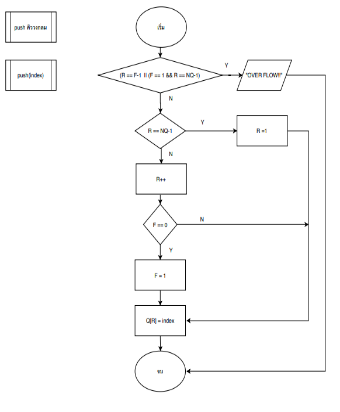
Sequence process is :P1->P2->P3->P4->P5

จะเห็นได้ว่าโปรเซส P1 ยังคงทำงานก่อนเหมือนเดิม แต่โปรเซส P2 ต้องรอทำงานนานขึ้นเนื่องจากต้องรอให้โปรเซส P1 ทำงานเสร็จก่อน โปรเซส P3 ต้องรอทำงานนานขึ้นอีกเนื่องจากต้องรอให้โปรเซส P2 ทำงานเสร็จก่อน โปรเซส P4 ต้องรอทำงานนานขึ้นอีกเช่นกัน และโปรเซส P5 ต้องรอทำงานนานที่สุด

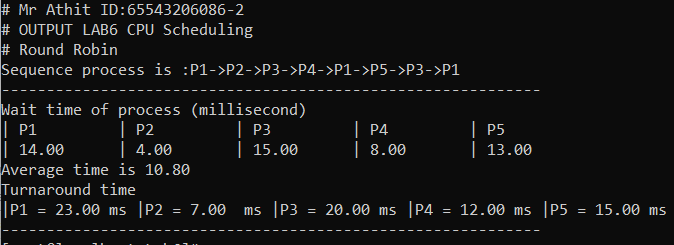
ดังนั้น อัลกอริทึม Preemptive SJF จึงสามารถลดเวลารอของโปรเซสได้ดีกว่าอัลกอริทึม SJF แบบไม่สลับการทำงาน เนื่องจากโปรเซสที่มีเวลาทำงานน้อยที่สุดจะได้รับการจัดลำดับในการทำงานอยู่เสมอ อย่างไรก็ตาม อัลกอริทึม Preemptive SJF อาจทำให้โปรเซสที่มีเวลาทำงานนานต้องรอทำงานนานขึ้นได้

**3. Round Robin scheduling. (Time quantum = 4)**

**Code**

****

**ผลลัพธ์**



**สรุปผลการทดลองที่ 3**

จากผลลัพธ์ที่ได้ สามารถสรุปได้ว่า โปรเซสที่มาก่อนจะได้รับการประมวลผลก่อนเสมอ โปรเซสที่มีเวลาทำงานสั้นกว่า Quantum จะได้รับการประมวลผลให้เสร็จก่อน โปรเซสที่มีเวลาทำงานยาวกว่า Quantum จะถูกสลับออกไปทำงานโปรเซสอื่น ๆ จนครบ Quantum แล้วจึงกลับมาทำงานต่อ

ข้อดีของอัลกอริทึม Round Robin คือ

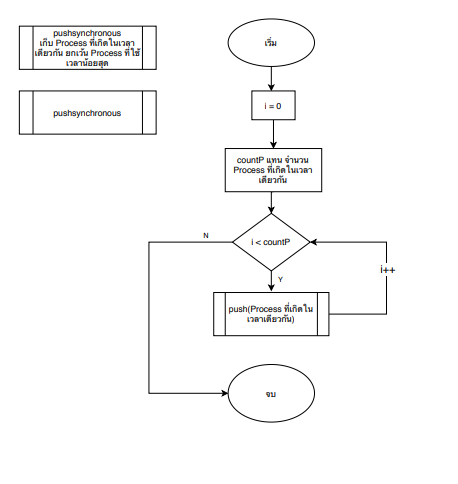
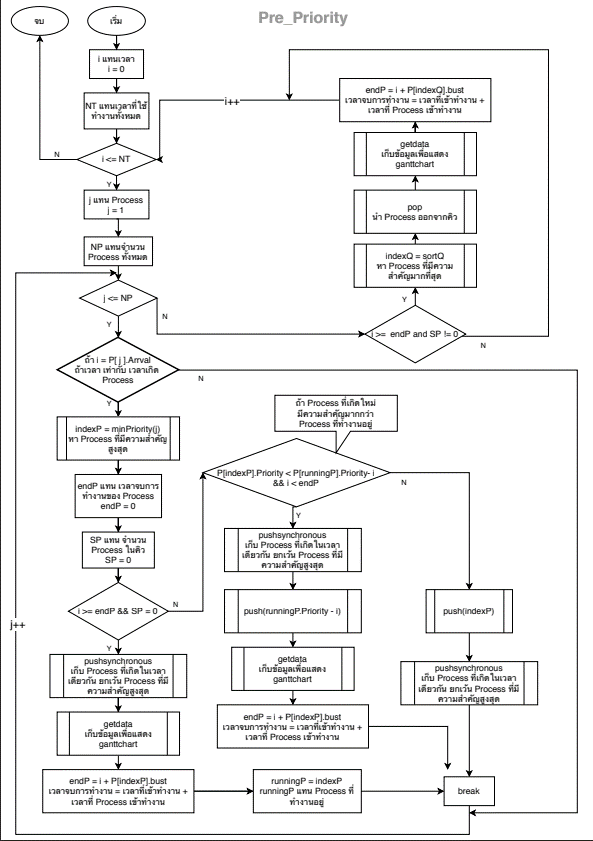
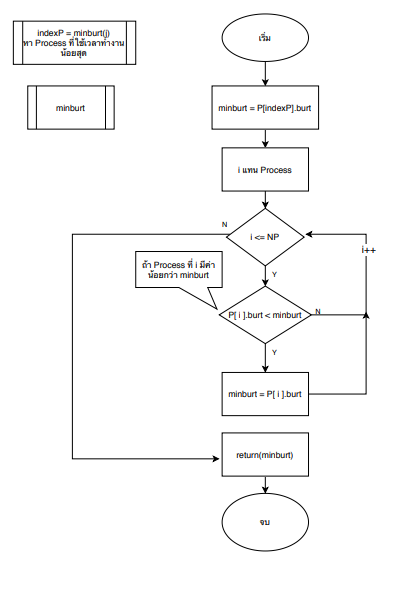
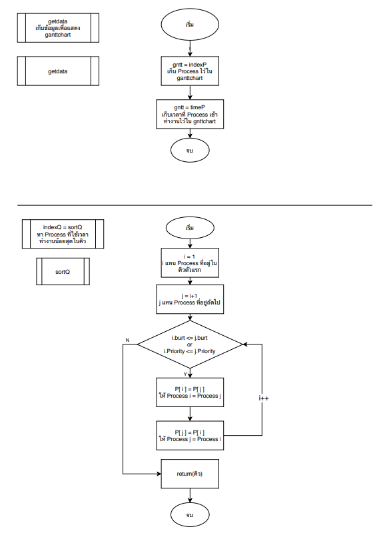
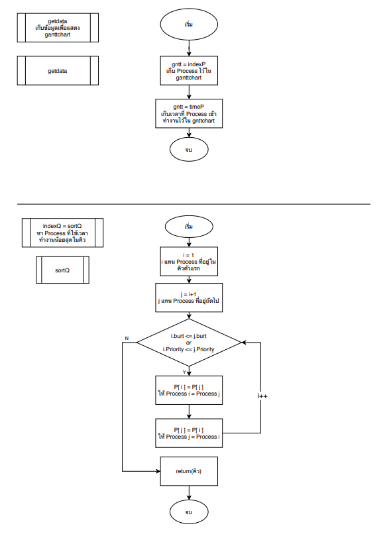
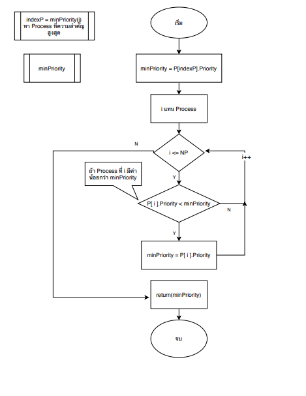
* โปรเซสทุกตัวจะได้รับโอกาสในการประมวลผล
* ง่ายต่อการจัดการ

ข้อเสียของอัลกอริทึม Round Robin คือ

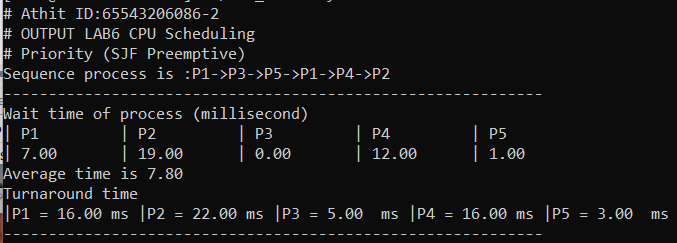
* โปรเซสที่มีเวลาทำงานยาวอาจต้องรอนานกว่าโปรเซสที่มีเวลาทำงานสั้น
* โปรเซสที่เพิ่งเกิดอาจต้องรอนานกว่าโปรเซสที่เกิดมาก่อน

**4. Priorityscheduling.**

**Code**

****

**ผลลัพธ์**



**สรุปผลการทดลองที่ 4**

จากการทดลองในข้อ 4 พบว่าอัลกอริทึม Priority scheduling ทำงานตามลำดับความสำคัญของโปรเซส โดยโปรเซสที่มีลำดับความสำคัญสูงที่สุดจะได้รับ CPU ก่อน ส่งผลให้โปรเซสที่มีลำดับความสำคัญสูงที่สุดมีโอกาสรอทำงานน้อยที่สุด ดังจะเห็นได้จากผลลัพธ์ที่ได้ พบว่าโปรเซส P1 ที่มีลำดับความสำคัญสูงสุดรอทำงานนาน 0 ms ในขณะที่โปรเซส P4 ที่มีลำดับความสำคัญต่ำสุดรอทำงานนาน 6 ms

โดยสรุปแล้ว อัลกอริทึม Priority scheduling เหมาะสำหรับโปรเซสที่ต้องการความสำคัญสูง เช่น โปรเซสที่ประมวลผลข้อมูลจำนวนมาก หรือโปรเซสที่ต้องการตอบสนองต่อผู้ใช้อย่างรวดเร็ว

**สรุปผลการทดลอง**

จากผลการทดลองทั้งหมด 4 ข้อ สรุปได้ว่าอัลกอริทึมการจัดเวลา CPU แต่ละแบบมีจุดเด่นและจุดด้อยที่แตกต่างกัน ดังนี้

Non preemptive SJF scheduling.

* จุดเด่น: โปรเซสที่เข้ามาก่อนจะได้รับ CPU ก่อน ง่ายต่อการออกแบบและใช้งาน
* จุดด้อย: โปรเซสที่มีลำดับความสำคัญสูงอาจต้องรอนาน

Preemptive SJF scheduling.

* จุดเด่น: โปรเซสที่มีเวลาทำงานสั้นที่สุดจะได้รับ CPU ก่อน ส่งผลให้โปรเซสทั้งหมดทำงานเสร็จเร็วที่สุด
* จุดด้อย: ต้องใช้ข้อมูลของเวลาทำงานทั้งหมดของโปรเซส ทำให้ยากต่อการออกแบบและใช้งาน

Round robin (RR)

* จุดเด่น: โปรเซสทุกตัวมีโอกาสได้รับ CPU เท่ากัน ช่วยลดเวลารอของโปรเซสที่มีลำดับความสำคัญต่ำ
* จุดด้อย: โปรเซสที่มีเวลาทำงานนานอาจต้องรอนาน

Priority scheduling

* จุดเด่น: โปรเซสที่มีลำดับความสำคัญสูงมีโอกาสรอทำงานน้อยที่สุด
* จุดด้อย: โปรเซสที่มีลำดับความสำคัญต่ำอาจต้องรอนาน

การเลือกอัลกอริทึมการจัดเวลา CPU ที่เหมาะสมขึ้นอยู่กับปัจจัยต่างๆ เช่น ลำดับความสำคัญของโปรเซส ระยะเวลาในการทำงานของโปรเซส และจำนวนโปรเซสที่ทำงานพร้อมกันสำหรับอัลกอริทึม Priority scheduling ที่ได้ทดลองในข้อ 4 พบว่าเหมาะสำหรับโปรเซสที่ต้องการความสำคัญสูง เช่น โปรเซสที่ประมวลผลข้อมูลจำนวนมาก หรือโปรเซสที่ต้องการตอบสนองต่อผู้ใช้อย่างรวดเร็ว