ขั้นตอนการทำงาน **SJF Preemptive**

1. กำหนดค่าเริ่มต้น
   * กำหนดจำนวนโปรเซส (NP)
   * กำหนดคุณสมบัติของโปรเซส (BurtT, ArrivalT, Priority)
   * คำนวณ NT = minArrival + sumBurtT
2. วนลูปตามเวลา (i = 0; i <= NT; i++)
   * ตรวจสอบว่ามีโปรเซสเกิดใหม่ (P[j].ArrivalT == i)
     + ถ้ามี
       - หาโปรเซสที่มีเวลาทำงานน้อยที่สุด (minBurt(indexP, tempP, &countP))
       - ถ้าไม่มีโปรเซสทำงานอยู่ (i >= ENDPro && SP == 0)
         * เก็บโปรเซสใหม่ไว้ในคิว และเริ่มทำงาน (ENDPro = i + P[indexP].BurtT, runningP = indexP, getData(indexP, i))
       - ถ้ามีโปรเซสทำงานอยู่ (i >= ENDPro && SP != 0)
         * ถ้าโปรเซสใหม่ใช้เวลาทำงานน้อยกว่าโปรเซสที่กำลังทำงานอยู่

เก็บโปรเซสที่กำลังทำงานไว้ในคิว และเริ่มทำงานด้วยโปรเซสใหม่ (ENDPro = i + P[indexP].BurtT, runningP = indexP, getData(indexP, i))

* + - * + ถ้าโปรเซสใหม่ใช้เวลาทำงานเท่ากันหรือนานกว่าโปรเซสที่กำลังทำงานอยู่

เก็บโปรเซสใหม่ไว้ในคิว

* + - ถ้าไม่มี
      * ถ้าไม่มีโปรเซสทำงานอยู่ (i >= ENDPro && SP == 0)
        + เก็บโปรเซสในคิวที่ทำงานน้อยที่สุด (indexPQ = sortQ())
        + นำโปรเซสออกจากคิว และเริ่มทำงาน (ENDPro = i + indexPQ.BurtT, runningP = indexPQ.indexP, getData(indexPQ.indexP, i))
      * ถ้ามีโปรเซสทำงานอยู่ (i >= ENDPro && SP != 0)
        + ไม่มีการทำงานใดๆ

1. คำนวณเวลารอและเวลาทำงานเสร็จ
   * วนลูปตามโปรเซส (i = 1; i <= NP; i++)
     + คำนวณเวลารอ (waitProcess(i))
     + คำนวณเวลาทำงานเสร็จ (waitProcess(i) + P[i].BurtT)
2. แสดงผลลัพธ์
   * แสดงลำดับการทำงาน
   * แสดงเวลารอของโปรเซส
   * แสดงเวลาทำงานเสร็จของโปรเซส

ขั้นตอนการทำงาน **SJF Non Preemptive**

1. กำหนดค่าเริ่มต้น
   * กำหนดจำนวนโปรเซส (NP)
   * กำหนดค่า arrival time และ burst time ของทุกโปรเซส
   * กำหนดค่า time
   * กำหนดค่า GanttChart ทั้งหมดเป็น 0
   * กำหนดค่า queue เป็น 0
2. วนลูปจนกว่า time ถึง NT
   * วนลูปทุกโปรเซส
     + ถ้า arrival time ของโปรเซสเท่ากับ time
       - หาโปรเซสที่ใช้เวลาทำงานน้อยที่สุดจากโปรเซสที่เกิดในเวลาเดียวกัน
       - ถ้า queue เป็น 0 และ time >= ENDPro
         * เก็บโปรเซสที่ใช้เวลาทำงานน้อยที่สุดลงใน queue
       - ถ้า queue != 0
         * เก็บโปรเซสที่ใช้เวลาทำงานน้อยที่สุดลงใน queue
3. ถ้า queue != 0 และ time >= ENDPro
   * ดึงโปรเซสที่ใช้เวลาทำงานน้อยที่สุดออกจาก queue
   * เพิ่ม time เท่ากับ burst time ของโปรเซสที่ดึงออกมา
   * เก็บข้อมูลโปรเซสที่ดึงออกมาลงใน GanttChart
4. วนลูปข้อ 2
5. คำนวณค่ารอ (wait time) และเวลารอทั้งหมด (turnaround time)
   * สำหรับทุกโปรเซส
     + วนลูป GanttChart
       - ถ้า index ของ GanttChart เท่ากับ index ของโปรเซส
         * ถ้า count = 0

wait time = time - arrival time

count++

* + - * + else

wait time = time - end

end = time

* + คำนวณเวลารอทั้งหมด (turnaround time) = wait time + burst time

1. แสดงผลลัพธ์
   * แสดง GanttChart
   * แสดงค่ารอ (wait time)
   * แสดงเวลารอทั้งหมด (turnaround time)

ขั้นตอนการทำโปรแกรม **Round Robin**

1. กำหนดตัวแปรที่จำเป็น
   * N: จำนวนโปรเซส
   * T\_SLICE: Quantum time
   * NQ: จำนวนช่องเก็บคิว
   * F: ตัวชี้คิววงกลม Fชี้หน้า Rชี้หลัง
   * NG: number Gantt\_chart
   * NT: number time
   * NP: number process
2. กำหนดฟังก์ชันที่จำเป็น
   * pushQ(): เก็บโปรเซสไว้ในคิว
   * popQ(): นำโปรเซสออกจากคิว
   * func\_FCFS(): หาโปรเซสที่มาก่อน (ถ้ามีโปรเซสเกิดพร้อมกัน)
   * Round\_Robin(): ทำงานแบบ Round Robin
   * waitProcess(): คำนวณหา เวลาที่โปรเซสรอ
   * calNT(): คำนวณหาจำนวนรอบ
3. เริ่มต้นโปรแกรม
   * กำหนดค่าตัวแปร
   * คำนวณหาจำนวนรอบ
4. วนลูปตามจำนวนรอบ
   * ตรวจสอบว่าโปรเซสใดครบรอบ Quantum time
   * ถ้ามีโปรเซสครบรอบ ให้เก็บโปรเซสในคิว
5. ตรวจสอบว่ามีโปรเซสเกิดใหม่
   * ถ้ามีโปรเซสเกิดใหม่ ให้เก็บโปรเซสในคิว
6. เลือกโปรเซสที่จะทำงาน
   * ถ้าไม่มีโปรเซสอยู่ในคิว ให้หยุดการทำงานของโปรแกรม
   * ถ้ามีโปรเซสอยู่ในคิว ให้นำโปรเซสออกจากคิว
7. ทำงานโปรเซสที่กำลังทำงาน
   * ถ้าเวลาทำงานของโปรเซสหมด ให้หยุดการทำงานของโปรเซส
   * ถ้าเวลาทำงานของโปรเซสยังไม่หมด ให้ทำงานโปรเซสต่อไป
8. วนลูปกลับไปที่ข้อ 4
9. แสดงผลลัพธ์
   * แสดงลำดับการทำงานของโปรเซส
   * แสดงเวลารอของโปรเซสแต่ละตัว
   * แสดงเวลาตอบสนองของโปรเซสแต่ละตัว

รายละเอียดเพิ่มเติม

* ข้อ 1 กำหนดตัวแปรที่จำเป็น ได้แก่
  + N: จำนวนโปรเซส
  + T\_SLICE: Quantum time
  + NQ: จำนวนช่องเก็บคิว
  + F: ตัวชี้คิววงกลม Fชี้หน้า Rชี้หลัง
  + NG: number Gantt\_chart
  + NT: number time
  + NP: number process
* ข้อ 2 กำหนดฟังก์ชันที่จำเป็น ได้แก่
  + pushQ(): เก็บโปรเซสไว้ในคิว
  + popQ(): นำโปรเซสออกจากคิว
  + func\_FCFS(): หาโปรเซสที่มาก่อน (ถ้ามีโปรเซสเกิดพร้อมกัน)
  + Round\_Robin(): ทำงานแบบ Round Robin
  + waitProcess(): คำนวณหา เวลาที่โปรเซสรอ
  + calNT(): คำนวณหาจำนวนรอบ
* ข้อ 3 เริ่มต้นโปรแกรม โดยกำหนดค่าตัวแปร และคำนวณหาจำนวนรอบ
* ข้อ 4 วนลูปตามจำนวนรอบ โดยตรวจสอบว่าโปรเซสใดครบรอบ Quantum time ถ้ามีโปรเซสครบรอบ ให้เก็บโปรเซสในคิว
* ข้อ 5 ตรวจสอบว่ามีโปรเซสเกิดใหม่ ถ้ามีโปรเซสเกิดใหม่ ให้เก็บโปรเซสในคิว
* ข้อ 6 เลือกโปรเซสที่จะทำงาน โดยตรวจสอบว่าไม่มีโปรเซสอยู่ในคิว ถ้าไม่มีโปรเซสอยู่ในคิว ให้หยุดการทำงานของโปรแกรม แต่ถ้ามีโปรเซสอยู่ในคิว ให้นำโปรเซสออกจากคิว
* ข้อ 7 ทำงานโปรเซสที่กำลังทำงาน โดยตรวจสอบว่าเวลาทำงานของโปรเซสหมด ถ้าหมด ให้หยุดการทำงานของโปรเซส แต่ถ้ายังไม่หมด ให้ทำงานโปรเซสต่อไป
* ข้อ 8 วนลูปกลับไปที่ข้อ 4
* ข้อ 9 แสดงผลลัพธ์ โดยแสดงลำดับการทำงานของโปรเซส แสดงเวลารอของโปรเซสแต่ละตัว และแสดงเวลาตอบสนองของโปรเซสแต่ละตัว

ขั้นตอนการทำงานแบบ Priority (SJF Preemptive)

1. กำหนดค่าตัวแปรเริ่มต้น
   * กำหนดจำนวนโปรเซส (NP)
   * กำหนดเวลาสิ้นสุด (NT)
   * กำหนดตัวชี้ค่าในคิว (SP)
   * กำหนดค่าตัวแปรสำหรับคำนวณเวลารอ (waitT) และเวลาจบการทำงาน (end)
2. คำนวณหาผลรวมของ burt time
   * วนลูปนับโปรเซสตั้งแต่ 1 ถึง NP
   * ถ้าเวลามาถึงโปรเซสตัวที่ i แล้ว
     + ถ้าเวลามาถึงโปรเซสตัวที่ i น้อยกว่าเวลามาถึงโปรเซสตัวที่ minArrival
       - minArrival = เวลามาถึงโปรเซสตัวที่ i
   * วนลูปนับโปรเซสตั้งแต่ 1 ถึง NP
     + sumBurt += P[i].BurtT
   * NT = minArrival + sumBurt
3. วนลูปตามเวลา (i)
   * ตั้งแต่ i = 0 ถึง NT
     + วนลูปนับโปรเซสตั้งแต่ 1 ถึง NP
       - ถ้าเวลามาถึงโปรเซสตัวที่ i แล้ว
         * ถ้าคิวว่าง

เก็บโปรเซสตัวที่ i ไว้ในคิว

เรียงโปรเซสในคิว โดยดูจากโปรเซสที่มีลำดับความสำคัญสูงที่สุด

นำโปรเซสออกจากคิว

เก็บข้อมูลโปรเซส

* + - * + ถ้าคิวไม่ว่าง

ถ้าลำดับความสำคัญของโปรเซสตัวที่ i สูงกว่าลำดับความสำคัญของโปรเซสที่กำลังทำงาน

เก็บโปรเซสที่กำลังทำงานไว้ในคิว

นำโปรเซสตัวที่ i ออกจากคิว

เก็บข้อมูลโปรเซส

* + ถ้าไม่มีโปรเซสทำงาน และยังมีโปรเซสอยู่ในคิว
    - เรียงโปรเซสในคิว โดยดูจากโปรเซสที่มีลำดับความสำคัญสูงที่สุด
    - นำโปรเซสออกจากคิว
    - เก็บข้อมูลโปรเซส

1. คำนวณเวลารอของโปรเซส
   * วนลูปนับโปรเซสตั้งแต่ 1 ถึง NP
     + วนลูปนับข้อมูล GanttChart ตั้งแต่ 0 ถึง NG
       - ถ้าข้อมูล GanttChart ลำดับที่ i หมายถึงโปรเซสตัวที่ j
         * ถ้า i == 0 waitT = ข้อมูล GanttChart ลำดับที่ i - P[j].ArrivalT
         * ถ้า i > 0 waitT = ข้อมูล GanttChart ลำดับที่ i - ข้อมูล GanttChart ลำดับที่ i-1
2. แสดงผล
   * แสดงลำดับการทำงาน
   * แสดงเวลารอของโปรเซส
   * แสดงเวลาจบการทำงาน

ตัวอย่าง Flowchart

เริ่ม

กำหนดค่าตัวแปรเริ่มต้น

คำนวณหาผลรวมของ burt time

วนลูปตามเวลา

วนลูปนับโปรเซส ถ้าเวลามาถึงโปรเซสตัวที่ i แล้ว ถ้าคิวว่าง

เก็บโปรเซสตัวที่ i ไว้ในคิว

เรียงโปรเซสในคิว โดยดูจากโปรเซสที่มีลำดับความสำคัญสูงที่สุด

นำโปรเซสออกจากคิว

เก็บข้อมูลโปรเซส ถ้าคิวไม่ว่าง ถ้าลำดับความสำคัญของโปรเซสตัวที่ i สูงกว่าลำดับความสำคัญของโปรเซสที่กำลังทำงาน

เก็บโปรเซสที่กำลังทำงานไว้ในคิว

นำโปรเซสตัวที่ i ออกจากคิว

เก็บข้อมูลโปรเซส ถ้าไม่มีโปรเซสทำงาน และยังมีโปรเซสอยู่ในคิว เรียงโปรเซสในคิว โดยดูจากโปรเซสที่มีลำดับความสำคัญสูงที่สุด นำโปรเซสออกจากคิว

เก็บข้อมูลโปรเซส

คำนวณเวลารอของโปรเซส

แสดงผล

จบ

หมายเหตุ

* ขั้นตอนการทำงานที่อธิบายข้างต้นเป็นการอธิบายอย่างละเอียด เพื่อความเข้าใจง่าย ในทางปฏิบัติอาจมีการเขียนโปรแกรมให้สั้นลงได้
* การคำนวณหาเวลารอของโปรเซสอาจทำได้หลายวิธี ในที่นี้ใช้วิธีคำนวณเวลารอจากเวลาที่โปรเซสได้เข้าทำงานครั้งแรกจนถึงเวลาที่โปรเซสจบการทำงาน