โปรแกรมนี้ใช้อัลกอริทึม Pre\_Priority เพื่อจัดเวลาซีพียูให้กับโปรเซส โดยกำหนดให้โปรเซสที่มีเวลาทำงานที่เหลือน้อยที่สุดจะได้รับสิทธิ์ในการทำงานก่อน

ฟังก์ชัน push() จะใช้เพื่อเก็บโปรเซสไว้ในคิว โดยระบุ index ของโปรเซสและเวลาทำงานที่เหลือ

ฟังก์ชัน pop() จะใช้เพื่อนำโปรเซสออกจากคิว

ฟังก์ชัน sortQ() จะใช้เพื่อเรียงลำดับโปรเซสในคิว โดยดูจากโปรเซสที่มีเวลาทำงานที่เหลือน้อยที่สุด

ฟังก์ชัน minPriority() จะใช้เพื่อหาโปรเซสที่มีลำดับความสำคัญสูงสุดจากกลุ่มโปรเซสที่เกิดในเวลาเดียวกัน

ฟังก์ชัน pushSynchronous() จะใช้เพื่อเก็บโปรเซสที่เกิดพร้อมกันไว้ในคิว โดยยกเว้นโปรเซสที่มีลำดับความสำคัญสูงสุด

ฟังก์ชัน getData() จะใช้เพื่อเก็บข้อมูลโปรเซสเพื่อทำ Gantt Chart โดยระบุ index ของโปรเซสและเวลาเริ่มทำงาน

ฟังก์ชัน calNT() จะใช้เพื่อคำนวณหาผลรวมของเวลาทำงานของโปรเซส

ฟังก์ชัน P\_Priority() จะเป็นฟังก์ชันหลักของโปรแกรม โดยจะใช้อัลกอริทึม SJF Preemptive เพื่อจัดเวลาซีพียูให้กับโปรเซส

ฟังก์ชัน waitProcess() จะใช้เพื่อคำนวณหาเวลารอของโปรเซส โดยคำนวณจากเวลาเริ่มทำงานของโปรเซสและเวลาเกิดของโปรเซส

โปรแกรมจะทำงานดังนี้

คำนวณหาผลรวมของเวลาทำงานของโปรเซส

เรียกใช้ฟังก์ชัน P\_Priority()

โปรแกรมจะวนลูปตรวจสอบโปรเซสทีละตัว

หากโปรเซสที่กำลังทำงานมีเวลาทำงานเหลือน้อยที่สุด และไม่มีโปรเซสอื่นที่พร้อมทำงานอยู่

โปรเซสที่กำลังทำงานจะทำงานต่อไปจนเสร็จ

หากโปรเซสที่กำลังทำงานมีเวลาทำงานเหลือไม่น้อยกว่าโปรเซสอื่นที่พร้อมทำงานอยู่

โปรเซสที่มีลำดับความสำคัญสูงสุดจะได้รับการยกเลิกการทำงาน และโปรเซสที่มีลำดับความสำคัญสูงสุดจะทำงานแทน

โปรแกรมจะวนลูปไปเรื่อยๆ จนกว่าโปรเซสทุกตัวจะทำงานเสร็จ

โค้ดนี้ใช้สำหรับจำลองการวางแผนงานแบบ Round Robin โดยกำหนดจำนวนโปรเซส N = 5 และกำหนด Quantum time T\_SLICE = 4

ฟังก์ชัน calNT() จะคำนวณหาจำนวนเวลาทั้งหมด (NT) โดยพิจารณาจากเวลาที่เร็วที่สุดที่โปรเซสเกิด (minArrival) และเวลาทำงานรวมของโปรเซสทั้งหมด (sumBurt)

ฟังก์ชัน pushQ() จะเพิ่มโปรเซสเข้าไปในคิว โดยกำหนด index ของโปรเซสและเวลาทำงานที่เหลือ

ฟังก์ชัน popQ() จะลบโปรเซสออกจากคิว โดยส่งคืนโปรเซสที่ลบออก

ฟังก์ชัน duplincate() จะตรวจดูว่ามีโปรเซสเกิดขึ้นในเวลาเดียวกันหรือไม่ โดยพิจารณาจากเวลาที่โปรเซสเกิด หากมีโปรเซสเกิดขึ้นพร้อมกัน จะหาโปรเซสที่ใช้เวลาทำงานน้อยที่สุด

ฟังก์ชัน Round\_Robin() จะทำงานวนซ้ำไปเรื่อยๆ จนถึงเวลาทั้งหมด (NT) โดยในแต่ละรอบจะพิจารณาสิ่งต่อไปนี้

หากโปรเซสที่กำลังทำงานอยู่มีเวลาทำงานเหลือน้อยกว่าหรือเท่ากับ Quantum time จะถือว่าโปรเซสเสร็จสิ้นการทำงาน

หากโปรเซสที่กำลังทำงานอยู่มีเวลาทำงานเหลือมากกว่า Quantum time จะถือว่าโปรเซสยังไม่เสร็จสิ้นการทำงาน แต่จะเก็บโปรเซสไว้ในคิวก่อน

หากไม่มีโปรเซสที่กำลังทำงานอยู่ และไม่มีโปรเซสอยู่ในคิว จะถือว่า Quantum time จบลง

ฟังก์ชัน waitProcess() จะคำนวณหาเวลาที่โปรเซสรอ โดยพิจารณาจากเวลาที่โปรเซสเริ่มทำงานและเวลาที่โปรเซสเกิด

ฟังก์ชันหลัก main() จะเรียกฟังก์ชันต่างๆ ตามที่อธิบายไว้ข้างต้น และแสดงผลลัพธ์ดังนี้

ลำดับการทำงานของโปรเซส

เวลาที่โปรเซสรอ

เวลารอเฉลี่ย

เวลารอรอบ (Turnaround time)

โปรแกรมนี้ใช้สำหรับจำลองระบบการวางแผนงานสำหรับ CPU แบบ SJF\_NPre

โปรแกรมเริ่มต้นด้วยการกำหนดจำนวนโปรเซส (NP) และกำหนดค่าเริ่มต้นให้กับตัวแปรต่างๆ จากนั้นจะเรียกใช้ฟังก์ชัน calNT() เพื่อคำนวณหาผลรวมของ burt time ของโปรเซสทั้งหมด

ฟังก์ชัน calNT() จะเริ่มต้นจากโปรเซสตัวที่ 1 และวนซ้ำไปเรื่อยๆ จนถึงโปรเซสตัวที่ NP หากพบโปรเซสที่มี arrival time น้อยกว่าค่าของตัวแปร minArrival ตัวแปร minArrival จะถูกแทนด้วยค่า arrival time ของโปรเซสตัวนั้น จากนั้นตัวแปร sumBurt จะถูกบวกด้วย burt time ของโปรเซสตัวนั้น

เมื่อวนซ้ำครบทุกโปรเซสแล้ว ตัวแปร NT จะถูกกำหนดให้เท่ากับค่า arrival time ของโปรเซสตัวแรกบวกกับผลรวมของ burt time ของโปรเซสทั้งหมด

จากนั้นโปรแกรมจะเรียกใช้ฟังก์ชัน SJF\_NP() เพื่อเริ่มการจำลองระบบการวางแผนงาน

ฟังก์ชัน SJF\_NP() จะเริ่มต้นด้วยการกำหนดค่าเริ่มต้นให้กับตัวแปรต่างๆ จากนั้นจะวนซ้ำไปเรื่อยๆ จนถึงค่าของตัวแปร NT

ในแต่ละรอบของการวนซ้ำ ฟังก์ชัน SJF\_NP() จะดำเนินการดังต่อไปนี้

ตรวจสอบว่า ณ เวลาปัจจุบัน มีโปรเซสใดเกิดใหม่หรือไม่ หากมี ฟังก์ชัน SJF\_NP() จะเรียกใช้ฟังก์ชัน minBurt() เพื่อหาโปรเซสที่ใช้เวลาทำงานน้อยที่สุดจากโปรเซสทั้งหมดที่เกิดใหม่

หากไม่มีโปรเซสใดทำงานอยู่และไม่มีโปรเซสใดรออยู่ในคิว ฟังก์ชัน SJF\_NP() จะเรียกใช้ฟังก์ชัน push() เพื่อเก็บโปรเซสที่เพิ่งเกิดใหม่ไว้ในคิว

หากไม่มีโปรเซสใดทำงานอยู่ แต่มีโปรเซสรออยู่ในคิว ฟังก์ชัน SJF\_NP() จะเรียกใช้ฟังก์ชัน sortQ() เพื่อเรียงโปรเซสในคิว โดยเรียงจากโปรเซสที่ใช้เวลาทำงานน้อยที่สุด

หากมีโปรเซสทำงานอยู่ ฟังก์ชัน SJF\_NP() จะตรวจสอบว่าโปรเซสตัวนั้นทำงานเสร็จสิ้นแล้วหรือไม่ หากเสร็จสิ้นแล้ว ฟังก์ชัน SJF\_NP() จะเรียกใช้ฟังก์ชัน pop() เพื่อนำโปรเซสออกจากคิว

เมื่อวนซ้ำครบทุกรอบแล้ว ฟังก์ชัน SJF\_NP() จะสิ้นสุดการทำงาน

หลังจากฟังก์ชัน SJF\_NP() สิ้นสุดการทำงาน โปรแกรมจะแสดงผลลัพธ์ดังต่อไปนี้

ลำดับการทำงานของโปรเซส

เวลารอของโปรเซสแต่ละตัว

เวลารอเฉลี่ย

เวลา turnaround ของโปรเซสแต่ละตัว

โปรแกรมนี้ใช้สำหรับจำลองการทำงานแบบ SJF Preemptive ของ CPU scheduling

ฟังก์ชัน calNT() คำนวณหาผลรวมของ burst time ของทุกโปรเซส เพื่อใช้หาเวลาสิ้นสุดการทำงานทั้งหมด (Total Time)

ฟังก์ชัน SJF\_P() ทำหน้าที่จำลองการทำงานแบบ SJF Preemptive โดยทำตามขั้นตอนดังนี้

วนลูปตั้งแต่เวลา 0 จนถึงเวลาสิ้นสุดการทำงานทั้งหมด

สำหรับทุกเวลา

ตรวจสอบว่ามีโปรเซสเกิดใหม่หรือไม่ ถ้ามี ก็ทำตามขั้นตอนที่ 3

ตรวจสอบว่ามีโปรเซสใดกำลังทำงานอยู่หรือไม่ ถ้ามี

ถ้าโปรเซสที่กำลังทำงานอยู่มี burst time น้อยกว่าโปรเซสที่เกิดใหม่ ก็ทำตามขั้นตอนที่ 4

ถ้าโปรเซสที่กำลังทำงานอยู่มี burst time มากกว่าหรือเท่ากับโปรเซสที่เกิดใหม่ ก็ทำตามขั้นตอนที่ 5

เก็บข้อมูลโปรเซสที่เกิดใหม่ไว้ในคิว พร้อมทั้งเวลาเริ่มต้นการทำงาน

หยุดโปรเซสที่กำลังทำงานอยู่ แล้วเก็บข้อมูลโปรเซสที่กำลังทำงานอยู่ไว้ในคิว พร้อมทั้งเวลาที่เหลือ

เก็บข้อมูลโปรเซสที่กำลังทำงานอยู่ไว้ในคิว พร้อมทั้งเวลาที่เหลือ

ฟังก์ชัน getData() ทำหน้าที่เก็บข้อมูลโปรเซสเพื่อทำ Gantt Chart โดยเก็บข้อมูลดังนี้

ลำดับของโปรเซส

เวลาเริ่มต้นการทำงาน

ฟังก์ชัน waitProcess() ทำหน้าที่คำนวณหาเวลารอของโปรเซส โดยทำตามขั้นตอนดังนี้

วนลูปตั้งแต่จุดเริ่มต้นของโปรเซสจนถึงจุดสิ้นสุดของโปรเซส

คำนวณหาเวลารอของโปรเซส ณ เวลานั้น

เก็บเวลารอของโปรเซส ณ เวลานั้น

ฟังก์ชัน main() ทำหน้าที่แสดงผลลัพธ์ โดยแสดงผลดังนี้

ลำดับของโปรเซสตาม Gantt Chart

เวลารอของทุกโปรเซส

เวลารอเฉลี่ย

เวลารอรอบของทุกโปรเซส