คำตอบต่อปฏิบัติการที่ 7

ศิระกร ลำใย (5910500023) และกรวิชญ์ ชัยกังวาฬ (5910501909)

ข้อที่ 1

ผลจากการคาดเดาก่อนรันโปรแกรมมีดังนี้

- โปรแกรมแสดงผลคำว่า “Hello from Foo” และ “Hello from Bar” โดย**ไม่่**เรียงลำดับกันเสมอ (มีโอกาสที่อันไหนจะขึ้นก่อนก็ได้)
    - ข้อความทั้งสอง จะได้ ID เป็น 0 และ 1 ตามลำดับ
- คาดเดาว่าตัวแปร `cnt` น่าจะเป็น 1 ทั้งคู่

ผลการรันแตกต่างจากการวิเคราะห์ตรงที่ค่าของ `cnt` นั้นออกเป็น 1 และ 2 โดยไม่เรียงลำดับกัน บางครั้ง `cnt` ขึ้นเป็น 1 ทั้งคู่

ตัวแปรที่เป็น shared variable ได้แก่ `cnt` (ซึ่งเป็น static variable) และตัวแปรที่เป็น private variable ได้แก่ `myid`

โค้ดในที่นี้อาจเกิด race condition ในตัวแปร `cnt` ซึ่งอาจทำให้ได้ค่าที่เหมือนกัน (ซึ่งไม่ควร เพราะเกิดการ increment ตามที่ต้องการควรจะให้ค่าที่ไม่ซ้ำ)

ข้อที่ 2

เป็นไปได้

- บรรทัดแรกเกิดจากการอ่านค่า i = 1 ที่ `create_thread`
- บรรทัดที่สองและสามเกิดจากการอ่านค่า i = 2 ที่ `create_thread`
- บรรทัดที่สี่เกิดจากการอ่านค่า i = 0 ที่ `join_thread`

ข้อที่ 3

ค่าจะไม่มีทางซ้ำกัน เพราะ pointer ที่ส่งให้กับแต่ละ thread นั้นถูกแยกกันโดยสมบูรณ์ (thread ที่ 0 รับ pointer ที่ค่าของ `i` ในขณะนั้นเป็น 0, thread ที่ 1 รับ pointer ที่ค่าของ `i` ในขณะนั้นเป็น 1, ...)

ผลลัพธ์เช่น Hello from thread 0, 1, 2, 3 // Hello from thread 0, 3, 1, 2 // Hello from thread 1, 2, 3, 0

ข้อที่ 4

แนวทางการแก้ปัญหาที่ `goodcnt` ใช้ คือการใช้ Semaphore เข้ามาช่วยกันในส่วนของ race condition ที่เกิดขึ้นเมื่อทั้งสอง threads พยายามอ่านและเขียนค่าของตัวแปร `cnt` (จะไม่กล่าวถึงหลักการของ Semaphore)

ฟังก์ชั่นแต่ละตัวทำงานดังนี้

- `sem_init` รับพารามิเตอร์ 3 ตัวได้แก่
  - pointer ไปยัง semaphore
  - ตัวแปร `pshared` ที่หากค่าไม่เป็น 0 แล้ว ตัว semaphores จะแชร์กันระหว่าง process ด้วย
  - ค่าเริ่มต้นของ semaphore
- `sem_wait` คือ V รอจนกว่า semaphore จะได้รับ signal
- `sem_post` คือ P เป็นการ post signal ไปยัง semaphore

ข้อที่ 5

ถูกต้อง โดยมีหลักการดังนี้

- `sbuf_insert` เรียก locks/semaphores ตามนี้โดยลำดับ
  - รอ semaphore ให้สัญญาณว่ามีช่องว่างใน buffer
  - ขอ lock ให้ buffer
  - ทำการเขียนค่าลง buffer
  - ปลด lock จาก buffer
  - ส่ง semaphore ให้สัญญาณว่ามี data ใน buffer
- `sbuf_consume` เรียก locks/semaphores ตามนี้โดยลำดับ
  - รอ semaphore ให้สัญญาณว่ามี data ใน buffer
  - ขอ lock ให้ buffer
  - ทำการดึงค่าจาก buffer
  - ปลด lock จาก buffer
  - ส่ง semaphore ให้สัญญาณว่ามีช่องว่างใน buffer

การทำแบบนี้จะทำให้มั่นใจได้ว่า (1) มี thread เดียวที่สามารถ consume/insert ค่าจาก buffer และ (2) จะมี thread เดียวที่ได้รับ permission ให้เป็น thread ที่พร้อมกระทำการต่อไปในการ consume/insert