Concurrency Part-I

Experiment #1

Experiment #1

แก้ไข for ทั้ง 2 จุด

```
for(i=0; i < 100000; i++)
```

```
1  // simple thread
2  using System;
3  using System.Threading;
4
5  namespace Lab_OS_Concurrency
6  {
7      class Program
8      {
9          static void TestThread1()
10         {
11             int i;
12             for (i = 0; i < 100; i++)
13                 Console.WriteLine("Thread# 1 i={0}", i);
14         }
15         static void TestThread2()
16         {
17             int i;
18             for (i = 0; i < 100; i++)
19                 Console.WriteLine("Thread# 2 i={0}", i);
20         }
21
22         static void Main(string[] args)
23         {
24             Thread th1 = new Thread(TestThread1);
25             Thread th2 = new Thread(TestThread2);
26             th1.Start();
27             th2.Start();
28         }
29     }
30 }
```

จากการสังเกตการสร้างและใช้งาน thread พบว่าผลการทำงานของโปรแกรมเป็นอย่างไร

(ในกรณีที่พบเจอความผิดปรกติของผลการทำงานของโปรแกรมให้ตั้งข้อสมมุติฐานว่าเกิดจากสาเหตุใด)

64010761 : เมื่อทำการ run program ดูหลายๆครั้ง พบว่าตัวเลขลำดับของ thread ไม่ได้เรียงตามลำดับ และในแต่ละครั้งที่ run program จะพบว่าลำดับ thread เรียงลำดับไม่เหมือนเดิม ทั้งนี้ขึ้นอยู่กับระบบปฏิบัติการณ์ และเครื่องคอมพิวเตอร์ด้วย

เหตุผลที่เป็นแบบนี้เพราะว่าคุณสมบัติการทำงานแบบ multithread ซึ่งจะมีเกณฑ์ที่จะจัดการลำดับการทำงาน การตัดสินว่า thread ไหนจะได้รับเวลา CPU ในการทำงานก่อน-หลัง จากการทำงานแบบ multithread การตัดสินใจเรื่องการทำงานของ thread ก็ไม่ได้รับการรับประกันว่าจะเรียงกันไปตามลำดับ โดย thread หนึ่งอาจจะทำงานเป็นระยะเวลาหนึ่งก่อนที่เธรดอื่นจะทำงานก็ได้

64010659 : แต่ละ Thread จะวนลูปที่มีค่า i ตั้งแต่ 0 ถึง 99999 และทำงานพร้อมกัน ซึ่งไม่สามารถคาดการณ์ลำดับที่แน่นอนได้เนื่องจากสอง Thread กำลัง Race Condition กันในการทำงาน

สมมติฐานที่เกี่ยวกับผลการทำงานที่ผิดปกติอาจเกิดจาก :

- Race Condition : เนื่องจากการเขียน-อ่านข้อมูล i ในลูป for แต่ละ Thread อาจทำให้เกิดการแย่งกันในการเขียนและอ่านค่า i ซึ่งอาจทำให้ค่า i ที่ถูกพิมพ์ใน Console ไม่ถูกต้องหรือสับสน

- Inconsistent Output : เนื่องจากการพิมพ์ข้อความใน Console มีการแชร์แหล่งทรัพยากรร่วมกัน อาจเกิดปัญหาที่ข้อความจะถูกพิมพ์ผสมกันระหว่าง Thread ทั้งสอง ทำให้ผลลัพธ์ที่แสดงผลบนหน้าจอไม่สมบูรณ์หรือไม่ถูกต้องตามลำดับที่คาดหวัง

64010845 : จากการทดลองรันโปรแกรมพบว่า ในการรันโปรแกรมแต่ละครั้งผลลัพธ์ไม่เหมือนกัน กล่าวคือการนับเลขในแต่ละ Thread ไม่เหมือนกันในแต่ละรอบการรัน โดยสาเหตุเกิดจาก Thread ได้รับงานมามากเกินไปทำให้ทำงานไม่เสร็จทันเวลาที่กำหนด จึงต้องไปทำงานอื่นก่อนจะกลับมาทำงานเดิม จึงเห็นการทำงานที่สลับระหว่าง Thread

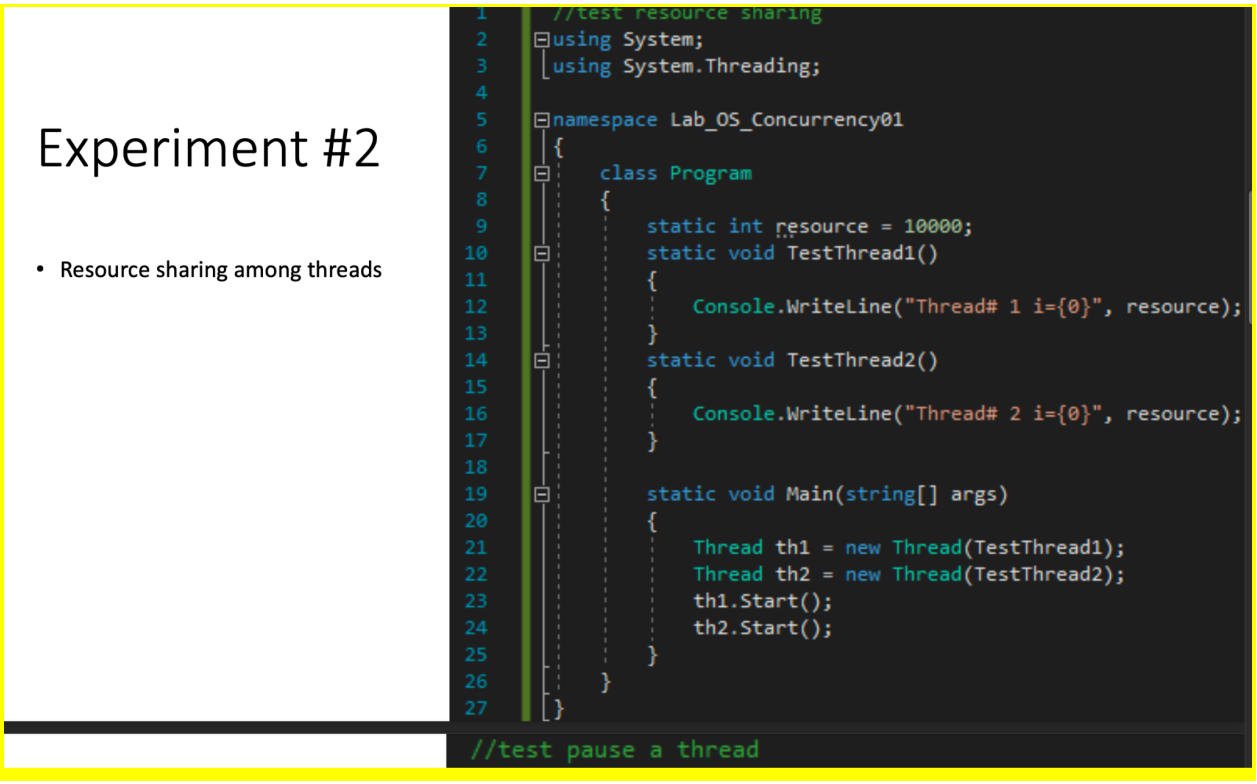
64010516 :

จากการทดลองรันโปรแกรมพบว่า ในการรันโปรแกรมแต่ล่ะรอบ จะแสดงผลลัพธ์ที่ไม่เหมือนกันในแต่ล่ะรอบ และเรียงลำดับไม่เหมือนกัน และขึ้นอยู่กับ OS ที่ใช้งานและ Hardware ของเครื่องคอมพิวเตอร์ด้วย

สาเหตุที่เป็นอย่างงั้นเนื่องจาก คอมพิวเตอร์เราทำงานแบบ Multithread ซึ่งจะมีกฎเกณฑ์ที่จะมีการจัดลำดับการทำงานว่าควรทำ task ไหนก่อน และจากการตัดสินใจของการทำงานของ Multithread เราจะไม่ทราบเลยว่า task ไหนจะถูกทำงานก่อนและหลัง ทำให้การรันโปรแกรมแต่ล่ะรอบได้ผลลัพธ์ไม่เหมือนกัน

64010683 : จากการทดลองรันโปรแกรมพบว่า ในการรันโปรแกรมแต่ละครั้งจะได้ผลลัพธ์ไม่เหมือนกัน เพราะ Thread ได้รับงานมามากเกินไป ทำให้เวลาไม่เพียงพอ จึงต้องสลับไปทำงานของ Thread อื่นก่อนเป็นระยะเวลาหนึ่งๆ ก่อนจะกลับทำงานของ Thread เดิม สลับกันไปเรื่อย ๆ จนงานที่แต่ละ Thread ได้รับถูกทำจนหมด

Experiment #2



จากการสังเกตการสร้างและใช้งาน thread พบว่าผลการทำงานของโปรแกรมเป็นอย่างไร

(ในกรณีที่พบเจอความผิดปรกติของผลการทำงานของโปรแกรมให้ตั้งข้อสมมุติฐานว่าเกิดจากสาเหตุใด)

64010761 : ผลลัพธ์ที่ได้จากการ run program หลายๆครั้งพบว่า บางครั้ง thread 1 แสดงก่อน thread 2 บางครั้ง thread 2 แสดงผลก่อน thread 1 เนื่องมาจาก Threads ที่ถูกสร้างขึ้นอาจถูกจัดตารางการทำงานโดยระบบปฏิบัติการเพื่อแบ่งเวลา CPU ซึ่งอาจทำให้ Threads ไม่ได้รับการทำงานที่มีลำดับตามที่คาดหวัง

64010659 : มีการแชร์ข้อมูล resource ระหว่าง Thread สองตัว (th1 และ th2) โดยทั้งสอง Thread จะอ่านค่าของ resource และพิมพ์ใน Console แต่ละ Thread อยู่ดังนี้ :

- TestThread1 ทำงานใน th1: พิมพ์ค่า resource ที่อ่านได้ใน Console
- TestThread2 ทำงานใน th2: พิมพ์ค่า resource ที่อ่านได้ใน Console

ไม่มีการแชร์ข้อมูลแบบ race condition เนื่องจากไม่มีการเขียนหรือเปลี่ยนแปลงค่าของ resource ในที่นี้ มันเป็นการอ่านค่า resource อย่างเดียว และไม่มีการแชร์ทรัพยากรร่วมกันที่อาจเกิดขึ้นในเวลาเดียวกัน

ผลการทำงานของโปรแกรม จะพบว่าผลลัพธ์ที่แสดงผลใน Console ขึ้นอยู่กับลำดับของ Thread ที่ทำงาน แต่ผลการทำงานที่ถูกต้อง คือไม่มีการ Race Condition ในการอ่านค่า resource จาก Thread ทั้งสอง เนื่องจากไม่มีการเปลี่ยนแปลงค่าของ resource ในการทำงานของทั้งสอง Thread

สมมติฐานที่อาจเกิดขึ้นเมื่อพบความผิดปกติ :

- Race Condition : ถึงแม้ในโปรแกรมนี้ไม่มีการเปลี่ยนแปลงค่าของ resource แต่ในกรณีที่คุณมีการเขียนข้อมูลเข้าสู่ resource จากอื่นๆ Thread ที่ไม่ได้แสดงในโค้ด อาจเกิดปัญหาที่ทำให้ผลการทำงานของ Thread แต่ละตัวเปลี่ยนแปลง

- Inconsistent Output : เนื่องจากการพิมพ์ข้อความใน Console มีการแชร์แหล่งทรัพยากรร่วมกัน อาจเกิดปัญหาที่ข้อความจะถูกพิมพ์ผสมกันระหว่าง Thread ทั้งสอง ทำให้ผลลัพธ์ที่แสดงผลบนหน้าจอไม่สมบูรณ์หรือไม่ถูกต้องตามลำดับ

64010845 : จากการทดลองรันโปรแกรมหลายๆ ครั้งพบว่าการทำงานไม่ได้เรียงตามลำดับ โดยบางครั้ง Thread 1 แสดงก่อน บางครั้ง Thread 2 แสดงก่อนเป็นช่วงๆ เนื่องจาก Thread ทำงานเป็นอิสระแยกออกจากกันทำให้ในช่วงเวลาที่รันโปรแกรม Thread ไหนว่างก็จะทำงานก่อน

64010516 : จากการทดลองพบว่า ทั้งสอง thread ที่ถูกสร้างมานั้นมีการแชร์ resource ร่วมกัน เนื่องจาก resource มีการประกาศตัวแปรแบบ global ที่เข้าถึงได้จาก method ใดๆ ใน class ไม่มีการแย่งข้อมูลใช้ก่อนหรือหลังสร้าง thread 1 และ 2 แต่การแสดงผล thread 1 หรือ 2 จะแสดงผลลำดับก่อนหลังบ้าง ทั้งนี้ขึ้นอยู่กับการประมวลผลของ CPU

64010683 : จากการทดลองรันโปรแกรมหลาย ๆ ครั้ง พบว่าการทำงานไม่ได้เรียงตามลำดับเหมือนกันทุกครั้ง บางครั้ง Thread 1 จะทำงานก่อน และบางครั้ง Thread 2 จะทำงานก่อน เนื่องจากในแต่ละช่วงเวลา Thread ที่ว่างจะไม่เหมือนกัน โดย Thread ที่ว่างจะทำงานก่อน

Experiment #3

จากการสังเกตการสร้างและใช้งาน thread พบว่าผลการทำงานของโปรแกรมเป็นอย่างไร

(ในกรณีที่พบเจอความผิดปรกติของผลการทำงานของโปรแกรมให้ตั้งข้อสมมุติฐานว่าเกิดจากสาเหตุใด)

64010761 : ข้อนี้ output ในเครื่องของผู้ทำการทดลองมีผลลัพธ์ออกมาได้เหมือนเดิมทุกครั้ง คือ resource=55555 แต่สิ่งที่แตกต่างคือ Thread.Sleep(10) กับ Thread.Sleep(100) ซึ่งเป็นคำสั่งที่ทำให้ ระยะเวลาการทำงานของโปรแกรมนานขึ้น

สามารถเกิด Race Condition ขึ้นเมื่อมีการแข่งขันกันในการเข้าถึงและแก้ไขค่าของ resource จากหลาย Thread ดังนั้นผลลัพธ์ที่แสดงออกมาอาจจะเป็นได้ทั้ง resource=10000 หรือ resource=55555 ขึ้นอยู่กับเวลาที่ Thread ที่ 1 ทำงานและเปลี่ยนค่า resource ในขณะที่ Main Thread อยู่ในสถานะหยุดรอ Thread.Sleep()

64010659 : มีการแชร์ข้อมูล resource ระหว่าง Main Thread และ Thread th1 โดยที่ th1 จะเปลี่ยนค่าของ resource เป็น 55555 และ Main Thread จะพิมพ์ค่า resource ออกทาง Console

โดยการเพิ่ม Thread.Sleep(100); ใน Main Thread หลังจากเรียก th1.Start(); นั้นจะทำให้ Main Thread หยุดรองรับการทำงานของ th1 ลำดับการทำงานคือ :

- Main Thread เรียก th1.Start(); เพื่อเริ่มการทำงานของ th1
- Main Thread หยุดรองรับการทำงานเป็นเวลา 100 มิลลิวินาที (Thread.Sleep(100);)
- th1 ทำงานและเปลี่ยนค่าของ resource เป็น 55555
- Main Thread ทำงานต่อจากการหยุดรองรับ และพิมพ์ค่า resource ออกทาง Console

เนื่องจาก Thread.Sleep(100); ทำให้ Main Thread หยุดรองรับเป็นเวลา 100 มิลลิวินาที ผลที่แสดงผลใน Console จะแสดงค่า resource ที่เป็นค่าเริ่มต้น 10000 เพราะ th1 ที่ทำงานเปลี่ยนค่า resource หลังจาก Thread.Sleep(100); และค่านี้ไม่มีการปรับปรุงใน Main Thread ก่อนนั่น ดังนั้นผลการทำงานที่แสดงผลใน Console จะเป็น :

Resource = 10000

สมมติฐานที่เกี่ยวกับผลการทำงานที่ผิดปกติอาจเกิดจาก :

- Race Condition : ในกรณีที่ไม่ได้ใช้ Thread.Sleep(100); ใน Main Thread อาจเกิดการ Race Condition ในการอ่าน-เขียนค่า resource ระหว่าง Main Thread และ th1 ทำให้ค่า resource ที่ถูกพิมพ์ใน Console ไม่ถูกต้องหรือสับสน

- Inconsistent Output : เนื่องจากการพิมพ์ข้อความใน Console มีการแชร์แหล่งทรัพยากรร่วมกัน ในกรณีที่ Main Thread และ th1 พยายามที่จะพิมพ์ค่า resource ลงใน Console พร้อมกัน อาจเกิดปัญหาที่ข้อความจะถูกพิมพ์ผสมกันทำให้ผลลัพธ์ที่แสดงผลบนหน้าจอไม่สมบูรณ์หรือไม่ถูกต้องตามลำดับที่คาดหวัง

64010845 : จากการทดลองรันโปรแกรมหลายๆ ครั้งพบว่าผลลัพธ์คือ resource=55555 เหมือนเดิมทุกครั้งไม่ว่าจะเป็น thread.sleep(10) หรือ thread.sleep(100) โดยคำสั่ง Thread.Sleep() เป็นคำสั่งที่ให้เวลาในการทำงานของ Thread ไปทำงานที่ได้รับตามเวลาที่กำหนด แต่หากนำ thread.sleep() ออกผลลัพธ์จะเป็น resource=10000 เนื่องจากการสั่งให้ thread ทำงานนั้นช้ากว่าการอ่านคำสั่ง Console.WriteLine(“resource={0}”, resource); ทำให้ tread ยังไม่ได้เข้าทำงาน ค่าจึงไม่มีการเปลี่ยนแปลงจาก 10000 เป็น 55555

64010516 : จากการทดลองรันพบว่า ผลลัพธ์ที่ได้จะเป็น resource=55555 เพราะ resource นั้นแชร์กันระหว่าง Main Process และ Child Process ที่เรียกจาก th1.Start(); ใน Process Main และ sleep Process Main ตามระยะเวลาที่รอ ให้ Child Process อัพเดทค่า resource ก่อน Process Main ทำงานต่อ นั้นคือการแสดงผล แต่ทั้งนี้ก็ขึ้นอยู่กับ clock rate ของ CPU ด้วย หาก CPU ความเร็วช้า อาจจะต้องใส่ระยะเวลาในการ sleep นานขึ้น เพื่อไม่ให้ Process Main แย่งทำงานก่อน Child Process ทำงาน ซึ่งสิ่งนี้เรียกว่า Race Condition

64010683 : จากการทดลองรันโปรแกรมหลาย ๆ ครั้ง พบว่าการทำงานจะได้ผลลัพธ์เหมือนกันทุกครั้ง เนื่องจากมีการกำหนดเวลาที่แต่ละ Thread ได้รับอย่างชัดเจนจากคำสั่ง Thread.Sleep(100)

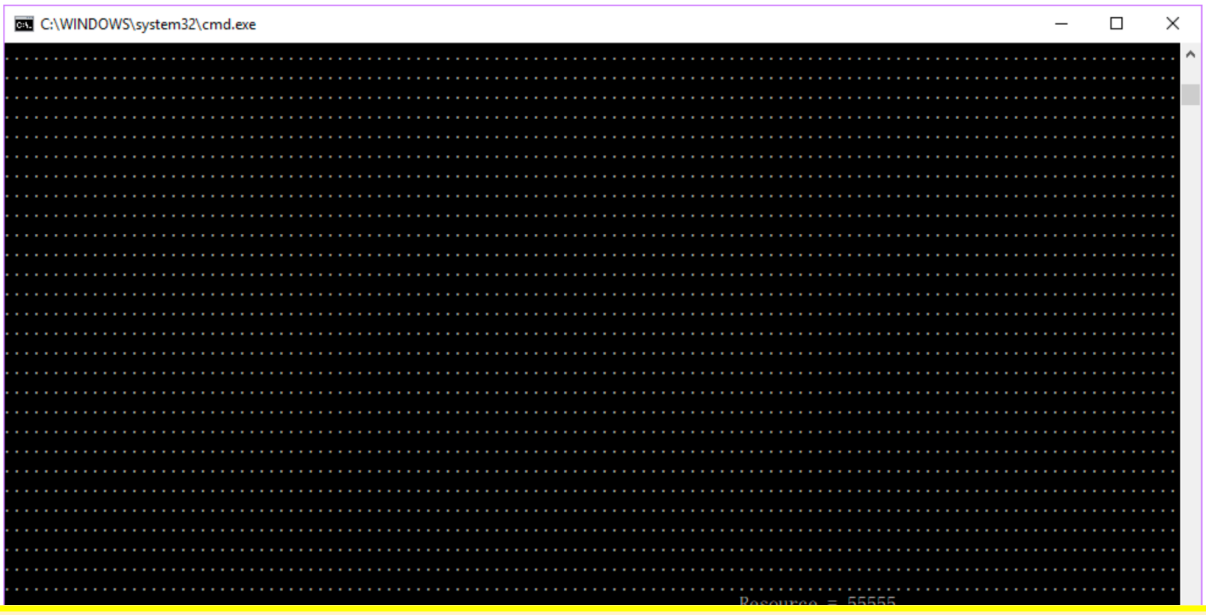
Experiment #3.1

Experiment #3.1

- Pause a thread #2

```
1   //test pause #2
2   using System;
3   using System.Threading;
4
5   namespace Lab_OS_Concurrency01
6   {
7       class Program
8       {
9           static int resource = 10000;
10          static void TestThread1()
11          {
12              int i;
13              for (i = 0; i < 45555; i++)
14              {
15                  resource++;
16                  Console.Write(".");
17              }
18          }
19
20          static void Main(string[] args)
21          {
22              Thread th1 = new Thread(TestThread1);
23              th1.Start();
24              Thread.Sleep(10);
25              Console.WriteLine("Resource = {0}", resource);
26          }
27      }
```

Experiment #3.1 desired result



จากการสังเกตการสร้างและใช้งาน thread พบว่าผลการทำงานของโปรแกรมเป็นอย่างไร

(ในกรณีที่พบเจอความผิดปรกติของผลการทำงานของโปรแกรมให้ตั้งข้อสมมุติฐานว่าเกิดจากสาเหตุใด)

64010761 : จากการทดลอง run หลายๆครั้งพบว่าผลลัพธ์ไม่ได้เหมือนกับของอาจารย์ เพราะว่าค่า resource ไม่ได้ถูกแสดงตอนจบและค่า resource ที่ออกมาในแต่ละครั้งก็มีค่าแตกต่างกัน สาเหตุอาจเกิดจากมีการเข้าถึงและแก้ไขค่า resource จากหลาย Thread พร้อมกัน อาจเกิดเหตุการแข่งขันที่ทำให้ค่า resource ไม่เป็นไปตามลำดับที่คาดหวัง

64010659 : มีการแชร์ข้อมูล resource ระหว่าง Main Thread และ Thread th1 โดยที่ th1 จะเพิ่มค่าของ resource โดยวนลูปจำนวน 45555 ครั้ง และ Main Thread จะพิมพ์ค่า resource ออกทาง Console

โดยการใช้ Thread.Sleep(10); ใน Main Thread หลังจากเรียก th1.Start(); นั้นจะทำให้ Main Thread หยุดรองรับการทำงานของ th1 ลำดับการทำงานคือ :

- Main Thread เรียก th1.Start(); เพื่อเริ่มการทำงานของ th1
- Main Thread หยุดรองรับการทำงานเป็นเวลา 10 มิลลิวินาที (Thread.Sleep(10);)
- th1 ทำงานและเพิ่มค่าของ resource โดยวนลูปจำนวน 45555 ครั้ง
- Main Thread ทำงานต่อจากการหยุดรองรับ และพิมพ์ค่า resource ออกทาง Console

เนื่องจากเราใช้ Thread.Sleep(10); ใน Main Thread อยู่ เรามีการสร้างการชะลอให้ th1 ทำงานและเพิ่มค่า resource ก่อน Main Thread ทำงานต่อ ดังนั้นผลการทำงานที่แสดงผลใน Console จะเป็น :

.....Resource = 55555

สมมติฐานที่เกี่ยวกับผลการทำงานที่ผิดปกติอาจเกิดจาก :

- Race Condition : เนื่องจากการเพิ่มค่าของ resource ใน TestThread1 และการอ่านค่าของ resource ใน Main Thread ที่มาก่อน Thread.Sleep(10); อาจเกิดขึ้นพร้อมกัน ทำให้เกิด Race Condition ในการอ่าน-เขียนค่า resource ทำให้ค่า resource ที่ถูกพิมพ์ใน Console ไม่ถูกต้องหรือสับสน

- Inconsistent Output : เนื่องจากการพิมพ์ข้อความใน Console มีการแชร์แหล่งทรัพยากรร่วมกัน ในกรณีที่ Main Thread และ th1 พยายามที่จะพิมพ์ค่า resource และจำนวนจุด (.) ลงใน Console พร้อมกัน อาจเกิดปัญหาที่ข้อความจะถูกพิมพ์ผสมกันทำให้ผลลัพธ์ที่แสดงผลบนหน้าจอไม่สมบูรณ์หรือไม่ถูกต้องตามลำดับที่คาดหวัง

64010845 : จากการทดลองรันโปรแกรมพบว่า Code จากอาจารย์ในตอนแรก Thread.Sleep(10) จะได้ผลลัพธ์ไม่เหมือนอาจาย์เนื่องจากช่วงเวลาในการทำงานน้อยเกินไปจนออกไปทำคำสั่ง Console.WriteLine() แล้วจึงกลับมาทำงานใน Thread ต่อโดยเราจะต้องเพิ่ม Thread.Sleep() แต่ทั้งนี้ก็ขึ้นอยู่กับความแรงของคอมและการทำงานของ Thread ในช่วงเวลานั้นๆ ด้วย

64010516 : จากการทดลองพบว่า เมื่อใช้ Thread.Sleep(10); นั้นจะได้ผลลัพธ์ไม่เหมือนของอาจารย์ ได้ Resource = 10001 แทรกระหว่างพิมพ์จุด นั้นหมายความว่า เรา Sleep Main Process ระยะเวลาสั้นเกินไปก่อนที่ Child Process จะทำงานเสร็จก่อน จากนั้นก็ลองเปลี่ยนเป็น Thread.Sleep(100); จึงได้ผลลัพธ์ตามอาจารย์ ซึ่งเหตุการณ์ได้เกิด race condition ขึ้น

64010683 : จากการทดลองรันโปรแกรมหลาย ๆ ครั้ง พบว่าการทำงานจะได้ผลลัพธ์ไม่เหมือนกับของอาจารย์ เนื่องจากจำนวนของคำสั่งที่ทำงานได้ในช่วงเวลาที่กำหนดน้อยกว่าของอาจารย์ คาดว่าเกิดจากขีดจำกัดของทรัพยากรในคอมพิวเตอร์แต่ละเครื่องไม่เหมือนกัน

Experiment #3.2

Experiment #3.2

- Join thread

```
//test pause #2
using System;
using System.Threading;

namespace Lab_OS_Concurrency01
{
    class Program
    {
        static int resource = 10000;
        static void TestThread1()
        {
            int i;
            for (i = 0; i < 45555; i++)
            {
                resource++;
                Console.Write(".");
            }
        }

        static void Main(string[] args)
        {
            Thread th1 = new Thread(TestThread1);
            th1.Start();
            //Thread.Sleep(10);
            th1.Join();
            Console.WriteLine("Resource = {0}", resource);
        }
    }
}
```

จากการสังเกตการสร้างและใช้งาน thread พบว่าผลการทำงานของโปรแกรมเป็นอย่างไร

(ในกรณีที่พบเจอความผิดปรกติของผลการทำงานของโปรแกรมให้ตั้งข้อสมมุติฐานว่าเกิดจากสาเหตุใด)

64010761 : จากการทดลอง run program พบว่า resource แสดงในตอนสุดท้ายและมีค่า reource=55555 ตามที่คาดหวังไว้ เนื่องมาจาก ใช้ th1.Join() เพื่อรอให้ Thread th1 ทำงานเสร็จก่อนที่จะทำงานต่อใน Main Thread โดยที่ th1.Join() จะบังคับให้ Main Thread รอจนกว่า th1 จะเสร็จสิ้นการทำงาน ก่อนที่จะแสดงผลลัพธ์ในบรรทัด Console.WriteLine("resource={0}", resource);

64010659 : มีการแชร์ข้อมูล resource ระหว่าง Main Thread และ Thread th1 โดยที่ th1 จะเพิ่มค่าของ resource โดยวนลูปจำนวน 45555 ครั้ง และ Main Thread จะพิมพ์ค่า resource ออกทาง Console

โดยการใช้ th1.Join(); ใน Main Thread เป็นการรอให้ Thread th1 ทำงานเสร็จสิ้นก่อนที่ Main Thread จะทำงานต่อ ดังนั้นลำดับการทำงานคือ :

- Main Thread เรียก th1.Start(); เพื่อเริ่มการทำงานของ th1
- th1 ทำงานและเพิ่มค่าของ resource โดยวนลูปจำนวน 45555 ครั้ง
- Main Thread รอให้ th1 ทำงานเสร็จสิ้น (th1.Join();)
- th1 เสร็จสิ้นการทำงาน
- Main Thread ทำงานต่อ และพิมพ์ค่า resource ออกทาง Console

เนื่องจาก th1 ถูก Join ใน Main Thread ทำให้ th1 ต้องเสร็จสิ้นการทำงานก่อน Main Thread ทำงานต่อ ดังนั้นผลการทำงานที่แสดงผลใน Console จะเป็น :

.....Resource 55555

สมมติฐานที่เกี่ยวกับผลการทำงานที่ผิดปกติอาจเกิดจาก :

- Race Condition: หากไม่ได้ใช้ th1.Join(); ใน Main Thread อาจเกิดการ Race Condition ในการอ่าน-เขียนค่า resource ระหว่าง Main Thread และ th1 ทำให้ค่า resource ที่ถูกพิมพ์ใน Console ไม่ถูกต้องหรือสับสน

- Inconsistent Output : เนื่องจากการพิมพ์ข้อความใน Console มีการแชร์แหล่งทรัพยากรร่วมกัน ในกรณีที่ Main Thread และ th1 พยายามที่จะพิมพ์ค่า resource และจำนวนจุด (.) ลงใน Console พร้อมกัน อาจเกิดปัญหาที่ข้อความจะถูกพิมพ์ผสมกันทำให้ผลลัพธ์ที่แสดงผลบนหน้าจอไม่สมบูรณ์หรือไม่ถูกต้องตามลำดับที่คาดหวัง

64010845 : จากการทดลองรันโปรแกรมคำสั่ง th1.Join() จะเป็นคำสั่งที่บอกให้ Thread นั้นทำงานให้เสร็จก่อนจะออกไปทำงานอื่นต่อ

64010516 : จากการทดลองพบว่าได้ผลลัพธ์เหมือน การทดลอง 3.1 ได้โดยไม่ต้อง Sleep Main Process เลย เนื่องจาก th1.Join() จะ join process ของ Child Process ให้รอ Child Process ทำงานเสร็จสิ้นก่อนที่จะทำงาน Main Process ต่อ ซึ่งจะช่วยทำให้เวลารัน Program ในเครื่องที่มีความเร็วการประมวลผลต่างกันช่วยให้ได้ผลลัพธ์ที่เหมือนกัน

64010683 : จากการทดลองรันโปรแกรมพบว่า Thread จะทำคำสั่งที่ได้รับจนเสร็จก่อนไปทำงานคำสั่งอื่น เนื่องจากคำสั่ง th1.Join();