Synchronization Part III

รายชื่อสมาชิก

64010516	ปัณณวิชญ์ วชิรเศรษฐหิรัญ
64010659	ภัทราภรณ์ จันเดชา
64010683	ภูมิรพี สินคีรี
64010761	วรพล รังษี
64010845	ศิรสิทธิ์ เทียนเจริญชัย

Version 0 (Original)

```
using System;
using System.Threading;

namespace OS_Problem_02
{
    class Thread_safe_buffer
    {
        static int[] TSBuffer = new int[10];
        static int Front = 0;
        static int Back = 0;
        static int Count = 0;

        static void EnQueue(int eq)
        {
            TSBuffer[Back] = eq;
            Back++;
            Back %= 10;
            Count += 1;
        }

        static int DeQueue()
        {
            int x = 0;
            x = TSBuffer[Front];
            Front++;
            Front %= 10;
            Count -= 1;
            return x;
        }
```

```
static void th01()
{
    int i;

    for (i = 1; i < 51; i++)
    {
        EnQueue(i);
        Thread.Sleep(5);
    }
}

static void th011()
{
    int i;

    for (i = 100; i < 151; i++)
    {
        EnQueue(i);
        Thread.Sleep(5);
    }
}


static void th02(object t)
{
    int i;
    int j;

    for (i=0; i< 60; i++)
    {
        j = DeQueue();
        Console.WriteLine("j={0}, thread:{1}", j, t);
        Thread.Sleep(100);
    }
}
static void Main(string[] args)
{
    Thread t1 = new Thread(th01);
    //Thread t11 = new Thread(th011);
    Thread t2 = new Thread(th02);
    //Thread t21 = new Thread(th02);
    //Thread t22 = new Thread(th02);

    t1.Start();
```

```
            //t11.Start();
            t2.Start(1);
            //t21.Start(2);
            //t22.Start(3);
        }
    }
}
```

ผลลัพธ์ :

โปรแกรมโจทย์ทำงานไม่ได้ตามต้องการ เพราะเหตุใด

จากผลลัพธ์จะเห็นได้ว่า Enqueue และ Dequeue ทำงานไม่ Synchronize กันโดยที่ Enqueue มีการทำงานไวกว่า เพราะคำสั่ง Thread.sleep() และเมื่อ Enqueue ทำงานเสร็จก่อนทำให้ Dequeue loop Dequeue ออกมาโดยที่ยังเป็นค่าเดิมใน Array

จุดใดบ้างที่ต้องแก้ไข + เหตุผลรองรับ

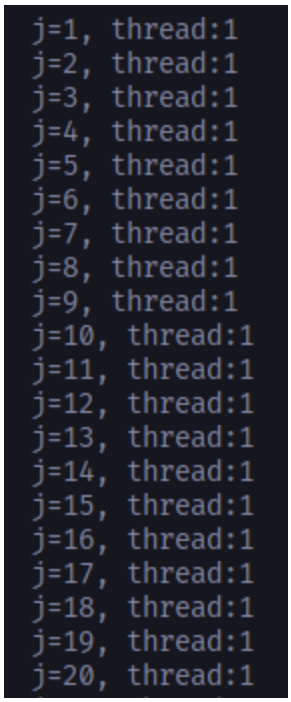
ต้องทำให้ Enqueue และ Dequeue ทำงาน Synchronize กันเพื่อให้ Enqueue และ Dequeue ทำงานไปพร้อมๆ กันอย่างเป็นลำดับเพื่อไม่ให้เกิดการ loop Dequeue ค่าซ้ำ

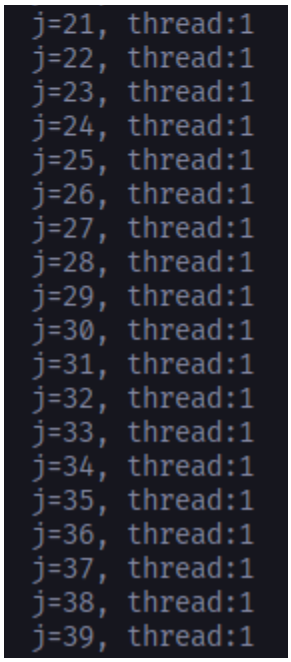
Version 1

```
using System;
using System.Threading;

namespace OS_Problem_02
{
    class Thread_safe_buffer
    {
        static int[] TSBuffer = new int[10];
        static int Front = 0;
        static int Back = 0;
        static int Count = 0;
        static readonly object lockObject = new object();
        static Semaphore semaphore = new Semaphore(1, 10);
        static void EnQueue(int eq)
        {
            semaphore.WaitOne();
            TSBuffer[Back] = eq;
            Back++;
            Back %= 10;
            Count += 1;
        }

        static int DeQueue()
        {
            int x = 0;
            x = TSBuffer[Front];
            Front++;
            Front %= 10;
            Count -= 1;
            semaphore.Release();
            return x;
        }

        static void th01()
        {
            int i;

            for (i = 1; i < 51; i++)
            {
                EnQueue(i);
                Thread.Sleep(5);
            }
        }
```

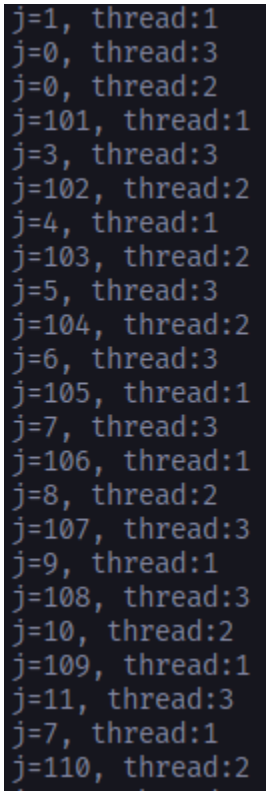
```
        static void th011()
        {
            int i;

            for (i = 100; i < 151; i++)
            {
                EnQueue(i);
                Thread.Sleep(5);
            }
        }


        static void th02(object t)
        {
            int i;
            int j;

            for (i = 0; i < 60; i++)
            {
                if (Count != 0)
                {
                    j = DeQueue();
                    Console.WriteLine("j={0}, thread:{1}", j, t);
                }
                Thread.Sleep(100);
            }
        }
        static void Main(string[] args)
        {
            Thread t1 = new Thread(th01);
            //Thread t11 = new Thread(th011);
            Thread t2 = new Thread(th02);
            //Thread t21 = new Thread(th02);
            //Thread t22 = new Thread(th02);

            t1.Start();
            //t11.Start();
            t2.Start(1);
            //t21.Start(2);
            //t22.Start(3);
        }
    }
}
```

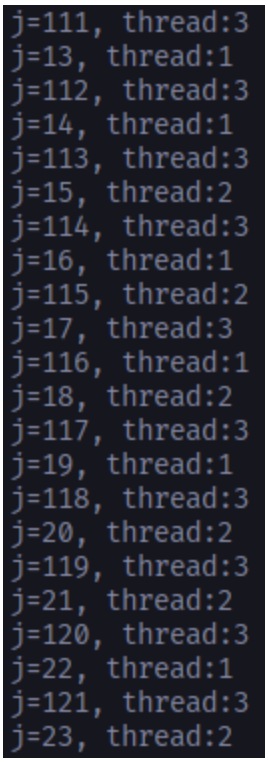
ผลลัพธ์ :





ก่อนเปิดคอมเม้น





หลังเปิดคอมเม้น

โปรแกรมโจทย์ทำงานไม่ได้ตามต้องการ เพราะเหตุใด :

จากการรันโปรแกรมก่อนเปิดคอมเม้นพบว่า ผลลัพธ์ของโปรแกรมถูกแล้วแต่มีการใช้เวลาที่มากกว่า Version ก่อนเพียงเล็กน้อย เนื่องจากการเพิ่มโปรแกรมใหม่เข้าไปทำให้การทำงานของ Enqueue และ Dequeue ทำงาน Synchronize กันแล้วโดย Semaphore(1, 10) ที่กำหนดให้มีการ Enqueue ได้เต็มที่แค่ 10 ตัวเพื่อการรอให้ Dequeque ทำงานโดยที่ Dequeue ทำงาน 1 ครั้งก็จะทำให้ Enqueue 1 ครั้ง หลังจากเพิ่มไปก่อนหน้าแล้ว 10 ตัวซึ่งตัวที่เพิ่มหลังจากนั้นจะเป็นการเขียนทับตัวเดิมใน array แต่ว่าเมื่อเปิดเม้นแล้วรันโปรแกรมอีกรอบปรากฏว่าการทำงานแบบหลาย Thread เกิดการ Race Condition ขึ้น ตาม Output ที่เห็นคือ Thread หลาย Thread DeQueue พร้อมกันทำให้ แสดงผลเลขซ้ำกันใน 2-3 Threads ในบางครั้งในการรันแต่ล่ะรอบ

จุดใดบ้างที่ต้องแก้ไข + เหตุผลรองรับ :

- ต้องทำการ lock ในส่วนการเข้าถึง ตัวแปร global เมื่อมีการเรียกใช้งานหลาย Thread และเปลี่ยนจากการใช้ Semaphore เป็น Conditional Variable เพื่อใช้การแก้ปัญหาที่ง่ายขึ้นและป้องกันการสับสน

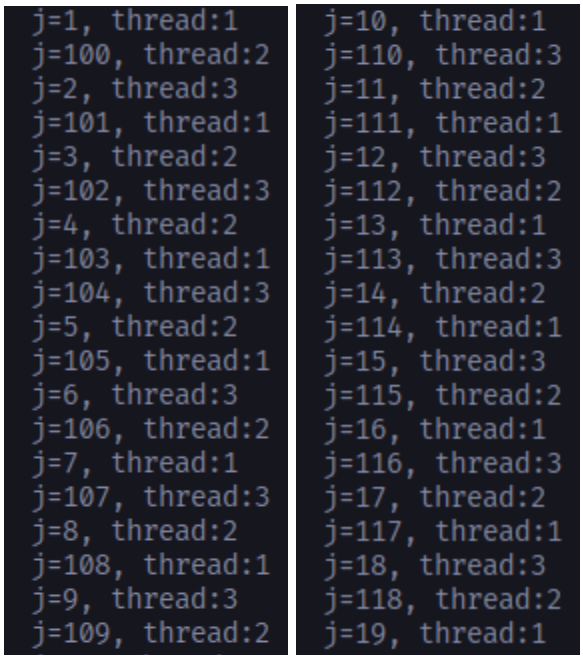
Version 2 (Final version)

```
using System;
using System.Threading;

namespace OS_Problem_02
{
    class ThreadSafeBuffer
    {
        static int[] TSBuffer = new int[10];
        static int Front = 0;
        static int Back = 0;
        static int Count = 0;
        static bool exitFlag = false; // Flag to signal threads to
exit
        static readonly object lockObject = new object();

        static void EnQueue(int eq)
        {
            lock (lockObject)
            {
                while (Count >= 10)
                    Monitor.Wait(lockObject);
                TSBuffer[Back] = eq;
                Back = (Back + 1) % 10;
                Count++;
                Monitor.Pulse(lockObject);
            }
        }

        static int DeQueue()
        {
            int x = 0;
            lock (lockObject)
            {
                while (Count == 0 && !exitFlag) // Check exitFlag
                    Monitor.Wait(lockObject);
                if (Count > 0)
                {
                    x = TSBuffer[Front];
                    Front = (Front + 1) % 10;
                    Count--;
                    Monitor.Pulse(lockObject);
                }
            }
```

```
    return x;
}

static void th01()
{
    int i;

    for (i = 1; i < 51; i++)
    {
        EnQueue(i);
        Thread.Sleep(5);
    }
}

static void th011()
{
    int i;

    for (i = 100; i < 151; i++)
    {
        EnQueue(i);
        Thread.Sleep(5);
    }
}

static void th02(object t)
{
    int i;
    int j;

    for (i = 0; i < 60; i++)
    {
        j = DeQueue();
        if (j == 0)
        {
            Console.WriteLine("Exiting Thread {0}", t);
            break;
        }
        Console.WriteLine("j={0}, thread:{1}", j, t);
        Thread.Sleep(100);
    }
}

static void Main(string[] args)
{
```

```
            Thread t1 = new Thread(th01);
            Thread t11 = new Thread(th011);
            Thread t2 = new Thread(th02);
            Thread t21 = new Thread(th02);
            Thread t22 = new Thread(th02);

            t1.Start();
            t11.Start();
            t2.Start(1);
            t21.Start(2);
            t22.Start(3);

            t1.Join();
            t11.Join();

            lock (lockObject)
            {
                exitFlag = true;
                Monitor.PulseAll(lockObject);
            }

            t2.Join();
            t21.Join();
            t22.Join();
        }
    }
}
```

ผลลัพธ์ :



```
j=1, thread:1
j=100, thread:2
j=2, thread:3
j=101, thread:1
j=3, thread:2
j=102, thread:3
j=4, thread:2
j=103, thread:1
j=104, thread:3
j=5, thread:2
j=105, thread:1
j=6, thread:3
j=106, thread:2
j=7, thread:1
j=107, thread:3
j=8, thread:2
j=108, thread:1
j=9, thread:3
j=109, thread:2
```

```
j=10, thread:1
j=110, thread:3
j=11, thread:2
j=111, thread:1
j=12, thread:3
j=112, thread:2
j=13, thread:1
j=113, thread:3
j=14, thread:2
j=114, thread:1
j=15, thread:3
j=115, thread:2
j=16, thread:1
j=116, thread:3
j=17, thread:2
j=117, thread:1
j=18, thread:3
j=118, thread:2
j=19, thread:1
```

สิ่งที่แก้ไขจาก Version ก่อนหน้า

- ได้ทำการ lock ในส่วนที่เข้าถึงตัวแปร global variable ไว้ และ มีการดักในกรณีที่ DeQueue เมื่อ Count = 0 และ EnQueue เมื่อ Count = 10 (Queue เต็ม) ให้ Task นั้นรอไว้ก่อนโดยใช้ lockObject เป็น Conditional Variable แต่ถ้าโปรแกรมสามารถ EnQueue หรือ DeQueue ได้ปกติให้สั่งคำสั่ง Pulse ให้บอก Thread นั้น ๆ ว่าให้สามารถทำงานต่อ ๆ ได้ และเมื่อ DeQueue จบการทำงานแล้ว ก็ขะใช้ตัวแปร exitFlag เป็นสัญญาณบอกว่าให้ Thread นั้นออกจากโปรแกรม (ทำงานเสร็จสิ้น) ไม่เช่นนั้น โปรแกรมจะค้างและไม่จบการทำงาน

สรุปผล

Version	การเปลี่ยนแปลง	ผลลัพธ์
0	โปรแกรมเริ่มต้นของอาจารย์	Enqueue และ Dequeue ไม่ Synchronize กัน
1	เพิ่ม Semaphore เพื่อให้มีการรอกันของ Enqueue และ Dequeue	Enqueue และ Dequeue Synchronize กัน แต่เมื่อมีการเปิดเม้นเพื่อทำงานเป็น Muti Thread เกิดปัญหา race condition
2	เปลี่ยนจาก Semaphore เป็น Monitor.Pulse	Enqueue และ Dequeue Synchronize กันและการทำงาแบบ Multi Thread ไม่เกิด race Condition