Synchronization

เสนอ
ผศ.วัจนพงศ์ เกษมศิริ

จัดทำโดย

64010462	บุริศ เสรีวัตตนะ
64010850	ศิวกร สุริยะ
64010926	สุพิชญา พรหมปาลิต
64010936	สุรางคนางค์ เกตุยั่งยืนวงศ์
64010994	อรัชฌา ปิ่นประยูร
64011131	ธนกฤต ธรรมภาณพินิจ

รายงานนี้เป็นส่วนหนึ่งของรายวิชา 01076011
OPERATING SYSTEMS
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

VERSION เริ่มต้น

```
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);
        }
    }
}
```

ผลลัพธ์ของโปรแกรม

```
j=1, thread:1
j=2, thread:1
j=13, thread:1
j=24, thread:1
j=25, thread:1
j=36, thread:1
j=47, thread:1
j=48, thread:1
j=49, thread:1
j=50, thread:1
j=41, thread:1
j=42, thread:1
j=43, thread:1
j=44, thread:1
j=45, thread:1
j=46, thread:1
j=47, thread:1
j=48, thread:1
j=49, thread:1
j=50, thread:1
j=41, thread:1
j=42, thread:1
j=43, thread:1
j=44, thread:1
j=45, thread:1
j=46, thread:1
j=47, thread:1
j=48, thread:1
j=49, thread:1
j=50, thread:1
```

```
j=41, thread:1
j=42, thread:1
j=43, thread:1
j=44, thread:1
j=45, thread:1
j=46, thread:1
j=47, thread:1
j=48, thread:1
j=49, thread:1
j=50, thread:1
j=41, thread:1
j=42, thread:1
j=43, thread:1
j=44, thread:1
j=45, thread:1
j=46, thread:1
j=47, thread:1
j=48, thread:1
j=49, thread:1
j=50, thread:1
j=41, thread:1
j=42, thread:1
j=43, thread:1
j=44, thread:1
j=45, thread:1
j=46, thread:1
j=47, thread:1
j=48, thread:1
j=49, thread:1
j=50, thread:1
```

ปัญหาที่พบ

1. ค่า j ที่ได้จากการ DeQueue นั้นมีค่าที่หายไปและไม่เรียงลำดับ
2. ในช่วงท้ายมีการแสดงข้อมูลที่ซ้ำกัน

สาเหตุของปัญหา

1. มีการทำ EnQueue เร็วกว่า DeQueue ทำให้ค่าที่ออกมาไม่เรียงลำดับ
2. มีการ DeQueue 60 ครั้งแต่ EnQueue แค่ 50 ครั้ง ทำให้เมื่อ EnQueue ครบแล้ว แต่ DeQueue ยังเหลือการทำงานต่อ จึงเกิดการแสดงข้อมูลที่ซ้ำกัน
3. ไม่มีการตรวจสอบว่า Buffer ณ ขณะนั้นมีข้อมูลอยู่กี่ตัว ทำให้เกิดเหตุการณ์เมื่อ Buffer เต็มแล้ว แต่ยังทำการ EnQueue ต่อไป ทำให้ข้อมูลที่ยังไม่ได้ DeQueue ถูกเขียนทับ

VERSION 1

สิ่งที่แก้

```
static void EnQueue(int eq)
{
    while (Count >= 10) { } ;

    TSBuffer[Back] = eq;
    Back++;
    Back %= 10;
    Count += 1;
}
```

คำอธิบาย

มีการใส่ Loop While(Count >= 10){}; เพื่อให้เมื่อ Buffer เต็มจะทำการรอไปเรื่อยๆจนกว่า Buffer จะว่าง (มีการ DeQueue ออก) ส่งผลให้ข้อมูลครบ ไม่มีการถูกเขียนทับ

```
1 reference
static int DeQueue()
{


    int x = 0;
    x = TSBuffer[Front];
    TSBuffer[Front] = -1 ;
    Front++;
    Front %= 10;
    Count -= 1;
    return x;
}
```

คำอธิบาย

มีการใส่ TSBuffer[Front] = -1; เมื่อเวลามีการ DeQueue ออกจะเปลี่ยนให้ค่าของ Buffer ตำแหน่งนั้นมีค่าเป็น -1 เพื่อให้เวลามีการ DeQueue มากกว่าข้อมูลที่ EnQueue เข้ามา เวลาไม่มีข้อมูลแล้วจะแสดงค่าข้อมูลเป็น -1 แทน

ผลลัพธ์ของโปรแกรม

แบบ Single Thread

```
j=1, thread:1
j=2, thread:1
j=3, thread:1
j=4, thread:1
j=5, thread:1
j=6, thread:1
j=7, thread:1
j=8, thread:1
j=9, thread:1
j=10, thread:1
j=11, thread:1
j=12, thread:1
j=13, thread:1
j=14, thread:1
j=15, thread:1
j=16, thread:1
j=17, thread:1
j=18, thread:1
j=19, thread:1
j=20, thread:1
j=21, thread:1
j=22, thread:1
j=23, thread:1
j=24, thread:1
j=25, thread:1
j=26, thread:1
j=27, thread:1
j=28, thread:1
j=29, thread:1
j=30, thread:1
```

```
j=31, thread:1
j=32, thread:1
j=33, thread:1
j=34, thread:1
j=35, thread:1
j=36, thread:1
j=37, thread:1
j=38, thread:1
j=39, thread:1
j=40, thread:1
j=41, thread:1
j=42, thread:1
j=43, thread:1
j=44, thread:1
j=45, thread:1
j=46, thread:1
j=47, thread:1
j=48, thread:1
j=49, thread:1
j=50, thread:1
j=-1, thread:1
j=-1, thread:1
j=-1, thread:1
j=-1, thread:1
j=-1, thread:1
j=-1, thread:1
j=-1, thread:1
j=-1, thread:1
j=-1, thread:1
j=-1, thread:1
```

แบบ Multi Thread

```
j=100, thread:1
j=1, thread:1
j=101, thread:1
j=0, thread:1
j=102, thread:1
j=3, thread:1
j=103, thread:1
j=4, thread:1
j=104, thread:1
j=0, thread:1
j=6, thread:1
j=7, thread:1
j=107, thread:1
j=108, thread:1
j=9, thread:1
j=10, thread:1
j=110, thread:1
j=111, thread:1
j=12, thread:1
j=112, thread:1
j=13, thread:1
j=114, thread:1
j=115, thread:1
j=116, thread:1
j=117, thread:1
j=16, thread:1
j=118, thread:1
j=119, thread:1
j=17, thread:1
j=18, thread:1
```

ปัญหาที่พบ

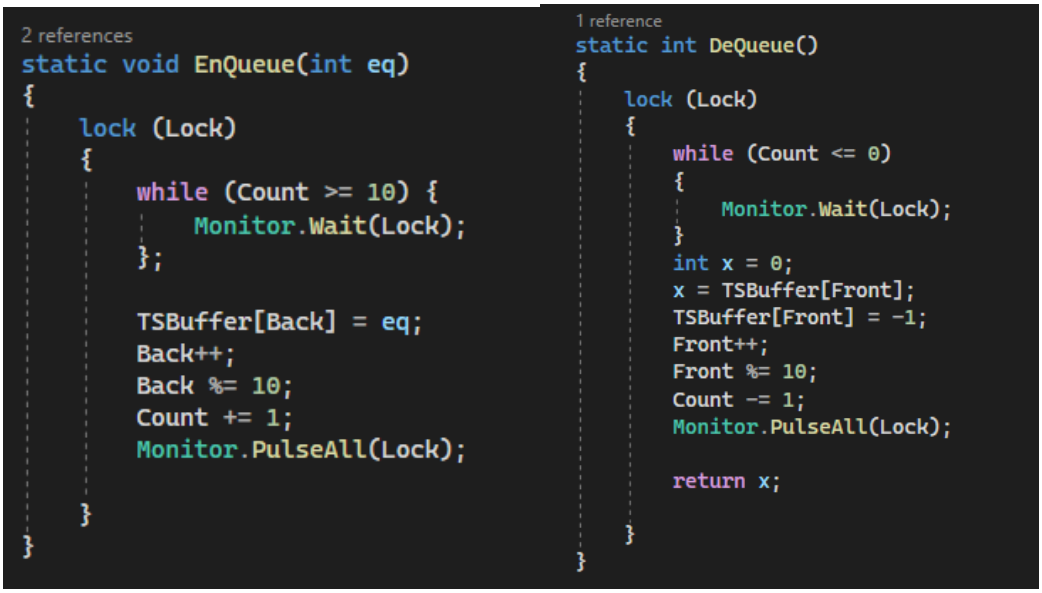
1. จากผลลัพธ์เป็นการทำงานแบบ single thread ซึ่งมีการทำงานถูกต้อง แต่เวลาไม่มีข้อมูลแล้วจะแสดงค่าข้อมูลเป็น -1
2. เมื่อลองเพิ่ม Thread t2 ที่ทำงาน EnQueue เพิ่มขึ้นมา พบว่ามีข้อมูลบางช่วงหายไปและไม่เรียงลำดับ

สาเหตุของปัญหา

1. เมื่อมี Thread ที่ทำ EnQueue มากกว่า 1 ทำให้มีจุดที่เป็น Critical Section ตรงตัวแปร Count ทำให้ข้อมูลบน Buffer บางตัวอาจถูกเขียนทับได้
2. โปรแกรมไม่สามารถทำงานจนจบได้เพราะ EnQueue ในแต่ละ Thread ทำงานรวมกันได้ 100 รอบ แต่ว่า DeQueue ทำได้แค่ 60 รอบ ทำให้ค่า Count นั้นไม่ลดลงต่ำกว่าหรือเท่ากับ 10 ส่งผลให้ ติด Loop while(Count >= 10)

VERSION 2

สิ่งที่แก้



```
2 references
static void EnQueue(int eq)
{
    lock (Lock)
    {
        while (Count >= 10) {
            Monitor.Wait(Lock);
        };

        TSBuffer[Back] = eq;
        Back++;
        Back %= 10;
        Count += 1;
        Monitor.PulseAll(Lock);

    }
}
```

```
1 reference
static int DeQueue()
{
    lock (Lock)
    {
        while (Count <= 0)
        {
            Monitor.Wait(Lock);
        }
        int x = 0;
        x = TSBuffer[Front];
        TSBuffer[Front] = -1;
        Front++;
        Front %= 10;
        Count -= 1;
        Monitor.PulseAll(Lock);

        return x;

    }
}
```

ในนี้มีการใช้ lock (Lock) และ Monitor ในฟังก์ชัน EnQueue และ DeQueue เพื่อควบคุมการเข้าถึง buffer โดยสร้าง Lock เพื่อให้สามารถทำงานร่วมกันระหว่าง Thread ได้อย่างปลอดภัยเพื่อป้องกันปัญหา Race Condition ในสถานการณ์ที่มีหลาย Thread พยายามเข้าถึง buffer พร้อมกัน

คำอธิบาย

เมื่อ Thread เข้าสู่ฟังก์ชัน EnQueue หรือ DeQueue จะมีการใช้ lock เพื่อล็อกส่วนที่เป็นที่สนใจของ Buffer ซึ่งทำให้ Thread อื่นไม่สามารถเข้าถึงส่วนนี้ได้ในขณะ เดียวกัน

เมื่อ Thread อื่นพยายามเข้าถึง Buffer ในฟังก์ชัน EnQueue หรือ DeQueue ในขณะที่มี Thread อื่นอยู่ในส่วน lock จะถูกบล็อกและรอจนกว่า Thread อื่นจะปลดล็อกส่วน lock นั้น จึงจะสามารถเข้าถึง Buffer ได้ ซึ่งสร้างความปลอดภัยในการเข้าถึงข้อมูลใน Buffer ระหว่าง Thread แต่ละตัว

เมื่อ Thread ทำงานเสร็จสิ้นในฟังก์ชัน EnQueue หรือ DeQueue จะปลดล็อกส่วน lock และใช้ Monitor.PulseAll เพื่อส่ง Signal แจ้งถึง Thread อื่นที่รอคอยในส่วน lock ว่าสามารถทำงานต่อได้ ซึ่งจะทำให้ Thread ที่รอคอยสามารถทำงานต่อไปได้อย่างถูกต้องและมีความปลอดภัย

ผลลัพธ์ของโปรแกรม

แบบ Single Thread

```
Dequeue : 1     thread : 1
Dequeue : 100   thread : 1
Dequeue : 2     thread : 1
Dequeue : 101   thread : 1
Dequeue : 3     thread : 1
Dequeue : 102   thread : 1
Dequeue : 4     thread : 1
Dequeue : 103   thread : 1
Dequeue : 5     thread : 1
Dequeue : 104   thread : 1
Dequeue : 105   thread : 1
Dequeue : 6     thread : 1
Dequeue : 106   thread : 1
Dequeue : 7     thread : 1
Dequeue : 8     thread : 1
Dequeue : 107   thread : 1
Dequeue : 108   thread : 1
Dequeue : 9     thread : 1
Dequeue : 10    thread : 1
Dequeue : 109   thread : 1
Dequeue : 110   thread : 1
Dequeue : 11    thread : 1
Dequeue : 12    thread : 1
Dequeue : 111   thread : 1
Dequeue : 13    thread : 1
Dequeue : 112   thread : 1
Dequeue : 113   thread : 1
Dequeue : 14    thread : 1
Dequeue : 114   thread : 1
Dequeue : 15    thread : 1
```

```
Dequeue : 16    thread : 1
Dequeue : 115   thread : 1
Dequeue : 116   thread : 1
Dequeue : 17    thread : 1
Dequeue : 18    thread : 1
Dequeue : 117   thread : 1
Dequeue : 19    thread : 1
Dequeue : 118   thread : 1
Dequeue : 20    thread : 1
Dequeue : 119   thread : 1
Dequeue : 120   thread : 1
Dequeue : 21    thread : 1
Dequeue : 22    thread : 1
Dequeue : 121   thread : 1
Dequeue : 23    thread : 1
Dequeue : 122   thread : 1
Dequeue : 123   thread : 1
Dequeue : 24    thread : 1
Dequeue : 124   thread : 1
Dequeue : 25    thread : 1
Dequeue : 26    thread : 1
Dequeue : 125   thread : 1
Dequeue : 27    thread : 1
Dequeue : 126   thread : 1
Dequeue : 127   thread : 1
Dequeue : 28    thread : 1
Dequeue : 128   thread : 1
Dequeue : 29    thread : 1
Dequeue : 30    thread : 1
Dequeue : 129   thread : 1
```

แบบ Multi Thread

```
Dequeue : 100   thread : 2
Dequeue : 1     thread : 1
Dequeue : 2     thread : 3
Dequeue : 101   thread : 2
Dequeue : 3     thread : 1
Dequeue : 102   thread : 2
Dequeue : 4     thread : 1
Dequeue : 103   thread : 3
Dequeue : 5     thread : 2
Dequeue : 104   thread : 1
Dequeue : 6     thread : 3
Dequeue : 105   thread : 1
Dequeue : 7     thread : 1
Dequeue : 106   thread : 2
Dequeue : 8     thread : 3
Dequeue : 107   thread : 1
Dequeue : 9     thread : 3
Dequeue : 108   thread : 1
Dequeue : 10    thread : 2
Dequeue : 109   thread : 3
Dequeue : 11    thread : 3
Dequeue : 110   thread : 3
Dequeue : 12    thread : 2
Dequeue : 111   thread : 2
Dequeue : 13    thread : 1
Dequeue : 112   thread : 3
Dequeue : 14    thread : 2
Dequeue : 113   thread : 2
Dequeue : 15    thread : 1
Dequeue : 114   thread : 2
Dequeue : 16    thread : 1
```

```
Dequeue : 115   thread : 1
Dequeue : 17    thread : 2
Dequeue : 116   thread : 2
Dequeue : 18    thread : 2
Dequeue : 117   thread : 1
Dequeue : 19    thread : 2
Dequeue : 118   thread : 1
Dequeue : 20    thread : 2
Dequeue : 119   thread : 3
Dequeue : 21    thread : 2
Dequeue : 120   thread : 3
Dequeue : 22    thread : 1
Dequeue : 121   thread : 1
Dequeue : 23    thread : 3
Dequeue : 122   thread : 2
Dequeue : 24    thread : 2
Dequeue : 123   thread : 3
Dequeue : 25    thread : 1
Dequeue : 124   thread : 2
Dequeue : 26    thread : 1
Dequeue : 125   thread : 1
Dequeue : 27    thread : 3
Dequeue : 126   thread : 2
Dequeue : 28    thread : 3
Dequeue : 127   thread : 1
Dequeue : 29    thread : 3
Dequeue : 128   thread : 1
Dequeue : 30    thread : 2
Dequeue : 129   thread : 3
Dequeue : 31    thread : 2
Dequeue : 130   thread : 1
Dequeue : 32    thread : 2
```

```
Dequeue : 131   thread : 2
Dequeue : 33    thread : 1
Dequeue : 132   thread : 3
Dequeue : 34    thread : 2
Dequeue : 133   thread : 3
Dequeue : 35    thread : 1
Dequeue : 134   thread : 3
Dequeue : 36    thread : 1
Dequeue : 135   thread : 1
Dequeue : 37    thread : 2
Dequeue : 136   thread : 1
Dequeue : 38    thread : 1
Dequeue : 137   thread : 3
Dequeue : 39    thread : 1
Dequeue : 138   thread : 3
Dequeue : 40    thread : 1
Dequeue : 139   thread : 2
Dequeue : 41    thread : 3
Dequeue : 140   thread : 1
Dequeue : 42    thread : 2
Dequeue : 141   thread : 1
Dequeue : 43    thread : 3
Dequeue : 142   thread : 1
Dequeue : 44    thread : 1
Dequeue : 143   thread : 1
Dequeue : 45    thread : 2
Dequeue : 144   thread : 2
Dequeue : 46    thread : 1
Dequeue : 145   thread : 3
Dequeue : 47    thread : 2
Dequeue : 146   thread : 1
Dequeue : 48    thread : 2
Dequeue : 147   thread : 2
Dequeue : 49    thread : 3
Dequeue : 148   thread : 3
Dequeue : 50    thread : 1
Dequeue : 149   thread : 2
Dequeue : 150   thread : 1
```

ปัญหาที่พบ

โปรแกรมไม่ยอมจบการทำงาน

สาเหตุของปัญหา

เนื่องจากการทำงานของ EnQueue และ DeQueue นั้น จะมีการตรวจสอบเงื่อนไขอยู่ทุกครั้ง คือ ตรวจสอบขนาดของ Buffer ว่ามีสมาชิกอยู่กี่ตัว แล้วสามารถที่จะทำการ EnQueue หรือ DeQueue ได้หรือไม่ ถ้า Buffer มีสมาชิกครบก็จะไม่สามารถ EnQueue เพิ่มได้ และถ้า Buffer ไม่มีสมาชิกเหลืออยู่เลย ก็จะไม่สามารถ DeQueue ได้

ซึ่งถ้าตรวจสอบเงื่อนไขแล้วพบว่าไม่สามารถที่จะ EnQueue หรือ DeQueue ได้ Thread ดังกล่าว จะเปลี่ยนสถานะจาก Running เป็น Wait แล้วรอจนกว่าจะมีสัญญาณ Signal จากอีก Thread หนึ่งส่งมา และต้องเป็น Thread ที่มีการทำงานอีกแบบหนึ่ง

โดยจะมีลักษณะก็คือ ถ้าเป็น Thread ที่ทำการ EnQueue นั้นไม่สามารถ EnQueue ต่อไปได้ ก็จะ เข้าสู่สถานะ Wait แล้วรอ Signal จาก Thread ที่มีการ DeQueue ผ่านคำสั่ง Monitor.PulseAll(lock_)

และในทางกลับกัน ถ้าเป็น Thread ที่ทำการ DeQueue นั้นไม่สามารถ DeQueue ต่อไปได้ ก็จะเข้าสู่สถานะ Wait แล้วรอ Signal จาก Thread ที่มีการ EnQueue ผ่านคำสั่ง Monitor.PulseAll(lock_)

สรุปได้ว่า จากการทำงานของโปรแกรมนั้น เป็นลักษณะที่มีการ EnQueue 50 ครั้ง และ DeQueue 60 ครั้ง ซึ่งแน่นอนว่าการ EnQueue นั้นครบ 50 ครั้ง แต่การ DeQueue จะเหลืออยู่ 10 ครั้ง ทำให้การ DeQueue ครั้งที่ 51 อยู่ในสถานะ Wait ค้างตลอด และไม่มีการได้รับ Signal จาก EnQueue เนื่องจากการ EnQueue นั้นได้เสร็จสิ้นลงไปแล้ว ทำให้ไม่มีการส่ง Signal ผ่าน Monitor.PulseAll(lock_) อีกแล้ว แล้วทำให้โปรแกรมค้างอยู่ที่คำสั่ง Monitor.Wait(lock_);

VERSION 3

สิ่งที่แก้

จากก่อนหน้านี้ที่ใช้ lock ในการป้องกัน race condition ทำให้โปรแกรมค้างที่คำสั่ง Monitor.Wait(lock_);

```
static void EnQueue(int eq)
{
    semaphore1.WaitOne(); // รอให้ได้สิทธิ์ในการเข้าถึง

    DateTime startTime = DateTime.Now; // เวลาเริ่มต้นการรอ

    while (Count >= 10)
    {
        if ((DateTime.Now - startTime).TotalMilliseconds > 300)
        {
            Environment.Exit(0);    // Timeout
        }
    }

    lock (lock_)
    {
        TSBuffer[Back] = eq;
        Back++;
        Back %= 10;
        Count += 1;
    }

    semaphore1.Release(); // รอให้ได้สิทธิ์ในการเข้าถึง
}
```

```
static int DeQueue()
{
    int x = 0;

    semaphore2.WaitOne(); // รอให้ได้สิทธิ์ในการเข้าถึง

    DateTime startTime2 = DateTime.Now;

    while (Count <= 0)
    {
        if ((DateTime.Now - startTime2).TotalMilliseconds > 300)
        {
            Environment.Exit(0);     // Timeout
        }
    }

    lock (lock_)
    {
        x = TSBuffer[Front];
        Front++;
        Front %= 10;
        Count -= 1;
    }

    semaphore2.Release(); // ปลดสิทธิ์การเข้าถึง
    return x;
}
```

แก้ไขโดยการใช้ semaphore เข้ามาช่วยควบคุมการเข้าถึง buffer ที่ใช้ร่วมกันระหว่าง EnQueue และ EnQueue แบบ Multitread ไปอีกชั้น แต่เหตุผลที่ใช้ semaphore คือการที่ไม่ต้องใช้ คำสั่ง Monitor.Wait(lock_); ซึ่งเป็นจุดที่ทำให้โปรแกรมค้าง แล้วเปลี่ยนไปใช้การตรวจสอบการเกิด Timeout แทน

```
DateTime startTime = DateTime.Now; // เวลาเริ่มต้นการรอ

while (Count >= 10)
{
    if ((DateTime.Now - startTime).TotalMilliseconds > 300)
    {
        Environment.Exit(0);    // Timeout
    }
}
```

```
DateTime startTime2 = DateTime.Now;

while (Count <= 0)
{
    if ((DateTime.Now - startTime2).TotalMilliseconds > 300)
    {
        Environment.Exit(0);     // Timeout
    }
}
```

ผลลัพธ์ของโปรแกรม

```
Dequeue : 115   thread : 3
Dequeue : 16    thread : 2
Dequeue : 116   thread : 1
Dequeue : 17    thread : 2
Dequeue : 18    thread : 3
Dequeue : 117   thread : 2
Dequeue : 118   thread : 1
Dequeue : 19    thread : 2
Dequeue : 20    thread : 3
Dequeue : 119   thread : 2
Dequeue : 120   thread : 1
Dequeue : 21    thread : 2
Dequeue : 121   thread : 3
Dequeue : 22    thread : 2
Dequeue : 23    thread : 3
Dequeue : 122   thread : 1
Dequeue : 24    thread : 2
Dequeue : 123   thread : 3
Dequeue : 124   thread : 1
Dequeue : 25    thread : 3
Dequeue : 26    thread : 2
Dequeue : 125   thread : 3
Dequeue : 27    thread : 1
Dequeue : 126   thread : 3
Dequeue : 127   thread : 2
Dequeue : 28    thread : 3
Dequeue : 29    thread : 1
Dequeue : 128   thread : 3
Dequeue : 129   thread : 2
Dequeue : 30    thread : 3
Dequeue : 1     thread : 1
Dequeue : 100   thread : 2
Dequeue : 2     thread : 3
Dequeue : 101   thread : 1
Dequeue : 3     thread : 2
Dequeue : 102   thread : 1
Dequeue : 103   thread : 3
Dequeue : 4     thread : 1
Dequeue : 104   thread : 2
Dequeue : 5     thread : 1
Dequeue : 105   thread : 3
Dequeue : 6     thread : 1
Dequeue : 106   thread : 2
Dequeue : 7     thread : 1
Dequeue : 8     thread : 3
Dequeue : 107   thread : 1
Dequeue : 9     thread : 2
Dequeue : 108   thread : 1
Dequeue : 10    thread : 3
Dequeue : 109   thread : 2
Dequeue : 11    thread : 1
Dequeue : 110   thread : 2
Dequeue : 111   thread : 3
Dequeue : 12    thread : 2
Dequeue : 112   thread : 1
Dequeue : 13    thread : 2
Dequeue : 14    thread : 3
Dequeue : 113   thread : 2
Dequeue : 15    thread : 1
Dequeue : 114   thread : 2
Dequeue : 31    thread : 1
Dequeue : 130   thread : 3
Dequeue : 32    thread : 2
Dequeue : 131   thread : 3
Dequeue : 132   thread : 1
Dequeue : 33    thread : 3
Dequeue : 34    thread : 2
Dequeue : 133   thread : 3
Dequeue : 134   thread : 1
Dequeue : 35    thread : 3
Dequeue : 36    thread : 2
Dequeue : 135   thread : 3
Dequeue : 136   thread : 3
Dequeue : 37    thread : 1
Dequeue : 137   thread : 3
Dequeue : 38    thread : 2
Dequeue : 138   thread : 1
Dequeue : 39    thread : 2
Dequeue : 40    thread : 3
Dequeue : 139   thread : 2
Dequeue : 41    thread : 1
Dequeue : 140   thread : 2
Dequeue : 42    thread : 3
Dequeue : 141   thread : 2
Dequeue : 43    thread : 1
Dequeue : 142   thread : 2
Dequeue : 44    thread : 3
Dequeue : 143   thread : 2
Dequeue : 144   thread : 1
Dequeue : 45    thread : 2
Dequeue : 46    thread : 3
Dequeue : 145   thread : 2
Dequeue : 47    thread : 1
Dequeue : 146   thread : 2
Dequeue : 147   thread : 3
Dequeue : 48    thread : 2
Dequeue : 49    thread : 1
Dequeue : 148   thread : 3
Dequeue : 50    thread : 2
Dequeue : 149   thread : 1
Dequeue : 150   thread : 3

C:\Users\jj\source\repos\OS2\OS2\bin\Debug\net7.0\OS2.exe (process 28832) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

โปรแกรมยังคงทำงานได้ถูกต้องตาม Version 2 ทุกกรณี ทั้งกรณีที่จำนวน EnQueue > DeQueue, จำนวน EnQueue < DeQueue และ จำนวน EnQueue = DeQueue ก็สามารถจบการทำงานได้โดยไม่มี Thread ใด ๆ ค้างอยู่ในสถานะ Wait