Tuần 4

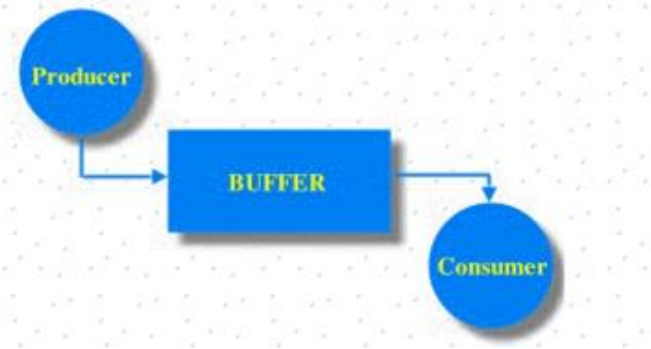
ĐỒNG BỘ HÓA CÁC TIẾN TRÌNH

I. Yêu cầu:

- Sử dụng kỹ thuật lập trình **C Sharp.**
- Viết chương trình đồng bộ hóa các tiến trình để giải quyết bài toán **Producer/Customer** sử dụng:
  - Mutex và Semaphore.
  - Lock và Mornitor

II. Mô tả bài toán: Người sản xuất – Người tiêu thụ (Producer - Consumer)

Vấn đề: hai tiến trình cùng chia sẻ một bộ đệm có kích thước giới hạn. Một trong hai tiến trình đóng vai trò người sản xuất – tạo ra dữ liệu và đặt dữ liệu vào bộ đệm- và tiến trình kia đóng vai trò người tiêu thụ – lấy dữ liệu từ bộ đệm ra để xử lý.



Để đồng bộ hóa hoạt động của hai tiến trình sản xuất tiêu thụ cần tuân thủ các quy định sau :

- Tiến trình sản xuất (**producer**) không được ghi dữ liệu vào bộ đệm đã đầy.***(synchronisation)***
- Tiến trình tiêu thụ (**consumer**) không được đọc dữ liệu từ bộ đệm đang trống.***(synchronisation)***
- Hai tiến trình sản xuất và tiêu thụ **không được thao tác trên bộ đệm cùng lúc** . ***(exclusion mutuelle)***

III. Cấu trúc dữ liệu: khai báo các biến toàn cục

```
const int Buffersize = 10;   // Kích thước tối đa của bộ đệm
static  Queue<string>   BUFFER = new Queue<string>() ;
//Dùng BUFFER.Enqueue(item) để đưa một item vào cuối BUFFER.
//Dùng BUFFER.Dequeue(item) để hủy một item ở đầu BUFFER
```

IV.Giải pháp Semaphore : sử 3 biến

```
// mutex Kiểm soát truy xuất độc quyền
  static Mutex mutex = new Mutex();

// empty Kiểm soát số chổ trống
  static Semaphore empty = new Semaphore(Buffersize, Buffersize);
// full kiểm soát số chổ đầy
  static Semaphore full = new Semaphore(0, Buffersize);
```

//Sử dụng các phương thức **WaitOne(),Release()** của semaphore tương ứng với phương thức **Down()** và **Up()** để đồng bộ hóa.

- **Code chương trình:**

```
11      class Program
12      {
13          const int Buffersize = 10;
14          static  Queue<string>   BUFFER = new Queue<string>() ;
15
16          static Mutex mutex = new Mutex();
17          static Semaphore empty = new Semaphore(Buffersize, Buffersize);
18          static Semaphore full = new Semaphore(0, Buffersize);
19
```

```
20          private static void Producer()
21          {
22              for (int i = 1; i <= 100; i++)
23              {
24                  //Nếu số chổ trống empty=0 thì đợi
25                  empty.WaitOne();
26                  //Truy xuất độc quyền buffer
27                  mutex.WaitOne();
28                  //Đưa item vào buffer
29                  BUFFER.Enqueue("item" + i);
30                  Console.Write("#Producer: item{0} => BUFFER = ", i);
31                  foreach (string s in BUFFER)
32                      Console.Write(s + " ");
33                  Console.WriteLine();
34                  //Cho phép truy xuất buffer
35                  mutex.ReleaseMutex();
36                  //full>0 kích hoạt consumer đang đợi
37                  full.Release();
38              }
39          }
```

```
41          private static void Consumer()
42          {
43              for (int i = 1; i <= 100; i++)
44              {
45                  //Nếu số chổ đầy full=0 thì đợi
46                  full.WaitOne();
47                  //Độc quyền truy xuất Buffer
48                  mutex.WaitOne();
49                  //Lấy một item ra khỏi Buffer
50                  string item = BUFFER.Dequeue();
51                  Console.Write("*Consumer: {0} => BUFFER = ", item);
52                  foreach (string s in BUFFER)
53                      Console.Write(s + " ");
54                  Console.WriteLine();
55                  //Cho phép truy xuất buffer
56                  mutex.ReleaseMutex();
57                  //empty>0 kích hoạt Producer đang đợi
58                  empty.Release();
59              }
60          }
```

```
            0 references
62          static void Main(string[] args)
63          {
64              Thread t_producer = new Thread(Producer);
65              Thread t_consumer = new Thread(Consumer);
66              t_producer.IsBackground = true;
67              t_consumer.IsBackground = true;
68              t_producer.Start();
69              t_consumer.Start();
70              Console.ReadKey();
71          }
72      }
```

V. Giải pháp Mornitor:

- Sử dụng biến điều kiện (Condition)**locked**

```
static  Object locked= new Object();
```

- Sử dụng lock để truy xuất độc quyền đến biến locked

```
lock (locked) { …… } //Truy xuất độc quyền trên biến locked
```

- Sử dụng 2 phương thức sau để đồng bộ hóa

```
Monitor.Wait(locked);// Đợi trên biến locked
Monitor.Pulse(locked); //Tái kích hoạt tiến trình đang đợi trên loked
```

➢ C.Code Chương Trinh:

```
10      class Program
11      {
12          const int Buffersize = 10;
13          static int count = 0;
14          static  Object locked= new Object();
15          static  Queue<string> BUFFER = new Queue<string>();
```

```
37          static void Consumer()
38          {
39              for (int i = 1; i <= 50; i++)
40              {
41                  lock (locked)
42                  {
43                      if (count == 0)           //Nếu buffer rỗng
44                          Monitor.Wait(locked); //Đợi trên biến locked
45                      //Lấy một item ở đầu BUFFER
46                      string item = BUFFER.Dequeue();
47                      count--;
48                      //In BUFFER
49                      Console.Write("*Consumer: {0} => BUFFER: ", item);
50                      foreach (string s in BUFFER)
51                          Console.Write(s + " ");
52                      Console.WriteLine();
53                      if (count < Buffersize) // Nếu buffer có chỗ trống
54                          Monitor.Pulse(locked);   // Tái kích hoạt Producer đang đợi trên locked
55                  }
56              }
57          }
```

```
16          static void Producer()
17          {
18              for (int i = 1; i <= 50; i++)
19              {
20                  lock (locked)  //Truy xuất độc quyền trên biến locked
21                  {
22                      if (count == Buffersize) //Nếu BUFFER đầy
23                          Monitor.Wait(locked);// Đợi trên biến locked
24                      //Dua Item vao BUFFER
25                      BUFFER.Enqueue("item" + i);
26                      count++;
27                      //In BUFFER
28                      Console.Write("#Producer: item{0} => BUFFER: ", i);
29                      foreach (string s in BUFFER)
30                          Console.Write(s + " ");
31                      Console.WriteLine();
32                      if (count > 0)  //Nếu Buffer có ít nhất 1 item
33                          Monitor.Pulse(locked); //Tái kích hoạt Consumer đang đợi trên loked
34                  }
35              }
36          }
```

```
58          static void Main(string[] args)
59              {
60                  Thread t_producer = new Thread(Producer);
61                  Thread t_consumer = new Thread(Consumer);
62                  t_producer.IsBackground = true;
63                  t_consumer.IsBackground = true;
64                  t_producer.Start();
65                  t_consumer.Start();
66                  Console.ReadKey();
67              }
68      }
69  }
```

bool Thread.IsBackground { get; set; }

VI. Bài tập: Sử dụng semaphore hoặc monitor viết chương trình đồng bộ hóa các bài tập sau:

1. **Bài tập 1**: Viết chương trình thực hiện 2 Thread chạy song song như sau:

```
static void ThreadA()
    {
        int na = 0;
        for(int i=1;i<=100;i++)
        {
            na = na + 1;
            Console.Write("na= " + na+ "\t");
        }

    }
static void ThreadB()
    {
        int nb = 0;
        while (true)
        {
            nb = nb + 1;
            Console.Write("nb= " + nb+"\t");
            if (nb == 100)
                break;
        }

    }
```

   - **Yêu cầu:** Thực đồng bộ hóa sao cho tại một thời điểm bất kỳ thì: **nb <= na <= nb+10.**

2. **Bài tập 2**: Viết chương trình theo mô hình song hành gồm 7 thread xử lý song song tương ứng 7 biểu thức sau:

   w := x1 * x2
   v := x3 * x4
   y := v * x5
   z := v * x6
   y := w * y
   z := w * z
   ans := y + z

   - **Yêu cầu:** Với x1, x2, x3, x4, x5, x6 được khởi tạo giá trị ban đầu**.** Thực hiện đồng bộ hóa 7 thread sau cho các biểu thức thực hiện đúng thứ tự và trả về đúng giá trị **ans.**

3. **Bài tập 3**: Viết chương trình nhập vào mảng X gồm n phần tử số nguyên. Tạo 3 Thread ThreadA, ThreadB,ThreadC thực hiện xử lý song song các thao tác trên mảng X thỏa mảng các yêu cầu sau:
   - ThreadA tính tổng các phần tử chẵn.
   - ThreadB tính tổng các phần tử lẻ.
   - ThreadC tính tổng các phần tử trong mảng được lấy kết quả từ threadA và threadB.