Chương 11

Lập trình song song bằng VC#

11.0 Dẫn nhập

Chương này giới thiệu kiến thức tổng quát về lập trình song song, class Process của môi trường .Net phục vụ lập trình multi-process, class Thread phục vụ lập trình multi-thread.

Chương này cũng giới thiệu các chương trình demo cho tính hiệu quả của lập trình multi-thread so với lập trình tuần tự, vấn đề tranh chấp giữa các thread về việc truy xuất tài nguyên dùng chung đồng thời, cách thức giải quyết tranh chấp và hệ lụy của việc giải quyết tranh chấp.

11.1 Tổng quát về lập trình song song

Thường để giải quyết bài toán nào đó, ta thường dùng giải thuật tuần tự nhờ tính dễ hiểu, dễ kiểm soát của nó. Chương trình dùng thuật giải tuần tự khi chạy trở thành process mono-thread hay process tuần tự.

Process tuần tự hoạt động không hiệu quả vì không lợi dụng triệt để được các CPU xử lý trên máy tính vật lý. Lưu ý rằng hiện nay các máy PC, smartphone hay tablet đều dùng CPU đa nhân. Thí dụ galaxy S4 ở thị trường Việt Nam có 8 nhân.

Để máy giải quyết bài toán hiệu quả hơn, ta nên dùng thuật toán song song bằng cách nhận dạng các hoạt động có thể thực hiện đồng thời rồi nhờ nhiều CPU thực hiện chúng đồng thời.

Một trong các phương pháp hiện thực thuật toán song song là lập trình multi-process và multi-thread.

11.2 Lập trình multi-process bằng class Process

Môi trường .Net cung cấp class tên là Process để giúp ta lập trình multi-process dễ dàng.

Class Process thuộc namespace System.Diagnostics, nó chứa các thuộc tính và tác vụ giúp ta quản lý process dễ dàng, thuận lợi.

Thí dụ thuộc tính StartInfo là 1 đối tượng gồm nhiều thuộc tính xác định thông tin để kích hoạt ứng dụng xác định :

```
//tạo đối tượng quản lý process mới
Process myProcess = new Process();
//thiết lập file khả thi cần chạy
myProcess.StartInfo.FileName = txtPath.Text;
//không dùng Shell để chạy process mới
myProcess.StartInfo.UseShellExecute = false;
//tạo cửa sổ giao diện riêng cho process mới
myProcess.StartInfo.CreateNoWindow = true;
....
```

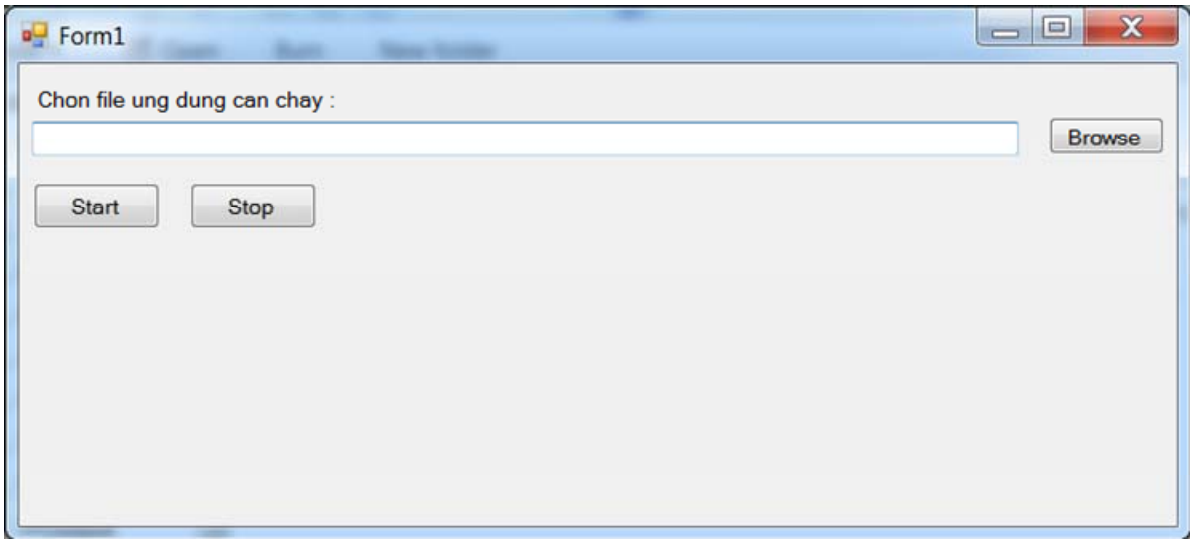
Sau khi thiết lập đầy đủ các thông tin để khởi tạo process, ta có thể gọi tác vụ Start để kích hoạt nó chạy :

```
myProcess.Start();
```

Sau khi được kích hoạt, process sẽ chạy song hành và độc lập với process kích hoạt nó cho đến khi kết thúc theo thuật giải của nó. Tuy nhiên, từ bên ngoài ta có thể giết process nhờ tác vụ Kill :

```
myProcess.Kill();
```

Ta hãy thử viết 1 ứng dụng quản lý process đơn giản có form giao diện như sau :



11.3 Lập trình multi-threads bằng class Thread

Môi trường .Net cung cấp class tên là Thread để giúp ta lập trình multi-thread dễ dàng.

Class Thread thuộc namespace System.Threading, nó chứa các thuộc tính và tác vụ giúp ta quản lý thread dễ dàng, thuận lợi.

Thường mỗi thread sẽ chạy đoạn code được miêu tả trong 1 hàm chức năng xác định. Thí dụ khi process được kích hoạt, HĐH sẽ tạo tường minh thread ban đấu cho process đó, thread chính này sẽ chạy đoạn code của hàm Main của class ứng dụng.

Để tạo thread mới, ta có thể dùng lệnh :

```
Thread t = new Thread
        (new ParameterizedThreadStart(tenhamcanchay));
```

Để kích hoạt chạy thread, ta có thể gọi tác vụ Start :

```
t.Start (new Params(danhsachthamso));
```

với Params là class đối tượng chứa các thông số mà ta muốn truyền/nhận cho thread mới.

Lưu ý tác vụ mà thread sẽ chạy phải được đặc tả với tham số hình thức là kiểu object :

```
void TinhTich (object obj) {    //tác vụ mà thread sẽ chạy
    Params p = (Params)obj; //ép kiểu tham số về kiểu mong muốn
```

```
    ...
}
```

Để tạm dừng thread, ta có thể gọi tác vụ Suspend :

```
t.Suspend();
```

Để chạy tiếp thread, ta có thể gọi tác vụ Resume :

```
t.Resume();
```

Để dừng và xóa thread, ta có thể gọi tác vụ Abort :

```
t.Abort();
```

Để thay đổi quyền ưu tiên thread, ta thực hiện lệnh gán :

```
t.Priority = ThreadPriority.Normal;
```

Trên Windows, mỗi process có thể ở 1 trong 6 cấp quyền ưu tiên sau đây :

IDLE_PRIORITY_CLASS

BELOW_NORMAL_PRIORITY_CLASS

NORMAL_PRIORITY_CLASS

ABOVE_NORMAL_PRIORITY_CLASS

HIGH_PRIORITY_CLASS

REALTIME_PRIORITY_CLASS

Cấp quyền ưu tiên của process sẽ quyết định các thread trong process đó chạy theo quyền ưu tiên như thế nào.

Trên Windows, mỗi thread trong process có thể ở 1 trong 7 cấp quyền ưu tiên sau đây :

THREAD_PRIORITY_IDLE

THREAD_PRIORITY_LOWEST

THREAD_PRIORITY_BELOW_NORMAL

THREAD_PRIORITY_NORMAL

THREAD_PRIORITY_ABOVE_NORMAL

THREAD_PRIORITY_HIGHEST

THREAD_PRIORITY_TIME_CRITICAL

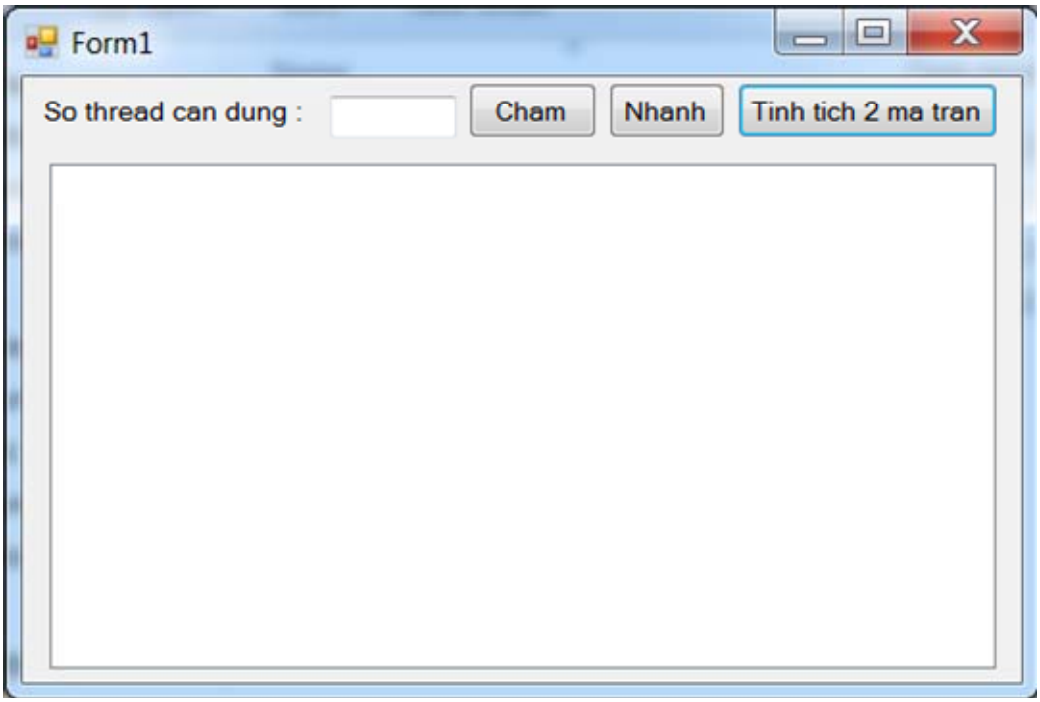
Cấp quyền ưu tiên của process sẽ quyết định các thread với từng cấp quyền ưu tiên trên đây sẽ được xử lý như thế nào.

Process priority class	Thread priority level	Base priority
IDLE_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	2
	THREAD_PRIORITY_BELOW_NORMAL	3
	THREAD_PRIORITY_NORMAL	4
	THREAD_PRIORITY_ABOVE_NORMAL	5
	THREAD_PRIORITY_HIGHEST	6
	THREAD_PRIORITY_TIME_CRITICAL	15
BELOW_NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	4
	THREAD_PRIORITY_BELOW_NORMAL	5
	THREAD_PRIORITY_NORMAL	6
	THREAD_PRIORITY_ABOVE_NORMAL	7
	THREAD_PRIORITY_HIGHEST	8
	THREAD_PRIORITY_TIME_CRITICAL	15
NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	6
	THREAD_PRIORITY_BELOW_NORMAL	7
	THREAD_PRIORITY_NORMAL	8
	THREAD_PRIORITY_ABOVE_NORMAL	9
	THREAD_PRIORITY_HIGHEST	10
	THREAD_PRIORITY_TIME_CRITICAL	15
ABOVE_NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	8
	THREAD_PRIORITY_BELOW_NORMAL	9

	THREAD_PRIORITY_NORMAL	10
	THREAD_PRIORITY_ABOVE_NORMAL	11
	THREAD_PRIORITY_HIGHEST	12
	THREAD_PRIORITY_TIME_CRITICAL	15
HIGH_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	11
	THREAD_PRIORITY_BELOW_NORMAL	12
	THREAD_PRIORITY_NORMAL	13
	THREAD_PRIORITY_ABOVE_NORMAL	14
	THREAD_PRIORITY_HIGHEST	15
	THREAD_PRIORITY_TIME_CRITICAL	15
REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	16
	THREAD_PRIORITY_LOWEST	22
	THREAD_PRIORITY_BELOW_NORMAL	23
	THREAD_PRIORITY_NORMAL	24
	THREAD_PRIORITY_ABOVE_NORMAL	25
	THREAD_PRIORITY_HIGHEST	26
	THREAD_PRIORITY_TIME_CRITICAL	31

11.4 Demo tính hiệu quả của multi-thread

Để demo tính hiệu quả của lập trình multi-thread trên các máy tính đa nhân, ta hãy thử viết ứng dụng tính tích của 2 ma trận có kích thước lớn (thí dụ 2000*2000). Ta thiết kế form ứng dụng như sau :

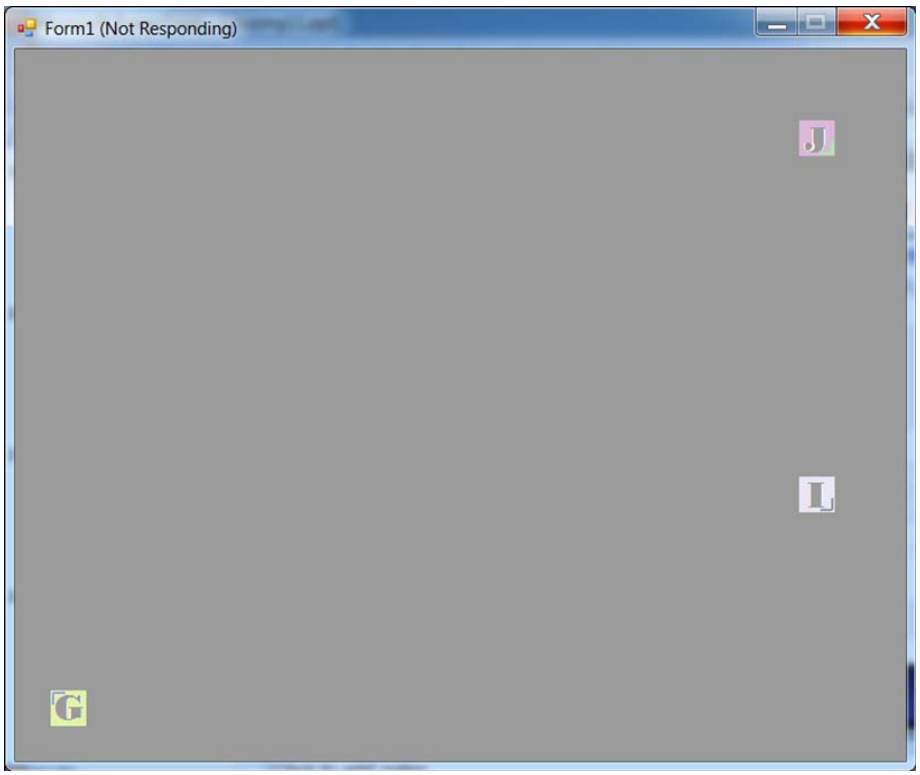


Ứng dụng cho phép người dùng nhập số thread cần dùng (n), rồi chia ma trận tích thành N/n nhóm hàng rồi tạo thread con để tính dùm từng nhóm hàng. Sau khi tính ma trận tích xong, ứng dụng sẽ hiển thị cho người dùng thấy thời gian tính toán để người dùng đánh giá độ hiệu quả.

```
void TinhTich (object obj) {
    DateTime t1 = DateTime.Now; //ghi nhận thời điểm bắt đầu chạy
    Params p = (Params)obj; //ép kiểu tham số về đối tượng cần dùng
    int h, c, k;
    for (h = p.sr; h < p.er; h++) //lặp theo hàng
        for (c = 0; c < N; c++) {  //lặp theo cột
            double s = 0;
            for (k = 0; k < N; k++)
                s = s + A[h, k] * B[k, c];
            C[h, c] = s;
        }
    stateLst[p.id] = 1;        //ghi nhận trạng thái hoàn thành
    dateLst[p.id] = DateTime.Now.Subtract(t1);//tính thời gian chạy
}
```

11.5 Demo vấn đề tương tranh giữa các thread

Để demo vấn đề tương tranh giữa các thread, ta hãy thử viết ứng dụng quản lý các thread với giao diện như sau :



Form là ma trận gồm nhiều cell, mỗi cell chứa được icon ảnh cho 1 thread đang chạy. Lúc đầu, chưa có thread nào chạy hết. Người dùng có thể ấn phím để quản lý các thread như sau :

- Ấn phím từ A-Z để kích hoạt chạy thread có tên tương ứng.
- Ấn phím Ctrl-Alt-X để tạm dừng chạy thread X.
- Ấn phím Alt-X để chạy tiếp thread X.
- Ấn phím Shift-X để tăng độ ưu tiên chạy cho thread X.
- Ấn phím Ctrl-X để giảm độ ưu tiên chạy cho thread X.
- Ấn phím Ctrl-Shift-X để dừng và thoát thread X.

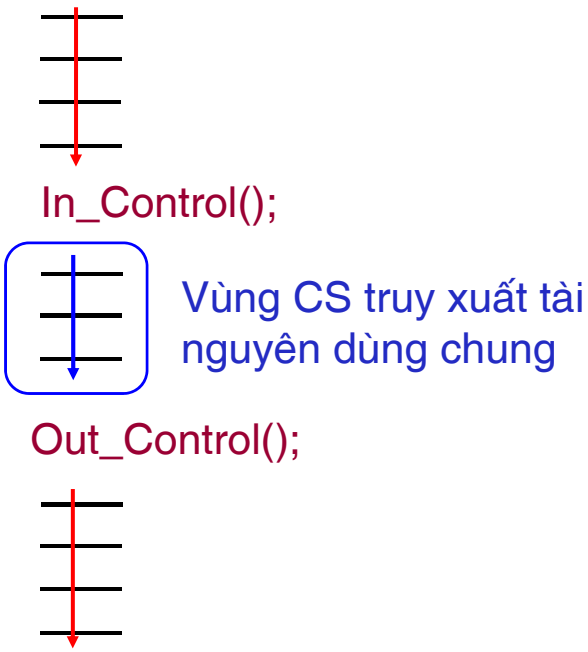
Để cho thấy hành vi hoạt động của thread, hoạt động của thread là hiển thị icon miêu tả mình lên form, icon này sẽ chạy theo 1 phương xác định, khi đụng thành form thì dội lại theo nguyên lý vật lý.

Quan sát quỹ đạo chạy icon miêu tả từng thread ta thấy thỉnh thoảng có hiện tượng icon thread này đè mất icon thread khác. Đây là hiện tượng lỗi không mong muốn do các thread được quyền tự do chiếm dụng từng cell hiển thị của form. Ta dùng thuật ngữ "tương tranh" giữa các thread trên các tài nguyên dùng chung (các cell của form).

Cần có biện pháp quản lý tương tranh sao cho các thread không được quyền truy xuất tài nguyên dùng chung đồng thời. Hiện nay ta dùng phương pháp loại trừ tương hỗ để giải quyết vấn đề này.

11.6 Demo việc giải quyết tương tranh giữa các thread

Mỗi lần muốn vào vùng CS, ta phải gọi hàm In_Control() để kiểm soát việc thi hành vùng CS, khi hoàn thành vùng CS, ta phải gọi hàm Out_Control() để thông báo cho các thread khác đang chờ để chúng kiểm tra lại việc đi vào.



Phương pháp loại trừ tương hỗ phổ dụng hiện nay là dùng semaphore nhị phân (Mutex). Semaphore là 1 đối tượng đơn giản chứa :

- 1 thuộc tính kiểu nguyên dương (s), ta còn gọi nó là biến semaphore.
- Tác vụ down(), có nhiệm vụ giảm s 1 đơn vị và luôn phải hoàn thành. Do đó trong trường hợp s = 0, tác vụ down sẽ phải ngủ chờ đến khi s <> 0 thì cố gắng thực hiện lại… ➔ Thời gian thi hành tác vụ down là không xác định.
- Tác vụ up(), có nhiệm vụ tăng s 1 đơn vị và luôn phải hoàn thành. Trong trường hợp s = 0, tác vụ up sẽ phải đánh thức các thread đang ngủ chờ down s dậy.

Môi trường .Net cung cấp class Mutex để quản lý semaphore nhị phân. Ta kết hợp mỗi tài nguyên dùng chung 1 mutex m với giá trị đầu = 1.

Hàm In_Control() sẽ là lệnh m.WaitOne(); Thread nào thực hiện lệnh này đầu tiên sẽ thành công ngay và sẽ chạy được đoạn lệnh CS truy xuất tài nguyên tương ứng. Các thread khác thực hiện lệnh trên để truy xuất tài nguyên dùng chung sẽ thất bại và bị ngủ trong khi thread đầu chưa hoàn thành truy xuất.

Khi hoàn thành việc truy xuất tài nguyên, thread gọi hàm Out_Control(). Trong trường hợp dùng Mutex, hàm Out_Control() sẽ là lệnh m.ReleaseMutex(); Nó sẽ đánh thức các thread đang ngủ chờ nếu có.

```
//xin khóa truy xuất cell bắt đầu (x1,y1)
mutList[y1, x1].WaitOne();
while (p.start) { //lặp trong khi chưa có yêu cầu kết thúc
    //hiển thị icon miêu tả mình lên cell
    //thực hiện công việc của thread
    //xác định vị trí mới của thread (x2,y2)
    //xin khóa truy xuất cell (x2,y2)
    while (true) {
        kq = mutList[y2, x2].WaitOne(new TimeSpan(0,0,2));
        if (kq==true || p.start==false) break;
    }
```

```
    // Xóa vị trí cũ
    //giải phóng cell (x1,y1) cho các thread khác truy xuất
    mutList[y1, x1].ReleaseMutex();
}
```

10.7 Kết chương

Chương này đã giới thiệu kiến thức tổng quát về lập trình song song, class Process của môi trường .Net phục vụ lập trình multi-process, class Thread phục vụ lập trình multi-thread.

Chương này cũng đã giới thiệu các chương trình demo cho tính hiệu quả của lập trình multi-thread so với lập trình tuần tự, vấn đề tranh chấp giữa các thread về việc truy xuất tài nguyên dùng chung đồng thời, cách thức giải quyết tranh chấp và hệ lụy của việc giải quyết tranh chấp.