Bài 12.11: Best practices

- ✓ Bế tắc và tương tranh
- ✓ Khuyến nghị chung
- ✓ Khuyến nghị khi thiết kế lớp cho thư viện

Deadlock và race conditions

- ✓Vấn đề nảy sinh khi xử lý tác vụ bằng đa luồng là bế tắc và tương tranh.
- ✓Bế tắc(deadlock) xảy ra khi 1 trong 2 luồng cố gắng chiếm khóa truy cập vào một tài nguyên mà luồng còn lại đang chiếm giữ.
- ✓Lúc này không có luồng nào có thể làm được gì hơn.
- ✓Để giải quyết vấn đề, ta sử dụng time-out(thời gian chờ). Nếu chờ quá thời gian chờ đợi được chỉ định thì nó tự động hủy bỏ việc chờ đợi chẳng hạn.
- ✓Tương tranh(race condition) xảy ra khi kết quả của 1 chương trình phụ thuộc vào việc luồng nào tiếp cận đoạn chương trình đó trước.
- ✓Mỗi lần chạy chương trình sẽ cho một kết quả khác nhau không dự đoán được.
- ✓Để giải quyết tương tranh, ta sử dụng các phương thức của lớp Interlocked.

Khuyến nghị chung

- ✓Không sử dụng Thread.Abort() để kết thúc các luồng khác. Gọi phương thức này trên một luồng nào đó sẽ văng ngoại lệ trên luồng đó mà không rõ thread đó đã đạt đến điểm nào trong quá trình xử lý.
- ✓Không sử dụng Thread.Suspend, Thread.Resume để đồng bộ hóa hoạt động đa luồng. Hãy sử dụng Mutex, Monitor, ManualResetEvent, AutoResetEvent.
- ✓Không kiểm soát việc thực thi của luồng phụ từ chương trình chính. Thay vào đó hãy thiết kế chương trình sao cho luồng phụ có thể phản hồi với việc chờ đợi tới khi công việc khả dụng, sau đó xử lý và thông báo cho phần chương trình khác khi nó kết thúc. Nếu thread của bạn không bị phong tỏa, cân nhắc sử dụng luồng thread pool. Monitor.PulseAll sẽ hữu ích nếu luồng phụ bị phong tỏa.
- ✓Không sử dụng các kiểu như các đối tượng khóa. Tức là tránh dùng lock(typeof(X)), Monitor.Enter với kiểu của các đối tượng. Với một kiểu cho trước, chỉ có 1 đối tượng System.Type trong mỗi miền ứng dụng. Nếu kiểu đó giữ khóa công khai thì có thể dẫn tới deadlock.

Khuyến nghị chung

- ✓Thận trọng khi sử dụng lock trên một đối tượng. Nếu mã khác trong ứng dụng, bên ngoài kiểu kiểu đó giữ khóa đối tượng, deadlock có thể xảy ra.
- ✓Hãy chắc chắn rằng một thread đi vào một monitor cũng rời monitor đó ngay cả khi ngoại lệ xảy ra. Câu lệnh lock hỗ trợ điều này tự động. Bổ sung khối finally để Monitor.Exit chắc chắn được gọi. Nếu không chắc chắn phương thức Exit được gọi, hãy sử dụng Mutex để tự động hóa việc giải phóng khi luồng kết thúc.
- ✓Hãy sử dụng đa luồng cho các công việc cần nhiều tài nguyên khác nhau, tránh sử dụng đa luồng cho 1 tài nguyên đơn lẻ.
- ✓Cân nhắc sử dụng các phương thức của lớp Interlocked cho các thay đổi trạng thái đơn giản thay vì sử dụng câu lệnh lock để tăng hiệu năng.

Khuyến nghị cho các lớp thư viện

- ✓Tránh đồng bộ hóa nếu có thể, đặc biệt các đoạn mã hay được sử dụng để tránh việc ảnh hưởng hiệu năng.
- ✓Tạo dữ liệu static an toàn luồng theo mặc định.
- ✓Không tạo các đối tượng instance an toàn luồng theo mặc định. Việc thêm khóa vào sẽ giảm hiệu năng, tăng khả năng deadlock. Với ứng dụng thông thường không cần phải an toàn luồng vì chỉ có 1 luồng truy cập mã người dùng tại 1 thời điểm.
- ✓Tránh cung cấp các phương thức static trong đó làm thay đổi các trường static. Với đa luồng, bất kỳ luồng nào cũng có thể truy cập vào phương thức static tại 1 thời điểm làm tăng khả năng lỗi luồng. Cân nhắc các mẫu thiết kế đóng gói dữ liệu vào trong đối tượng không chia sẻ chung. Hơn nữa nếu dữ liệu static đồng bộ hóa, các lời gọi thay đổi trạng thái có thể dẫn tới deadlock hoặc đồng bộ dư thừa, ảnh hưởng xấu tới hiệu năng.

Nội dung tiếp theo

Lập trình giao diện