# MÔN: HỆ ĐIỀU HÀNH

## Bài thực hành số 5.2: Viết phần mềm giải quyết deadlock

### I. Mục tiêu

- Giúp SV củng cố kiến thức về deadlock và hiện trạng giải quyết deadlock trên Windows.
- Giúp SV làm quen với việc dùng class Mutex của namespace System.Threadings, sự hiện thực semaphore nhị phân của môi trường .NET, để loại trừ tương hỗ giữa các thread khi chúng cùng truy xuất vào tài nguyên dùng chung.
- Giúp SV biết cách dùng tác vụ WaitOne() của class Mutex để giải quyết deadlock (dùng chiến lược giết 1 hay nhiều thread tham gia deadlock).

### II. Nội dung

- Tìm cách giải quyết deadlock của các thread khi chùng cùng chờ lẫn nhau trong việc hiển thị icon của mình lên các cell dùng chung.
- Dùng tác vụ WaitOne(thời gian chờ hữu hạn) class Mutex để giải quyết deadlock.

### III. Chuẩn đầu ra

- Sinh viên nắm vững và sử dụng thành thạo class Thread để quản lý thread.
- Sinh viên nắm vững vấn đề deadlock giữa các thread khi chúng cùng chờ lẫn nhau.
- Sinh viên nắm vững và sử dụng thành thạo class Mutex để loại trừ tương hỗ giữa các thread khi chúng cùng truy xuất vào tài nguyên dùng chung, để giải quyết deadlock bằng cách giết 1 hay nhiều thread tham gia deadlock.

### IV. Qui trình

1. Chạy VS .Net, mở lại Project 5.1 vừa thực hiện và hiệu chỉnh lại hàm Running của class Form1 như sau :

```
//định nghĩa hàm mà mỗi thread sẽ chạy
void Running(object obj)
{
   //ép kiểu tham số về MyThread theo yêu cầu xử lý
   MyThread p = (MyThread)obj;
   //tạo đối tượng vẽ
   Graphics g = this.CreateGraphics();
   //tạo chổi màu đen để xóa cell cũ
   Brush brush = new SolidBrush(Color.FromArgb(0, 0, 0));
   //xin khóa truy xuất cell (x1,y1)
   mutList[p.Pos.Y, p.Pos.X].WaitOne();
   int x1, y1;
   int x2, y2;
   int x, y;
   bool kq=true;
   try
   {
      while (p.start)
      { //lặp trong khi chưa có yêu cầu kết thúc
         //xác định tọa độ hiện hành của thread
         x1 = p.Pos.X; y1 = p.Pos.Y;
         //hiển thị logo của thread ở (x1,y1)
         g.DrawImage(p.Pic, xCell * x1, yCell * y1);
         Color c = p.Pic.GetPixel(1,1);
```

```
            int yR, yG, yB;
            if (c.R > 128) yR = 0; else yR = 255;
            if (c.G > 128) yG = 0; else yG = 255;
            if (c.B > 128) yB = 0; else yB = 255;
            Pen pen = new Pen(Color.FromArgb(yR, yG, yB), 2);
            if (p.tx >= 0 && p.ty >= 0) { //hiện mũi tên góc dưới phải
               x = xCell * x1 + xCell - 2;
               y = yCell * y1 + yCell - 2;
               g.DrawLine(pen, x, y, x - 10, y);
               g.DrawLine(pen, x, y, x, y-10);
            } else if (p.tx >= 0 && p.ty < 0) { //hiện mũi tên góc trên phải
               x = xCell * x1 + xCell - 2;
               y = yCell * y1  + 2;
               g.DrawLine(pen, x, y, x - 10, y);
               g.DrawLine(pen, x, y, x, y + 10);
            } else if (p.tx < 0 && p.ty >= 0) { //hiện mũi tên góc dưới trái
               x = xCell * x1 + 2;
               y = yCell * y1 + yCell - 2;
               g.DrawLine(pen, x, y, x + 10, y);
               g.DrawLine(pen, x, y, x, y - 10);
            } else {//hiện mũi tên góc trên trái
               x = xCell * x1 + 2;
               y = yCell * y1 + 2;
               g.DrawLine(pen, x, y, x + 10, y);
               g.DrawLine(pen, x, y, x, y + 10);
            }
            //giả lập thực hiện công việc của thread tốn 500ms
            MySleep(500);
            //xác định vị trí mới của thread
            p.HieuchinhVitri();
            x2 = p.Pos.X; y2 = p.Pos.Y;
            //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
            g.FillRectangle(brush, xCell * x1, yCell * y1, xCell, yCell);
            //trả cell (x1,y1) cho các thread khác truy xuất
            mutList[y1, x1].ReleaseMutex();
         }
      }
      catch (Exception e) { p.stop = true; p.t.Abort(); }
      //dọn dẹp thread trước khi ngừng
      x1 = p.Pos.X; y1 = p.Pos.Y;
      g.FillRectangle(brush, xCell * x1, yCell * y1, xCell, yCell);
      //trả cell (x1,y1) cho các thread khác truy xuất
      mutList[y1, x1].ReleaseMutex();
      // dừng Thread
      p.stop = true;
```

```
        p.t.Abort();
    }
```

2. Chọn menu Debug.Start Debugging để dịch và chạy thử ứng dụng.
3. Khi Form chương trình hiển thị, hãy thực hiện gỏ phím qui định như sau để quản lý các thread :
   - 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.

   Khi số thread chạy tương đối nhiều, hãy quan sát hiện tượng deadlock, nghĩa là các thread có thể dừng chờ lẫn nhau và không có thread nào chạy tiếp được. Để giải quyết deadlock giữa 1 nhóm thread dừng chờ lẫn nhau, ta có thể xác định 1 thread trong nhóm và xóa nó (bằng cách dùng tổ hợp phím Ctrl-Shift-X), tiếp tục giải quyết deadlock nếu còn cho đến khi không còn deadlock nữa (có thể nhiều thread sẽ bị giết). Thread nào bị giết mà muốn chạy lại, ta dùng phím từ A-Z để kích hoạt lại nó.