# CS4231
# Parallel and Distributed Algorithms

Solution for Homework 2

Instructor: Haifeng YU

# Homework Assignment

- Page 51:
  - Problem 3.3, 3.4, 3.6
  - All using monitors

# Reader-Writer Problem

Overall idea: Maintain an explicit queue. Each (implicit) monitor queue has at most one element.

```
Vector queue; // shared among all threads
int numReader; int numWriter;

// code for writer entry
Writer w = new Writer(myname);
synchronized (queue) {
    if (numReader > 0 || numWriter > 0) {
        w.okToGo = false;
        queue.add(w);
    } else {
        w.okToGo = true;
        numWriter++;
    }
}
synchronized (w) { if (!w.okToGo) w.wait(); }
```

# Reader-Writer Problem

```
// code for writer exit
synchronized (queue) {
    numWriter--;
    if (queue is not empty) {
        remove a single writer or a batch of readers from queue
        for each request removed do {
            numberWriter++ or numberReader++;
            synchronized (request) {
                request.okToGo = true;
                request.notify();
            }
        }
    }
}
```

# Reader-Writer Problem

```
// code for reader entry
Reader r = new Reader(myname);
synchronized (queue) {
    if ((numWriter > 0) || !queue.isEmpty()) {
        r.okToGo = false;
        queue.add(r);
    } else {
        r.okToGo = true;
        numReader++;
    }
}
synchroized (r) { if (!r.okToGo) r.wait(); }
```

# Reader-Writer Problem

```
// code for reader exit
synchronized (queue) {
    numReader--;
    if (numReader > 0) exit(); // I am not the last reader
    if (queue is not empty) {
        remove a single writer or a batch of readers from queue
        for each request removed do {
            numWriter++ or numReader++;
            synchronized (request) {
                request.okToGo = true;
                request.notify();
            }
        }
    }
}
```

# Customer

```
Vector customQueue; // shared data among all threads

synchronized (numberChair) {
    if (numberChair > 0) numberChair--;
    else return; // leave // this releases monitor lock
}

// middle part (see next slide)

synchronized (numChair) {
    numberChair++;
}
```

# Customer – Middle part

```
// middle part
Customer myself = new Customer(myname);
myself.done = false;
synchronized (customQueue) {
    customQueue.add(myself); // can also simulate chair here
    customQueue.notify();
}

synchronized (myself) {
    if (!myself.done) myself.wait(); // no need to use while
}
```

# Barber

```
while (true) {
    Customer current;
    synchronized (customerQueue) {
        if (customerQueue.isEmpty()) customerQueue.wait();
        // no need to use "while" here because only one barber
        current = customerQueue.removeFirst();
    }

    // hair cut the current customer

    synchronized (current) {
        current.done = true;
        // this flag helps to avoid needing nested monitor
        current.notify();
    }
}
```

# The PQR Problem

```
int numP, numQ, numR; Object obj;
thread1:
    while (true) {
        print P;
        synchronized (obj) { numP++; obj.notify(); }
    }
thread2: similar as above
thread3:
    while (true) {
        synchronized (obj) {
            while (numR == numberP + numberQ) obj.wait();
        }
        print R;
        synchronized (obj) { numR++; }
    }
```