CSE Lab2 Synchronization

By 彭超 黄子豪 俞哲轩

Description

信号量和锁常用来在并发过程中做线程和资源的管理。本次Lab中，你将需要查阅资料，使用代码实现一个多线程互斥的锁（Part A），并利用该锁来解决“哲学家吃饭问题”（Part B）。

DDL

Deadline：**2021.11.26, 23:59**
提交方式：将开发文档（包含Part A和Part B的运行结果截图）和代码文件打包为 学号-姓名-lab2.zip 上传至elearning。
注意：抄袭零分

Part A

使用代码实现锁。

要求：

1. 无死锁、无饥饿。
2. 不得使用依赖硬件指令（TAS（Test And Set）、 CAS（Compare And Set）等）实现的语言特性，如synchronized、AtomicInteger、ReentrantLock、ConcurrentHashMap。即编程语言中原有的线程安全的特性均不能使用。
3. 可以保证任意多个线程互斥。
4. 锁的接口如下，你需要实现MyLock接口：

```
interface MyLock{
    void lock();
    void unLock();
}
```

5. 你可以通过以下测试来测试你的锁是否有效（互斥性）：

```
public class test {
    public static void main(String[] args){
        MyLock lock;
        //TODO: initialize the lock

        testA1(lock);
    }

    static int cnt = 0;
    public static void testA1(MyLock lock){
        System.out.println("Test A start");

        int threadNumber = 5;
```

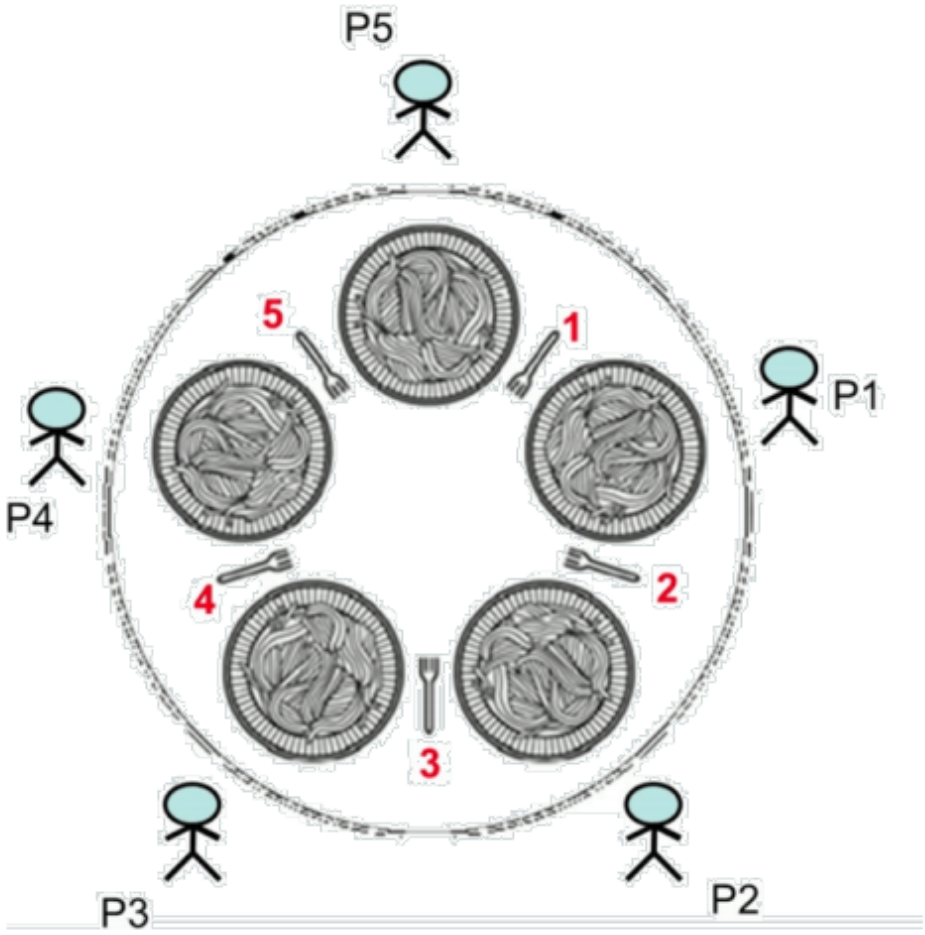
```
        final CountDownLatch cdl = new CountDownLatch(threadNumber);//参数为
线程个数

        Thread[] threads = new Thread[threadNumber];
        for (int i = 0; i < threadNumber; i++){
            threads[i] = new Thread(() -> {
                lock.lock();
                int tmp = cnt;
                try {
                    Thread.sleep(50);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                cnt = tmp + 1;
                lock.unLock();
                cdl.countDown();//此方法是CountDownLatch的线程数-1
            });
        }
        for (int i = 0; i < threadNumber; i++){
            threads[i].start();
        }

        //线程启动后调用countDownLatch方法
        try{
            cdl.await();//需要捕获异常，当其中线程数为0时这里才会继续运行
            String res = cnt == 5 ? "Test A passed" : "Test A failed,cnt
should be 5";
            System.out.println(res);
        }catch(InterruptedException e){
            e.printStackTrace();
        }
    }
}
```

Part B

哲学家吃饭问题：



5个哲学家围坐在一起吃饭（eating）和思考（thinking）。有5只叉子可以供他们共享，每个哲学家需要拿起身旁的2只叉子进行吃饭，吃完之后会放下叉子，进行思考，然后叉子会被别的哲学家使用。

使用Java语言模拟上述场景，保证每个哲学家都能吃到饭，同时避免死锁。

要求：

1. 主要实现Philosopher和Dining两个类，前者用于模拟哲学家的吃饭、思考行为，后者用于进行吃饭场景的模拟；
2. 需要实现Runnable接口以使每个哲学家作为独立的线程运行；
3. 为了保证每只叉子不被多个人拿到，需要为叉子上锁；
4. Philosopher类的框架如下，你需要实现run()方法的细节：

```
public class Philosopher implements Runnable{
    private final Object leftFork;
    private final Object rightFork;
    Philosopher(Object left, Object right){
        this.leftFork = left;
        this.rightFork = right;
    }

    private void doAction(String action) throws InterruptedException{
        System.out.println(Thread.currentThread().getName() + " " +
                action);
        Thread.sleep(((int) (Math.random() * 100)));
    }
    @Override
    public void run(){
```

```
        try {
            while(true){
                doAction(System.nanoTime() + ": Thinking"); // thinking
                // your code
            }

        } catch(InterruptedException e){
            Thread.currentThread().interrupt();
        }
    }
}
```

可参考伪代码：

```
while(true){
    think();
    pick_up_left_fork();
    pick_up_right_fork();
    eat();
    put_down_right_fork();
    put_down_left_fork();
}
```

5. Dining类的框架如下，你需要完成对象的初始化并让每个线程运行起来，以进行场景的模拟：

```
public class Dining{
    public static void main(String[] args) throws Exception {
        Philosopher[] philosophers = new Philosopher[5];
        Object[] forks = new Object[philosophers.length];
        for (int i = 0; i < forks.length; i++) {
            // initialize fork object
        }
        for (int i = 0; i < philosophers.length; i++) {
            // initialize Philosopher object
        }
    }
}
```

6. 示例输出如下：

```
Philosopher 4 88519840870182: Thinking
Philosopher 5 88519840956443: Thinking
Philosopher 3 88519864404195: Picked up left fork
Philosopher 5 88519871990082: Picked up left fork
Philosopher 4 88519874059504: Picked up left fork
Philosopher 5 88519876989405: Picked up right fork - eating
Philosopher 2 88519935045524: Picked up left fork
```

7. 别的地方也可进行修改，合理即可。

注意事项：

1. 本次lab要求使用java语言完成。
2. Part A占比40%，Part B占比50%，代码风格占比10%。
3. Part B可不使用Part A实现的锁，但会酌情扣分（约总评5%）。

4. **严禁抄袭，抄袭零分。**