CSE Lab2 Synchronization

TAs

- 刘林方 19212010051
- 王孟辉 19212010074
- 姜尔玲 17307130291

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，阐述自己对二者较为深入的理解（Exercise 1），并结合程序了解其是如何工作的（Exercise 2）；在此基础上，你还需要完成相应的伪代码和Java代码解决同步问题中经典的“读者-写者”问题（Exercise 3）和“哲学家吃饭”问题（Exercise 4）。

DDL

Deadline：**2020.11.27, 23:59**

提交方式：将答案和代码文件打包为lab2-姓名-学号.zip上传至elearning。

注意：抄袭零分

Exercise 1

思考分析Semaphore和Mutex的异同，举例说明二者适用的场景，在何种情况下可以达到相同的功能，在何种情况下具有各自的独特性，可以从二者考虑问题的方向、解决问题的表现、线程和资源的关系等角度分析，500字以内。

Exercise 2

考虑下述程序：

```
int x = 0, y = 0, z = 0;
sem lock1 = 1, lock2 = 1;
process foo{
    z = z + 2;
    P(lock1);
    x = x + 2;
    P(lock2);
    V(lcok1);
    y = y + 2;
    V(lock2);
}
process bar{
    P(lock2);
```

```
    y = y + 1;
    P(lock1);
    x = x + 1;
    V(lock1);
    V(lock2);
    z = z + 1;
}
```

回答以下问题：

1. 在何种情况下程序会产生死锁?
2. 在死锁状态下，x，y，z的最终值可能是多少?
3. 如果程序正常终止，x，y，z的最终值可能是多少？（提示：对z的赋值操作不是原子性的）

Exercise 3

多个进程共享U个资源，1个进程1次可以获取（request）1个资源，但是可能释放（release）多个。使用信号量做同步，合理进行P，V操作，实现request和release方法的伪代码，确保request和release操作是原子性的。提前声明和初始化你用到的变量。

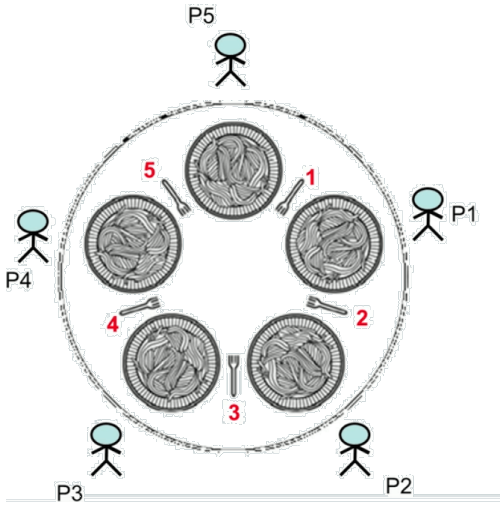
```
int free = U;
# your variables

# <await (free > 0) free = free - 1;>
request(){
    # your code
}

# <free = free + number;>
release(int number){
    # your code
}
```

Exercise 4

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，建议使用*synchronized*关键字；
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。