**中山大学数据科学与计算机学院本科生实验报告**

**（2017学年秋季学期）**

课程名称：嵌入式系统 任课教师：黄凯

|  |  |  |  |
| --- | --- | --- | --- |
| 年级 | **2015级** | 专业（方向） | **软件工程(移动信息工程）** |
| 学号 | **15352008** | 姓名 | **蔡荣裕** |
| 电话 | **13727021990** | Email | [**897389207@qq.com**](mailto:897389207@qq.com) |
| 开始日期 | **2017/9/22** | 完成日期 | **2017/9/21** |

实验题目

使用java编程实现死锁的模拟。

实验目的

1. 了解如何进行java环境的配置，以及如何使用命令行进行编译运行java。
2. 复习死锁发生的4个条件。
3. 分析已有的模拟死锁代码，通过改变参数了解死锁的发生时机。

实验原理

1. **死锁发生的必要条件**
   1. **互斥条件**：一个资源每次只能被一个进程使用
   2. **请求与保持条件**：一个进程因请求资源而阻塞时，对已获得的资源保持不放
   3. **不剥夺条件**:进程已获得的资源，在末使用完之前，不能强行剥夺
   4. **循环等待条件**:若干进程之间形成一种头尾相接的循环等待资源关系
2. **Java关键字 synchronized**

* 当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码。
* 当一个线程访问object的一个synchronized同步代码块或同步方法时，其他线程对object中所有其它synchronized同步代码块或同步方法的访问将被阻塞。

代码实现

class A{

synchronized void methodA(B b){

b.last();

}

synchronized void last(){

System.out.println("Inside A.last()");

}

}

class B{

synchronized void methodB(A a){

a.last();

}

synchronized void last(){

System.out.println("Inside B.last()");

}

}

class Deadlock **implements** Runnable{

A a=**new** A();

B b=**new** B();

Deadlock(){

Thread t = **new** Thread(**this**);

int count = 14500;

t.start();

**while**(count-->0);

a.methodA(b);

}

public void run(){

b.methodB(a);

}

public static void main(String args[]){

**new** Deadlock();

}

}

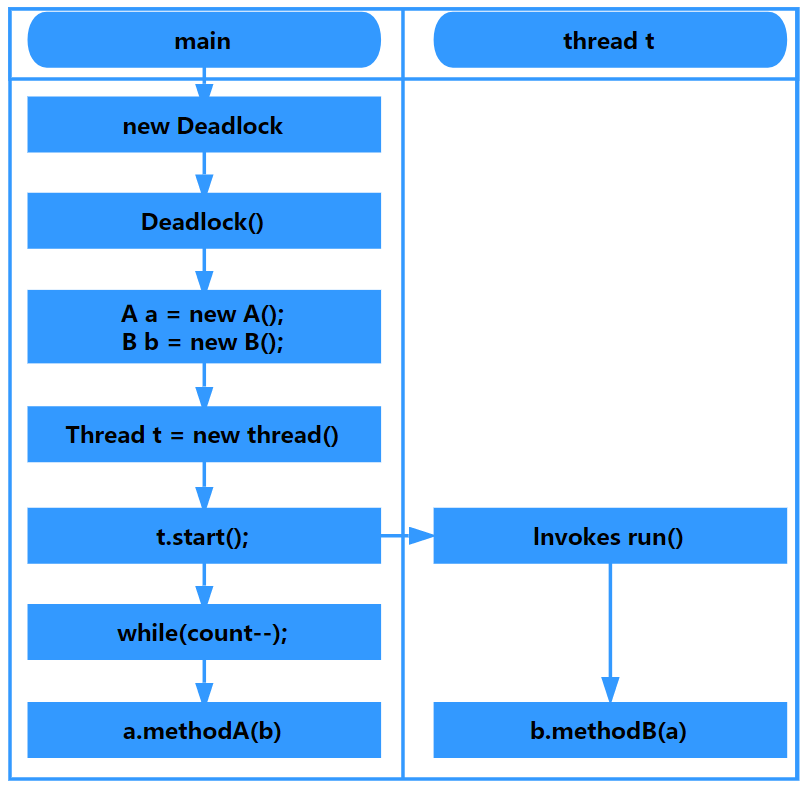
结果及分析

**改变不同Count的测试结果如下：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Count值 | Count=1000 | Count=8000 | Count=10000 | Count=14500 |
| 死锁发生时间 | 1000次内不发生死锁 | 1000次内基本不发生死锁 | 100+后发生死锁 | 大概率个位数内发生死锁 |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Count值 | Count=15000 | Count=20000 | Count=30000 | Count=50000 |
| 死锁发生时间 | 20以内大概率发生死锁 | 300次左右死锁 | 1000次内较少发生死锁 | 1000次以内不发生死锁 |
|  |  |  |  |

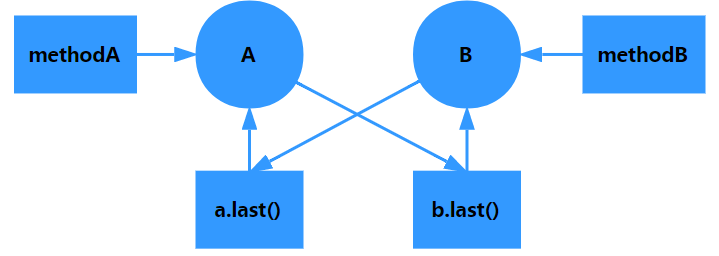
**代码运行流程如下：**



**死锁原因分析：**

1. 互斥条件：在两个class A和B中我们看到其两个函数都被synchronized修饰，此时只有唯一个一线程能够访问一个类中的这两个函数。所以此处构成了互斥条件。
2. 请求与保持条件：代码中如果请求不到资源，那么自身资源不会释放
3. 不剥夺条件：代码中不会杀死线程释放资源。
4. 循环等待：这个条件代码中的类相互访问造成。

代码中实际的死锁情况如图：

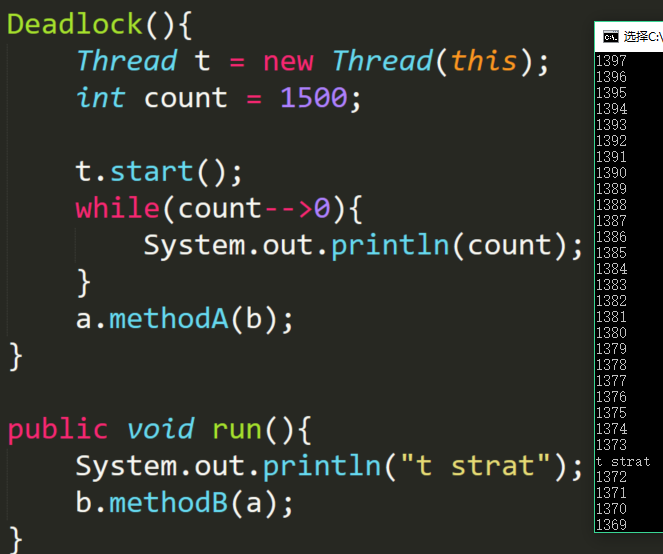


因为class中的method和last都是用synchronized修饰所以可以认为他们一体的资源

线程A(线程main)调用class A中的methodA，线程B(线程t)调用class B中的methodB，此时这两个函数分别要去访问b.last()和a.last()。此时就构成了死锁，因为A没有获得b.last()所以没法释放classA也就是a.last(),此时B也因为没有获得a.last()所以无法释放classB也就是b.last()。此时就造成了死锁。

这段代码想要造成死锁的关键就在于如何让main线程访问classA的同时t线程访问classB，之后才能造成上图发生死锁情况。

我们修改一下代码如下图：



我们可以发现实际中t开始运行的时间和t.start()之间还是有一定的时间差的。

那么代码中调整Count的大小就是在调整时间差，造成他们的同时访问的情况，至于.bat文件是为了多次运行，如果真正把握好时间差其实一次就能死锁，但是为什么需要.bat，就是因为whlie(count--)和t开始运行的时间之间并不是固定的，他们受到当前cpu负载之类的影响，所以需要多次运行实现让其刚好符合，也就是要寻找while(count--)和t开始运行的时间的平衡点。

实验感想

这次实验主要就就是复习之前OS中学到死锁条件，只不过是用了java来进行模拟，之前电脑上也就有java环境，所以实验过程中没有什么困难，就是之前没有学过java，所以分析java代码比较麻烦点，还要去看看java一些代码的意思。