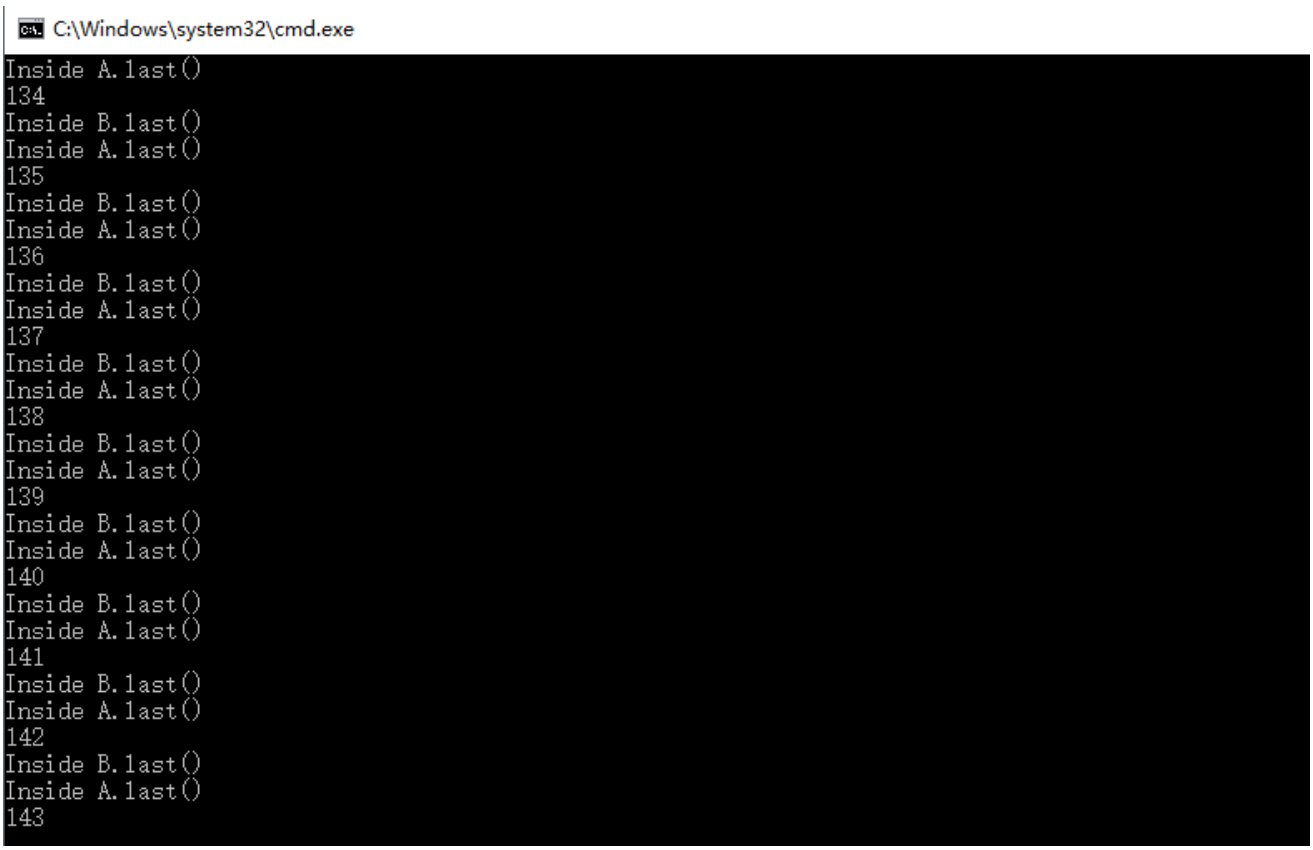
嵌入式系统导论实验报告

姓名	学号	班级	电话	邮箱
赵佐骑	15352434	1519	18686667712	1398772969@qq.com

1.实验题目

Lab1:死锁

2.实验结果



1. 产生死锁的4个必要条件：

- 互斥条件：一个资源每次只能被一个进程使用
- 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
- 非抢占条件: 资源不能被抢占，即资源只能在进程完成后自动释放。
- 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. 对程序产生死锁进行解释

关键在于理解代码中`synchronized`这个关键字，我在网上找到的一个比较通俗易懂的解释如下：

> 1. 某个对象实例内，**synchronized aMethod(){}**可以防止多个线程同时访问这个对象的**synchronized**方法（如果一个对象有多个**synchronized**方法，只要一个线程访问了其中的一个**synchronized**方法，其它线程不能同时访问这个对象中任何一个**synchronized**方法）。
>
> 2. 这时，不同的对象实例的 **synchronized**方法是不相干扰的。也就是说，其它线程照样可以同时访问相同类的另一个对象实例中的**synchronized**方法

大概流程如下，下面是相关代码：

```
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=20000;
    t.start();
    while(count-->0);
    a.methodA(b);
}
public void run()
{
    b.methodB(a);
}
public static void main(String args[])
{
    new Deadlock();
}
}
```

```
@echo off
:start
set /a var+=1
echo %var%
java Deadlock
if %var% leq 1000 GOTO start
pause
```

- 首先批处理函数的意思是定义一个变量**var**来记录执行**Deadlock.java**函数的次数并输出出来，最多不能超过**1000**次。在**Deadlock.java**函数中，一共有两个线程，一个是新建立的线程**t**，将执行`b.last()`；一个是用**runnable**定义的在后台执行的线程，用来执行`a.last()`。定义变量`count`是为了产生延迟。
- 由于是通过时间片轮转的方法来调度，有可能在某一时刻在执行`b.last()`的时候，它需要调用`class B`里面的`last()`函数来输出，而这时刚好轮转到调用`a.last()`，这样由于`synchronized`的影响，线程**t**就无法调用`class B`里面的`last()`函数，而`a.last()`则需要调用`class A`里面的`last()`函数来输出，同样，无法调用，必须等到`class A`中的`b.last()`释放之后才能执行，这样一来，就形成了互锁，也就形成了死锁，他们都在等待对方释放。

3.实验心得

在本次实验中，由于第一次接触**java**语言相关的内容，一开始在安装环境方面遇到了一些问题，后来得以解决，在实验内容方面，虽然对**java**的相关语法不是太熟，但本次实验主要是理解产生死锁的条件，弄清楚`synchronized`这个关键字的作用。在产生死锁的方面貌似没有遇到其他人产生不了死锁的现象，不过我也试着通过改变**count**的值来观察现象，发现它主要是决定是先执行哪一个进程的，同时这也是个概率事件，每次产生死锁的次数都不一样。