

## 21307077 凌国明 嵌入式系统作业四

```
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();
    }
}
```

在三个不同的平台下，运行以上代码 100 次

```
for((c=1; c<=100; c++))
do
    echo "$c times"
    java Deadlock
done
```

## 代码分析

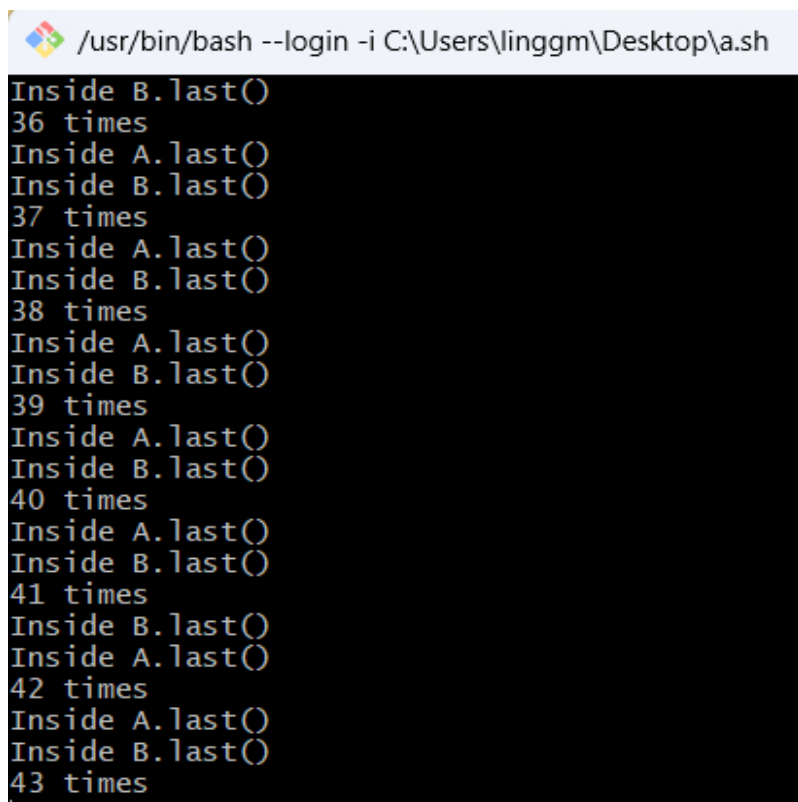
在 Deadlock 类的构造函数中，创建了一个新线程，用于执行 run() 方法。

在新线程中，调用 b.methodB(a)。由于 methodB 是 synchronized 方法，这个线程会持有 b 的锁，并且尝试获取 a 的锁以执行 a.last()。

同时，主线程在 while 循环之后调用 a.methodA(b)。由于 methodA 也是同步方法，主线程会持有 a 的锁，并尝试获取 b 的锁以执行 b.last()。

此时，如果两个线程同时运行，就会发生死锁。主线程持有 a 的锁，等待 b 的锁；而新线程持有 b 的锁，等待 a 的锁。由于都在等待对方释放锁，所以都无法继续执行，从而造成死锁。

## Laptop + Windows 运行结果



```
/usr/bin/bash --login -i C:\Users\linggm\Desktop\a.sh
Inside B.last()
36 times
Inside A.last()
Inside B.last()
37 times
Inside A.last()
Inside B.last()
38 times
Inside A.last()
Inside B.last()
39 times
Inside A.last()
Inside B.last()
40 times
Inside A.last()
Inside B.last()
41 times
Inside B.last()
Inside A.last()
42 times
Inside A.last()
Inside B.last()
43 times
```

## Desktop + Windows 运行结果

```
1 times
Inside A.last()
Inside B.last()
2 times
Inside A.last()
Inside B.last()
3 times
Inside A.last()
Inside B.last()
4 times
Inside A.last()
Inside B.last()
5 times
Inside A.last()
Inside B.last()
6 times
Inside A.last()
Inside B.last()
7 times
|
```

## Laptop + Linux 运行结果

```
linggm@linggm-virtual-
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Inside A.last()
Inside B.last()
994 times
Inside A.last()
Inside B.last()
995 times
Inside A.last()
Inside B.last()
996 times
Inside A.last()
Inside B.last()
997 times
Inside A.last()
Inside B.last()
998 times
Inside A.last()
Inside B.last()
999 times
Inside B.last()
Inside A.last()
1000 times
Inside A.last()
Inside B.last()
```

# 比较 Windows 和 Linux

Windows 运行42 次出现死锁，Linux 运行 1000 次都没有死锁。初步分析这是由于操作系统的线程调度和管理策略不同导致的

1. 线程调度策略的差异：Windows 和 Linux 操作系统在线程调度上有不同的策略和优化。这些差异可能导致线程在不同的操作系统上以不同的方式和顺序执行。
2. 锁的竞争条件：死锁的发生依赖于线程获取锁的顺序。如果两个线程以不同的顺序获取锁，就可能导致死锁。由于操作系统对线程调度的处理不同，可能在一个系统上更容易出现死锁的条件。
3. 优化和性能差异：不同操作系统的 JVM 实现可能有不同的性能优化和行为特征。这可能包括锁的获取和释放机制、线程管理、内存分配和回收等方面的差异。

还有一种可能的变量是核心数，这里我是在虚拟机上运行的 Linux，虚拟机分配的核心数可能比较少，多核处理器并行执行的线程数少，从而降低了发生死锁的风险

# 比较 Laptop 和 Desktop

Laptop 和 Desktop 的 OS 都是 win11，但 Desktop 6 次就出现死锁，初步分析这与硬件配置和运行时环境相关。以下是进一步的与死锁相关的因素分析：

1. 处理器核心数：多核处理器能够并行执行更多线程，这可能增加线程间锁竞争的机会，从而可能增加死锁的风险。
2. 线程调度和管理：操作系统和 JVM 的线程调度策略可能在不同的硬件配置上有所不同，这可能影响线程间的交互和同步。
3. 系统负载：系统上运行的其他进程和应用程序的数量和性质可能影响线程调度，从而影响死锁发生的可能性。

Desktop 拥有更强大的处理器和更多的核心，导致了可以实现更多的并发线程执行，从而可能增加死锁的风险。因此，Desktop 6 次发生死锁，Laptop 42 次后发生