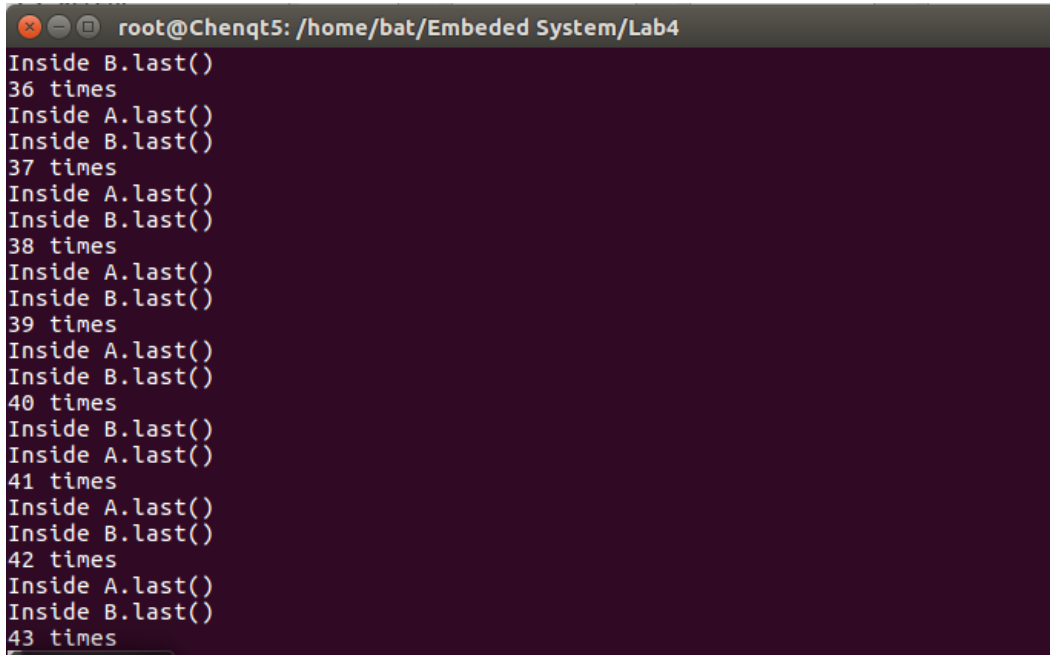


DEADLOCK

实验过程

1. 首先编写Deadlock的代码
2. 使用 `javac Deadlock.java` 进行编译
3. 然后创建 `Deadlock.sh` 文件，文件中将 `Deadlock` 执行100次，然后在terminal中 `bash Deadlock.sh` 执行文件
4. 观察结果

实验结果



```
root@Chenqt5: /home/bat/Embeded System/Lab4
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 B.last()
Inside A.last()
41 times
Inside A.last()
Inside B.last()
42 times
Inside A.last()
Inside B.last()
43 times
```

可以看到在第43次的时候出现了死锁的情况

死锁的必要条件

1. 互斥：至少有一个资源必须处于非共享模式，即一次只能有一个进程使用，如果另一个进程申请该资源，那么申请进程必须等待到该资源被释放为止
 - 占有并等待：一个进程必须占有至少一个资源，并等待另一个资源。而该资源为其他进程所占有
 - 非抢占：资源不能被抢占，即资源只能在进程完成任务之后自动释放
 - 循环等待：有一组等待进程 $\{P_0, P_1, \dots, P_n\}$ ， P_0 等待的资源为 P_1 所占有， P_1 等待的资源为 P_2 所占有， \dots ， P_{n-1} 等待的资源为 P_n 所占有， P_n 等待的资源为 P_0 所占有

以上四个条件必须同时满足时才会出现死锁

程序死锁分析

- 首先我们分析 `Deadlock.java` 文件中的代码，这份代码中首先定义了两个类，然后每个类都拥有两个成员函数，并规定这些成员函数都是可以同步执行的，这个同步执行为死锁奠定一个很关键的基础
- 然后 `Deadlock.java` 又定义了一个 `Deadlock` 类，这个类首先创建 `A` 类实例 `a`，创建 `B` 类实例 `b`，然后在这个类的构造函数中，创建一个线程，然后把这个线程插入调度队列，轮到这个线程执行时就是 `run` 函数，这个函数执行了 `b` 对象的 `methodB()` 函数，以 `a` 为参数
- 最关键的是，这个线程插入调度队列之后，过了20000个时间单位，执行 `a` 对象的 `methodA()` 函数，以 `b` 为对象。如果此时刚好轮到上面定义的线程执行，这个线程要使用 `a` 对象，然而 `a` 对象也同时也在执行，需要 `b` 对象，而 `b` 对象也在运行。这样就造成了一个死锁。

