

操作系统原理实验报告

**实验名称:** 实验六 并发与锁机制

**授课教师：** 张青

**学生姓名:**

**学生学号:** 2233

1. **实验要求**





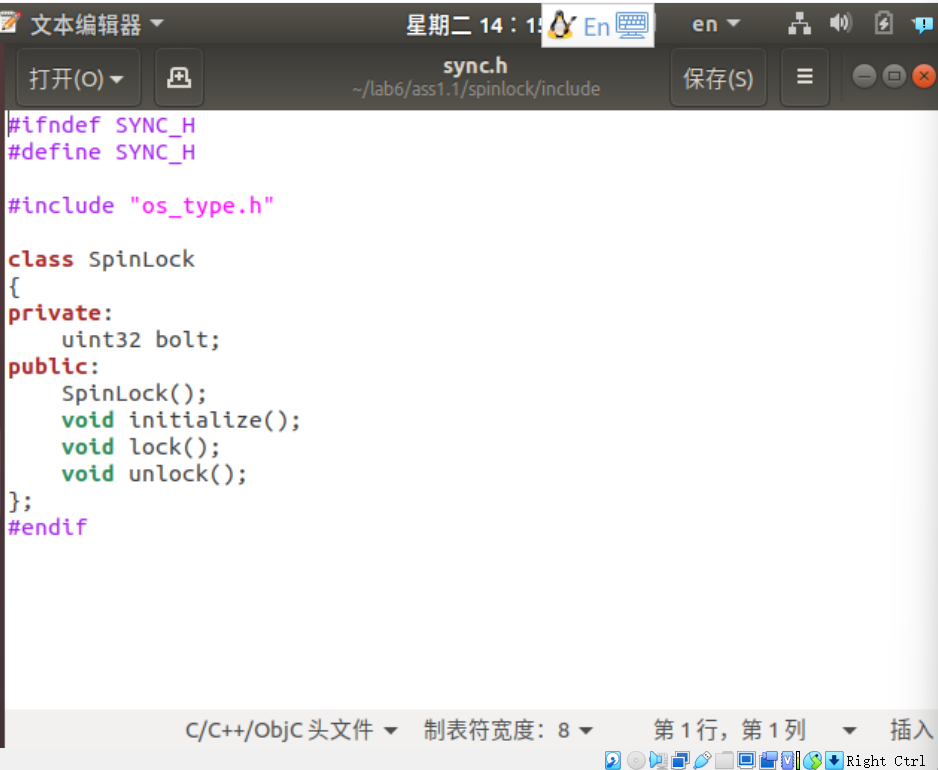
1. **实验过程**

（一）Assignment 1

1.1）复现自旋锁和信号量

此处我们解决教程中提及的消失的芝士汉堡问题。

首先考虑自旋锁实现互斥，自旋锁的基本思想是定义一个共享变量bolt，bolt会被初始化为0。在线程进入临界区之前，即访问共享变量之前，都需要去检查bolt是否为0。如果bolt为0，那么这个线程就会将bolt设置为1，然后进入临界区。待线程离开临界区后，线程会将bolt设置为0。如果线程在检查bolt时，发现bolt为1，说明有其他线程在临界区中。此时这个线程就会一直在循环检查bolt的值（类似陀螺在原地旋转，所以被称为自旋），直到bolt为0，然后进入临界区。根据自旋锁的基本思想，我们可以先定义一个自旋锁的类SpinLock，并依序完善类中各方法：

****

Lock是线程请求进入临界区的函数，unlock函数里放置线程离开临界区时我们对bolt的相应操作。

注意，在上锁过程中，bolt和key交换的指令asm\_atomic\_exchange是一个“原子”指令，用于保证两个值在交换过程中不会被中断，使用汇编代码实现。

现在我们完成了自旋锁的实现，只需要在母亲制作芝士汉堡会为其加上锁，在晾完衣服并检查芝士汉堡后，才释放锁。就可以避免儿子回家将做好的芝士汉堡全部偷吃了。

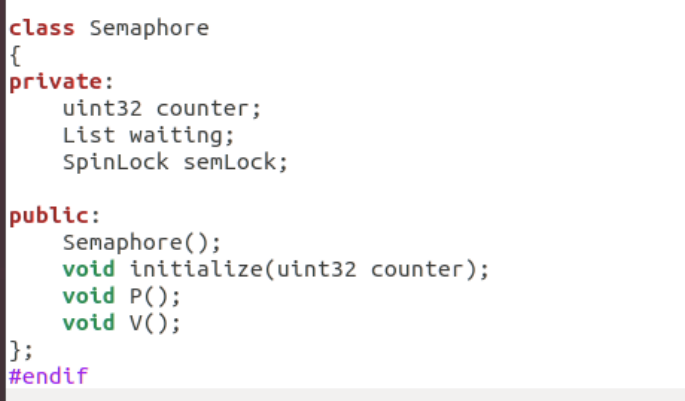
接下来我们考虑信号量实现互斥：

我们首先定义一个非负整数counter表示临界资源的个数。

当线程需要申请临界资源时，线程执行P操作检查counter的数量， counter大于0，表示临界资源有剩余，那么就将一个临界资源分配给请求的线程；如果counter等于0，表示没有临界资源剩余，那么这个线程会被阻塞，然后挂载到信号量的阻塞队列当中。当线程释放临界资源时，执行V操作。V操作会使counter的数量递增1，然后V操作会检查信号量内部的阻塞队列是否有线程，如果有，那么就将其唤醒。

从上面的描述可以看到，counter和阻塞队列是共享变量，需要实现互斥访问。目前，我们实现互斥工具只有SpinLock，因此，我们实际上使用SpinLock去实现信号量。

依旧先定义一个信号量类：



依据概念实现各函数，并使用信号量来为芝士汉堡上锁。

1.2）更换原子指令来实现自旋锁：

此处选用了bts指令来实现，首先我们知道bts，位测试并置位，主要用于测试和设置某个位的值。操作时会首先将指定位的值储存到CF中，然后再将位串中的指定位设置为1，所以我们此处利用CF位值来停止外界循环，利用置位改变bolt的值，从而实现自旋锁功能。

（二）信号量解决生产者—消费者问题

提出一个生产者-消费者问题，此处采用搜索所得的宴会蛋糕服务问题：某位商人在餐厅举行生日宴会。 餐桌上有一个点心盘，最多可以容纳 5 块蛋糕，每个人（服务生或者来宾）每次只能放入/拿出 1 块蛋糕。服务生 A 负责向点心盘中放入抹茶蛋糕，服务生 B 负责向点心盘中放入芒果蛋糕。生日宴会有 6位男性来宾， 4 位女性来宾，男性来宾等待享用芒果蛋糕， 女性来宾等待享用抹茶蛋糕。如果盘中没有对应口味的蛋糕且点心盘没有放满，来宾会给相应的服务生发送一个请求服务信号，服务生受到信号会放入 1 块蛋糕。

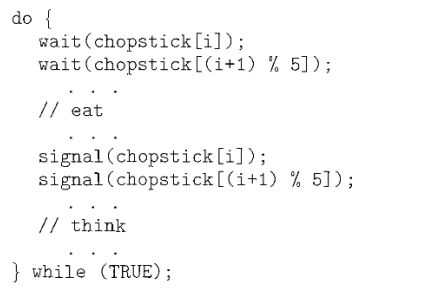
首先基于场景编写线程，呈现出此时会出现的冲突的情况，然后结合教程中对于信号量的示例代码，结合场景分析需要分配的资源，使用信号量表示，即可使此生产者消费者问题得到较好解决。

（三）线程调度切换

3.1）线程模拟哲学家就餐场景

结合问题背景：假设有 5 个哲学家，他们的生活只是思考和吃饭。这些哲学家共用一个圆桌，每位都有一把椅子。在桌子中央有一碗米饭，在桌子上放着 5 根筷子。当一位哲学家思考时，他与其他同事不交流。时而，他会感到饥饿，并试图拿起与他相近的两根筷子（筷子在他和他的左或右邻居之间）。一个哲学家一次只能拿起一根筷子。显然，他不能从其他哲学家手里拿走筷子。当一个饥饿的哲学家同时拥有两根筷子时，他就能吃。在吃完后，他会放下两根筷子，并开始思考。

其中筷子是待分配的资源，我们暂时使用一个信号量表示，一个哲学家有进食和思考两种状态，进食需要拿到其左右的筷子，我们给每个哲学家及筷子编号，很容易发现期间的对应关系，chopstick[i]是第i个哲学家左手边的筷子,chopstick[（i+1）%5]为其右手边的筷子，参考课本我们可来实现每个哲学家线程：

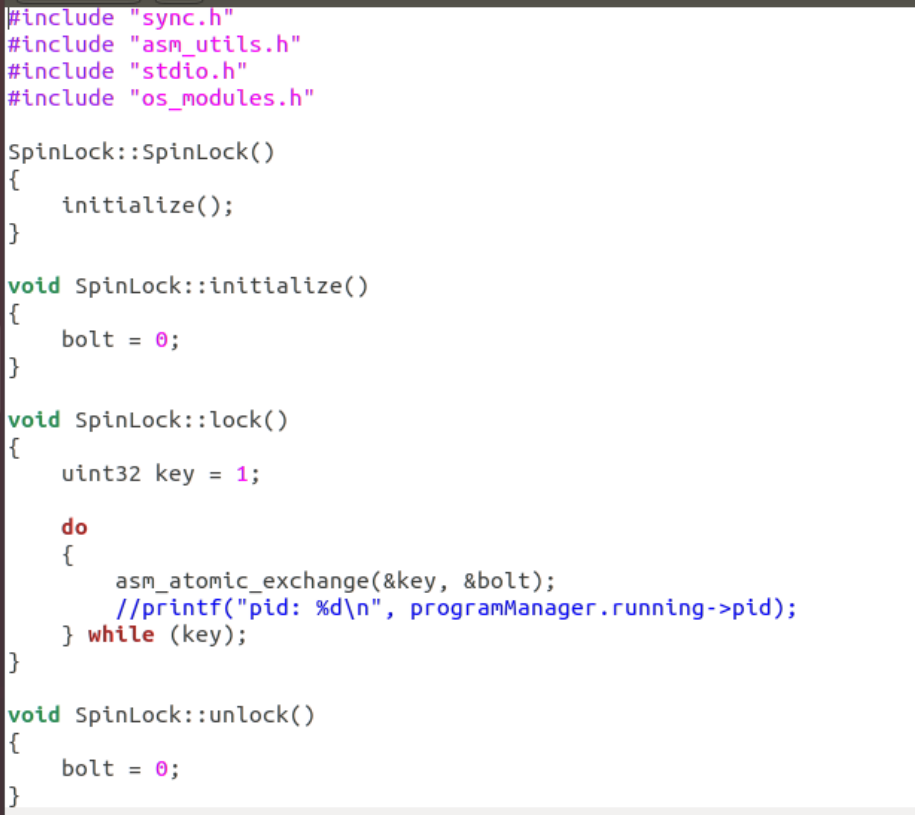


3.2）解决死锁

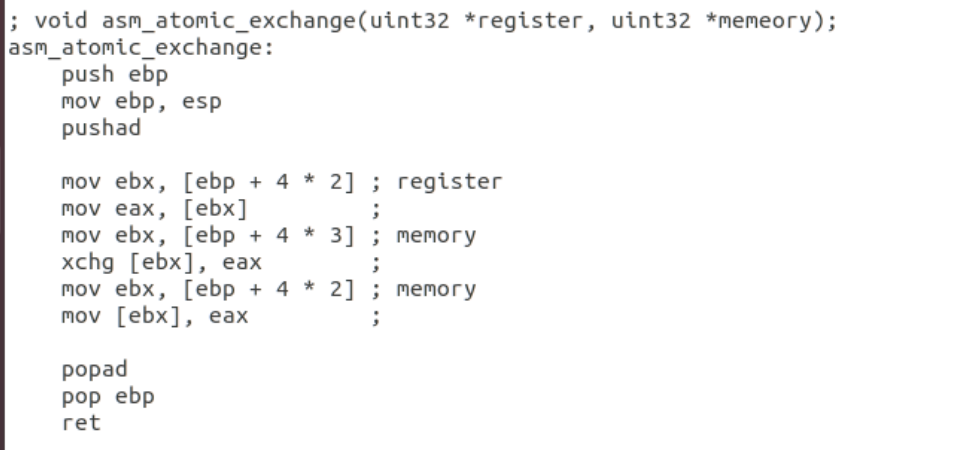
为了让死锁过程更容易发生，我们假设哲学家每拿起一根筷子都需要一定时间，为线程增加延时，较明显的看到一个死锁现象发生：所有的哲学家都拿起了左边的筷子但都不能就餐，为了解决此死锁问题，我们采用非对称的解决方案：即规定奇数号的哲学家先拿起他左边的筷子，然后再去拿起他右边的筷子；而偶数号的哲学家则先拿起他右边的筷子，然后再去拿他左边的筷子。可较好解决此死锁问题。

1. **关键代码**

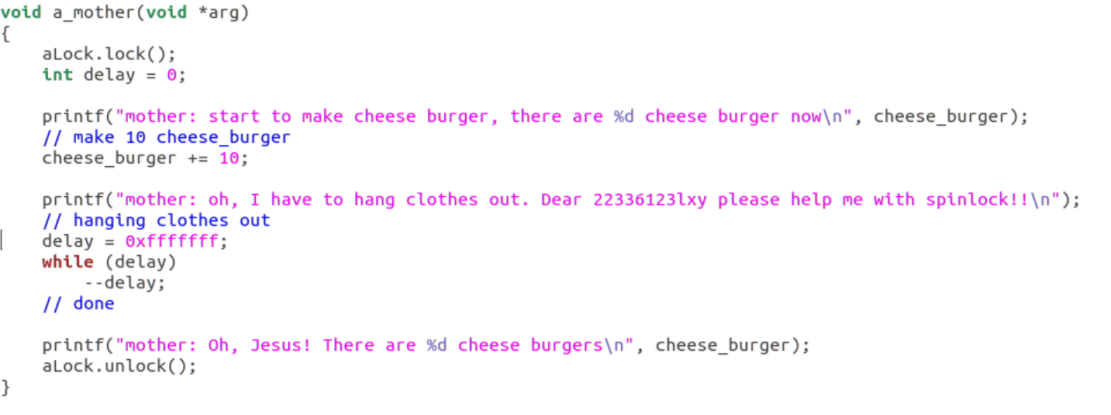
（1.1）自旋锁：

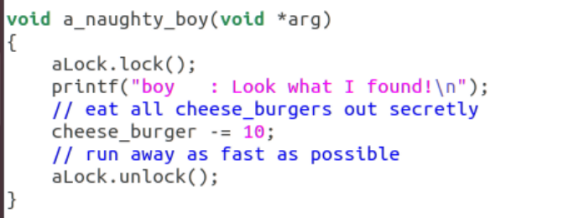


“原子”指令实现：实现了key值和bolt值的交换



加锁过程：



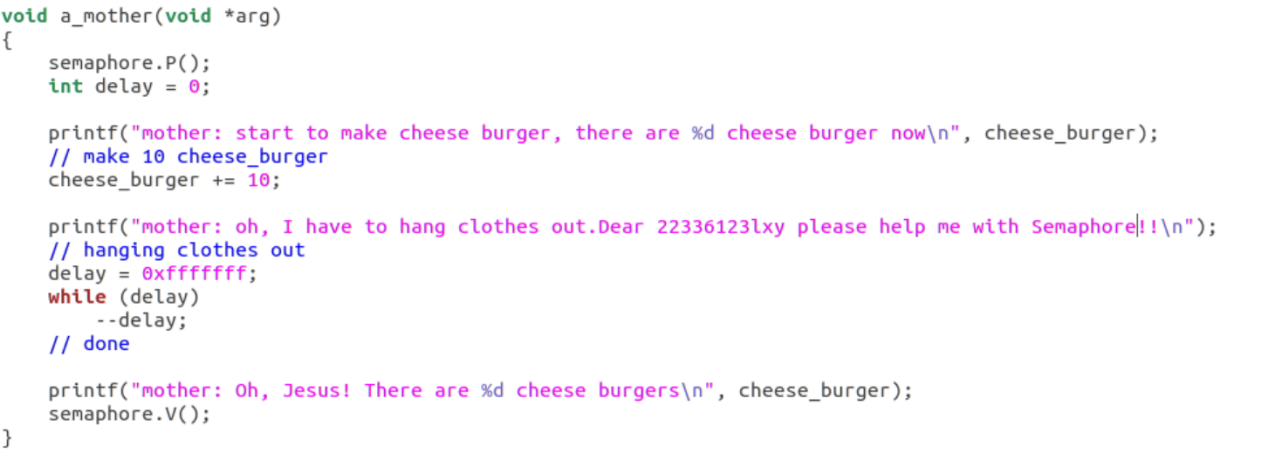


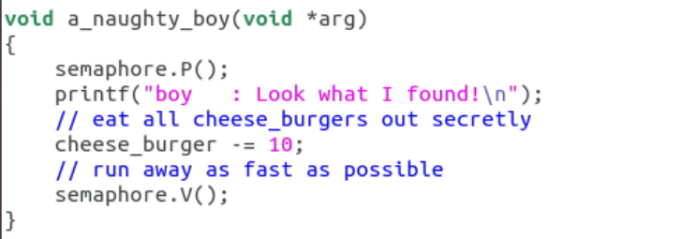
信号量实现：



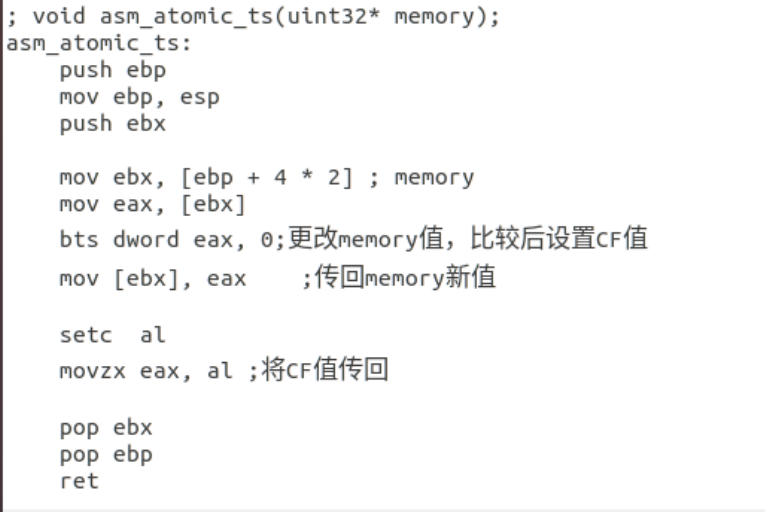


上锁过程：

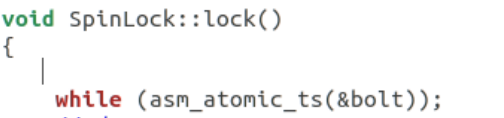




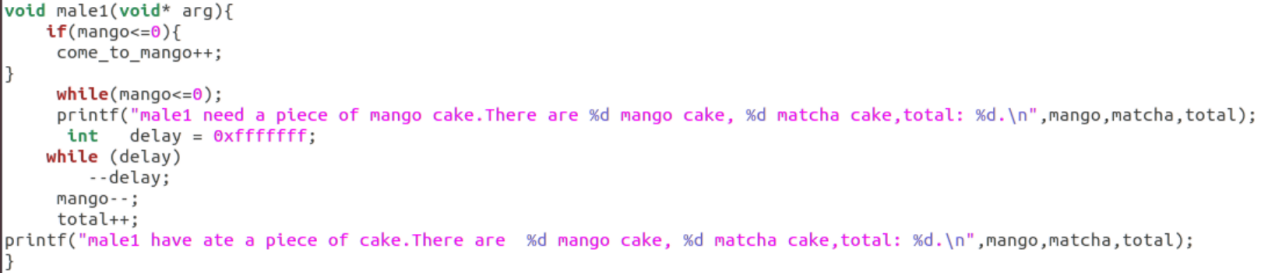
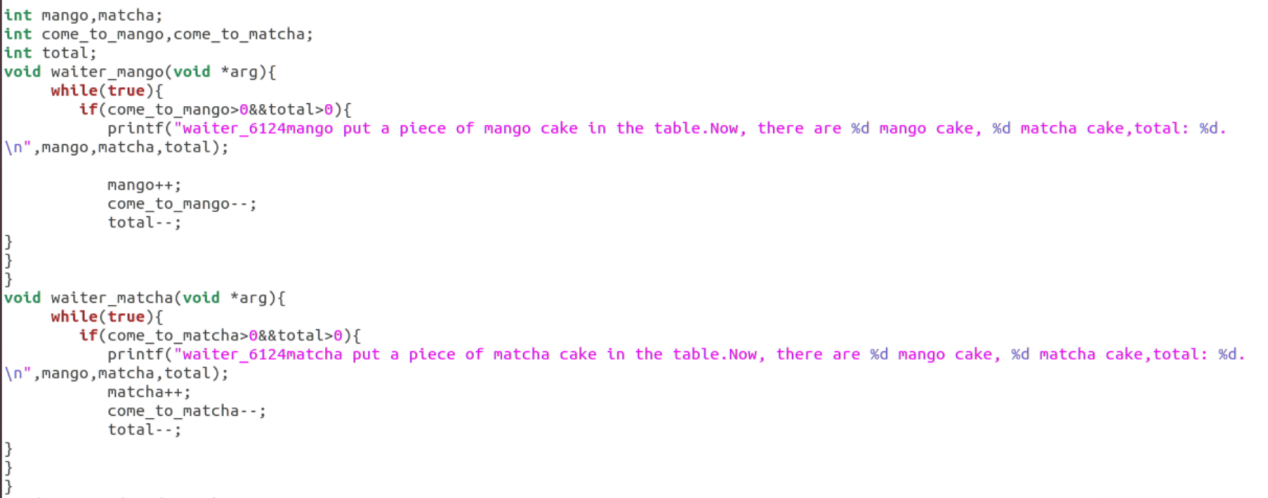
（1.2）换用bts实现自旋锁实现：

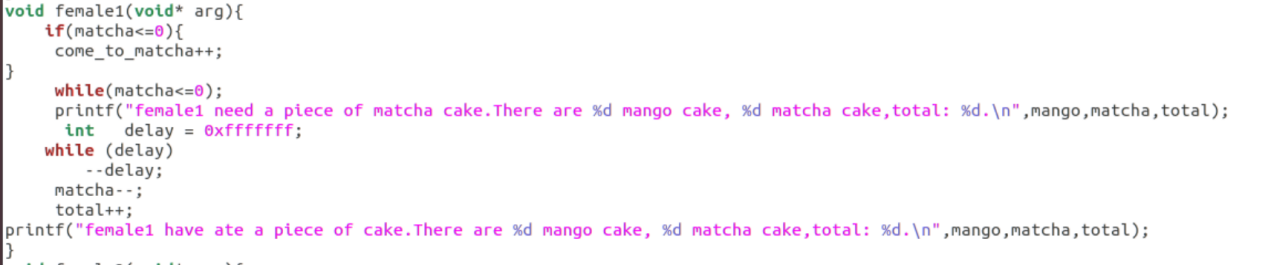


相应更改外界循环检查函数：

****

宴会蛋糕问题的简单线程实现：为了与后期信号量实现产生对比并易于想清楚信号量的设置，先采用常用数据类型进行资源记录，come\_to\_mango记录男士发出的蛋糕请求数,come\_to\_matcha记录女士发出的蛋糕请求数,mango,matcha分别记录蛋糕数量,total记录盘子空位数量，然后根据问题描述来理清蛋糕分配过程：当来到的男士发现芒果蛋糕没有了时会发出蛋糕请求，负责芒果蛋糕的服务员会先检查餐盘里是否可以放入新的蛋糕，如若可以将放入一块芒果蛋糕，后续男士将取出这块蛋糕进行享用，女士同理。

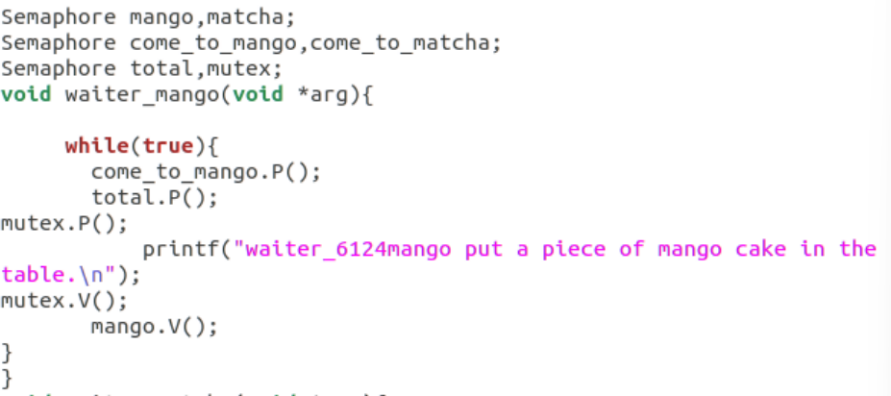


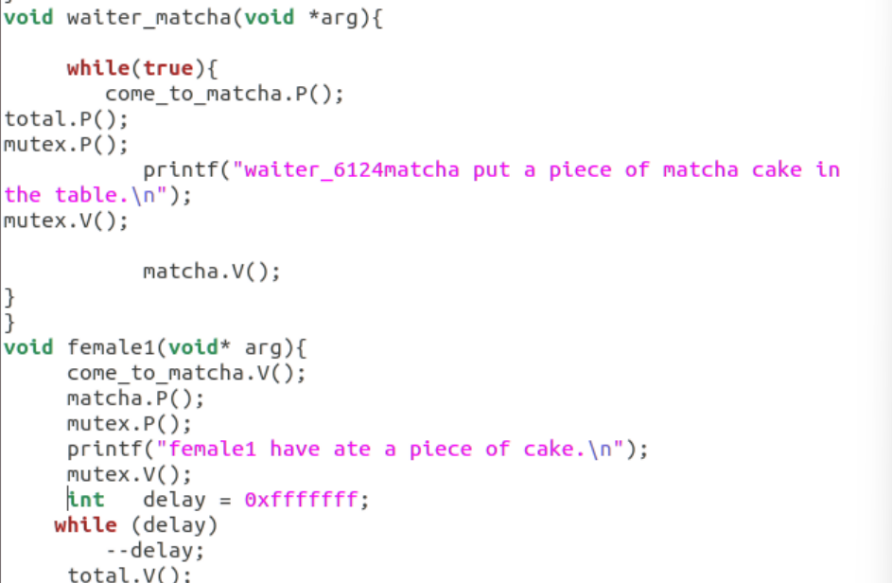


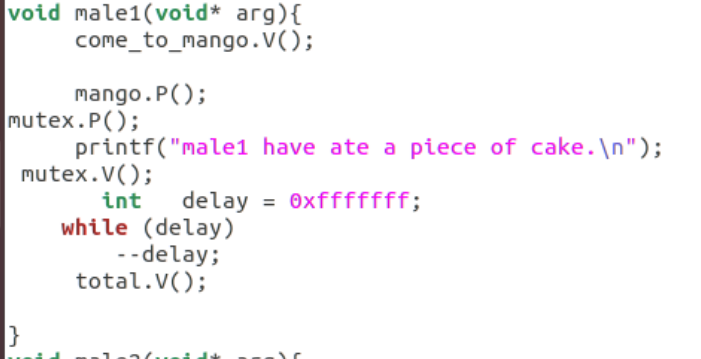
First\_thead中的改动：



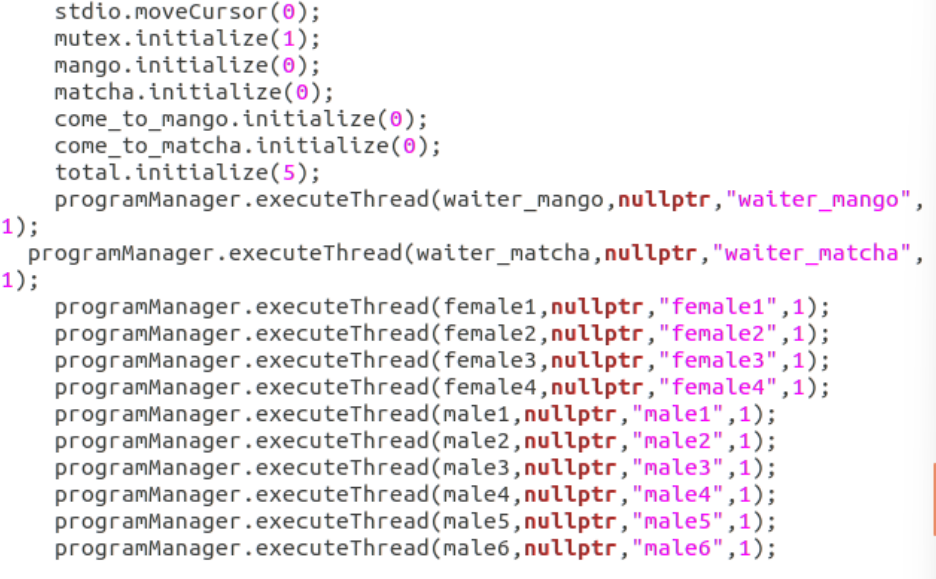
理顺思路后，我们使用信号量来表示此过程，信号量come\_to\_mango记录男士发出的蛋糕请求数,come\_to\_matcha记录女士发出的蛋糕请求数,mango,matcha分别记录蛋糕数量，以上信号量初始化为0,total记录盘子空位数量，初始化为5，mutex保证我们每次只对盘子里的蛋糕作一次操作，初始化为1。







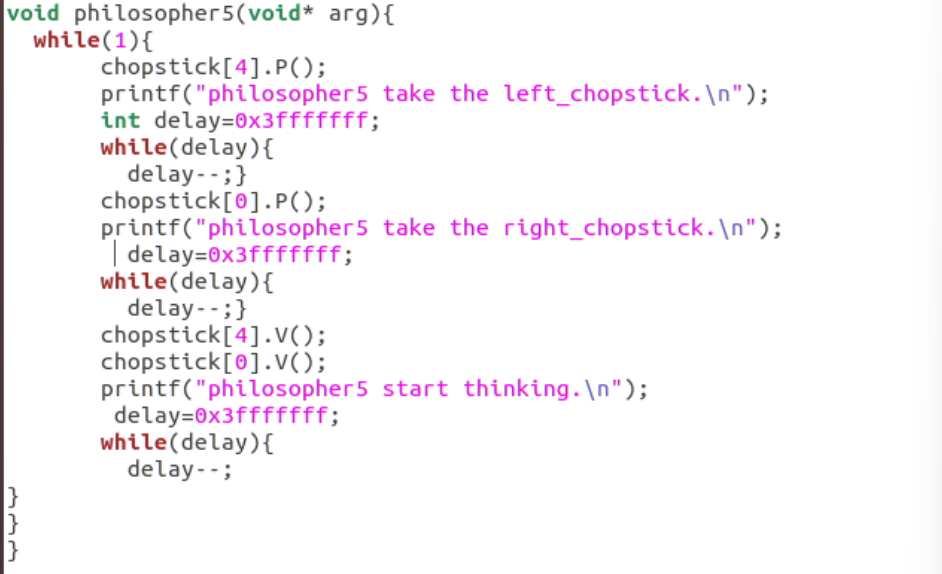
First\_thead中的改动：



（3.1）哲学家就餐问题：

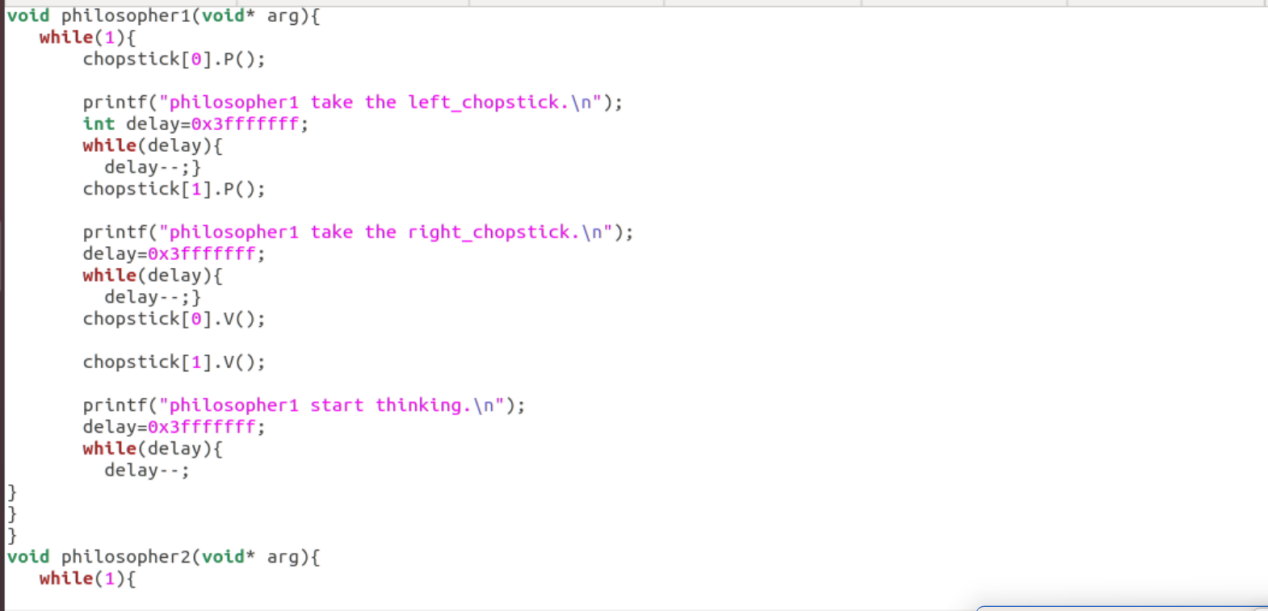
下方为显示死锁问题增加每次拿取筷子的时间后的代码，将此部分删除后，可以运行一段时间而不是立马死锁。

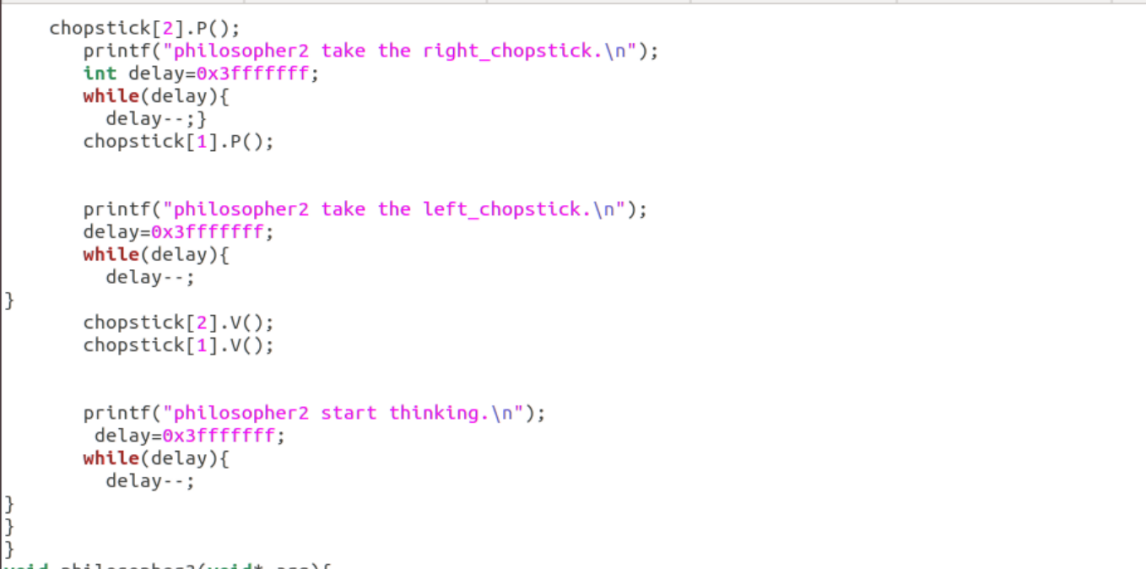




（3.2）非对称解决策略

奇数哲学家先拿左筷子，偶数哲学家先拿右筷子。

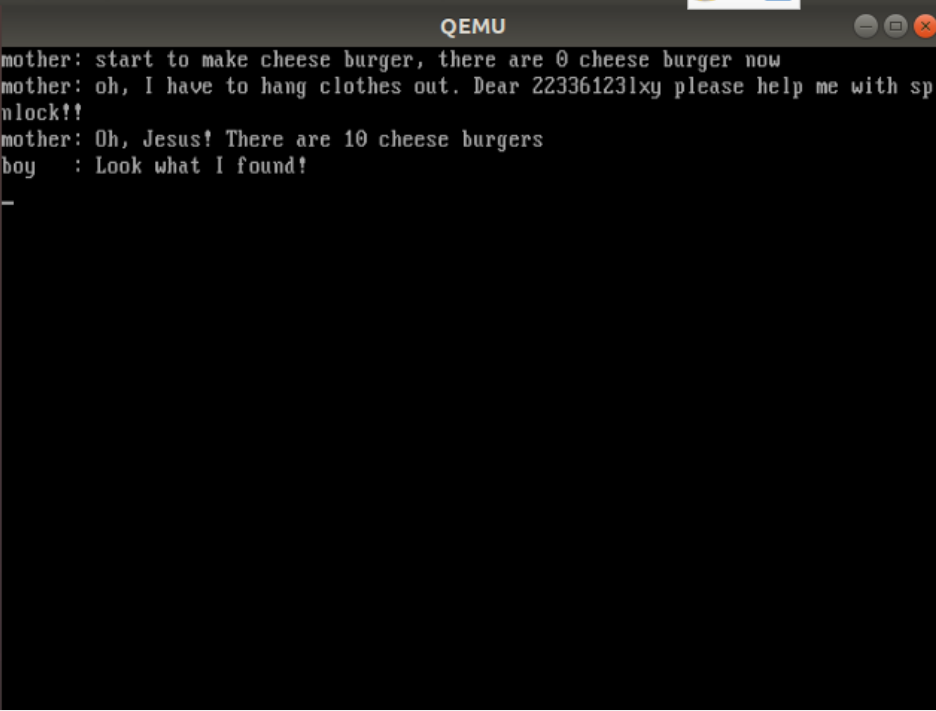




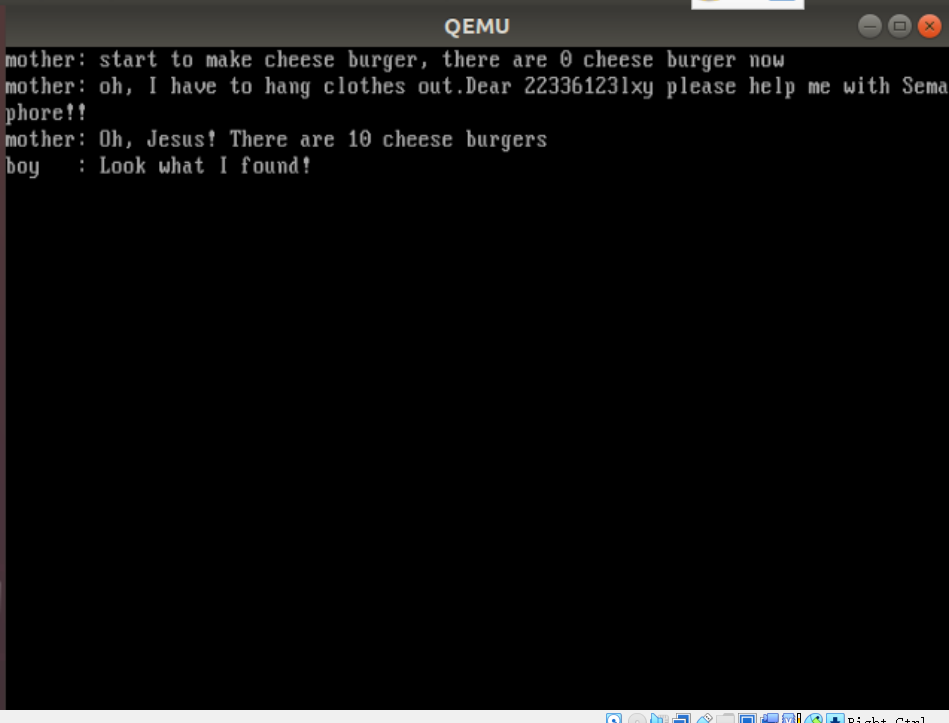
1. **实验结果**

（1.1）自旋锁和信号量解决互斥问题：

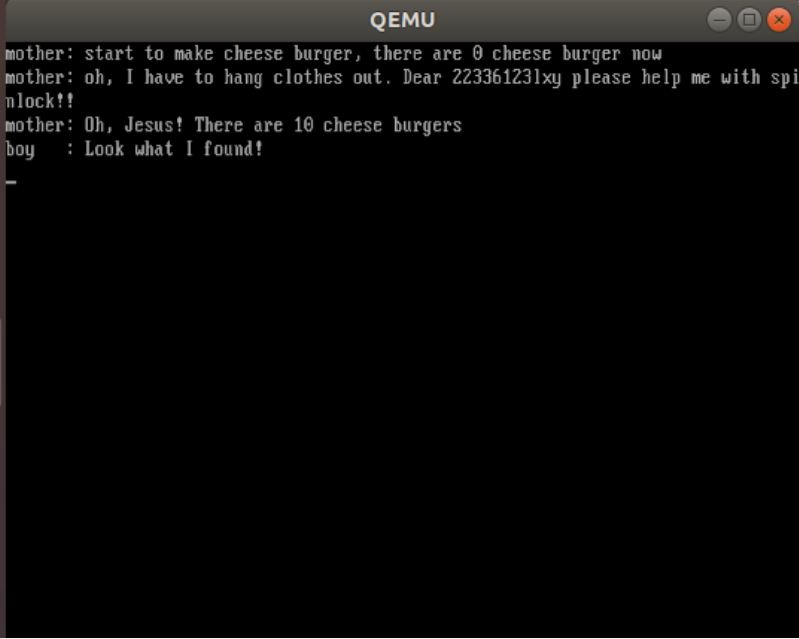
自旋锁：

****

信号量：

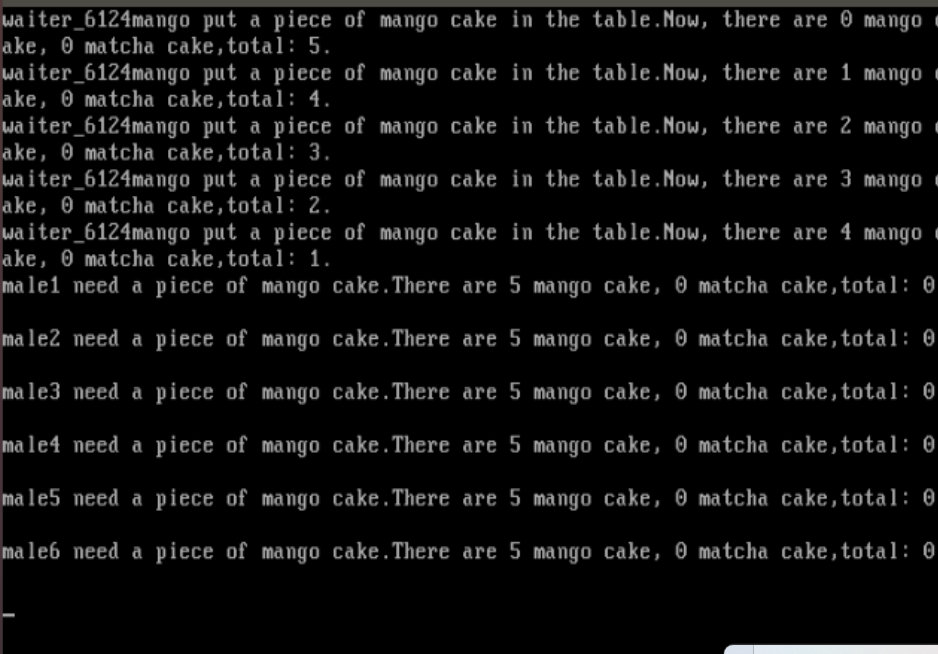
****

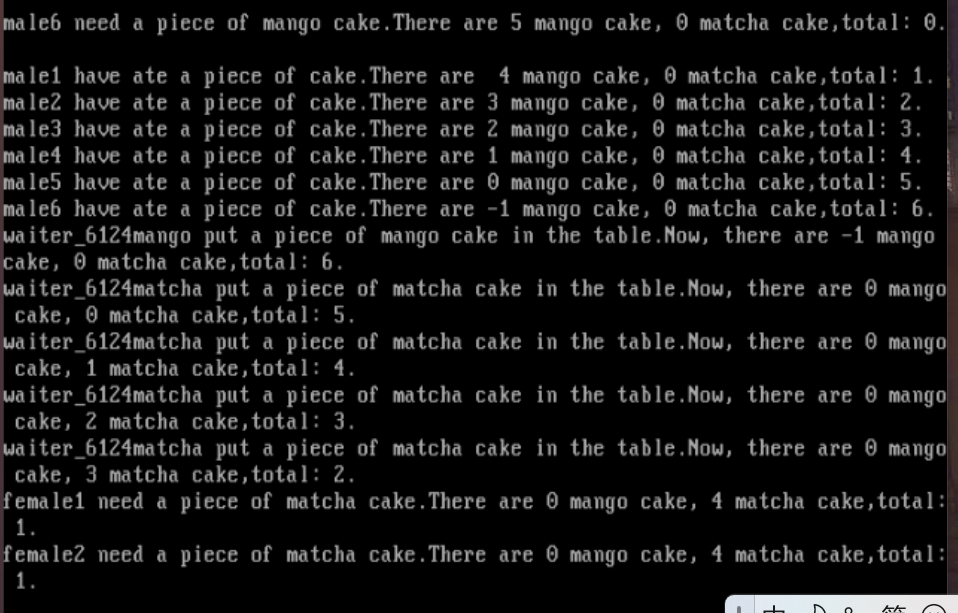
（1.2）改用bts实现



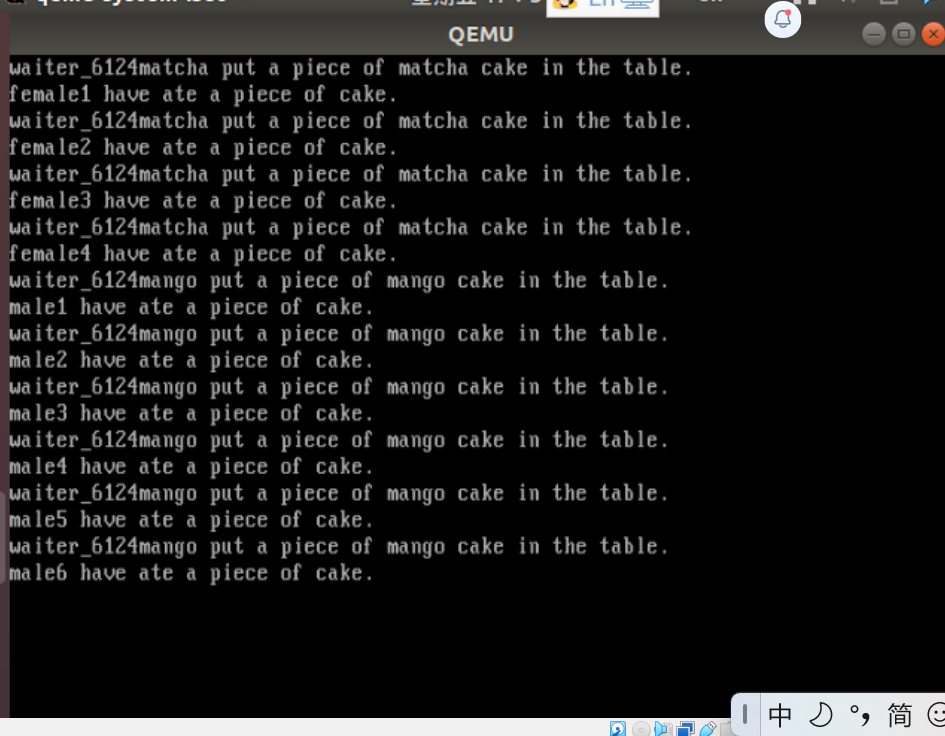
（2）信号量解决生产者—消费者问题

未使用信号量时会出现线程冲突：

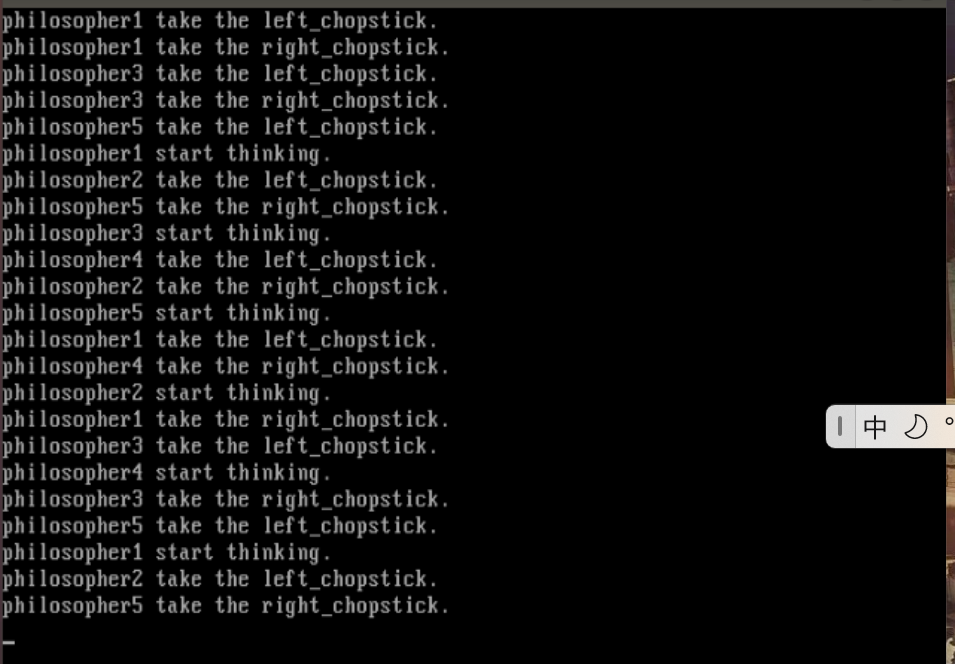




信号量实现：



（3.1）哲学家就餐问题：

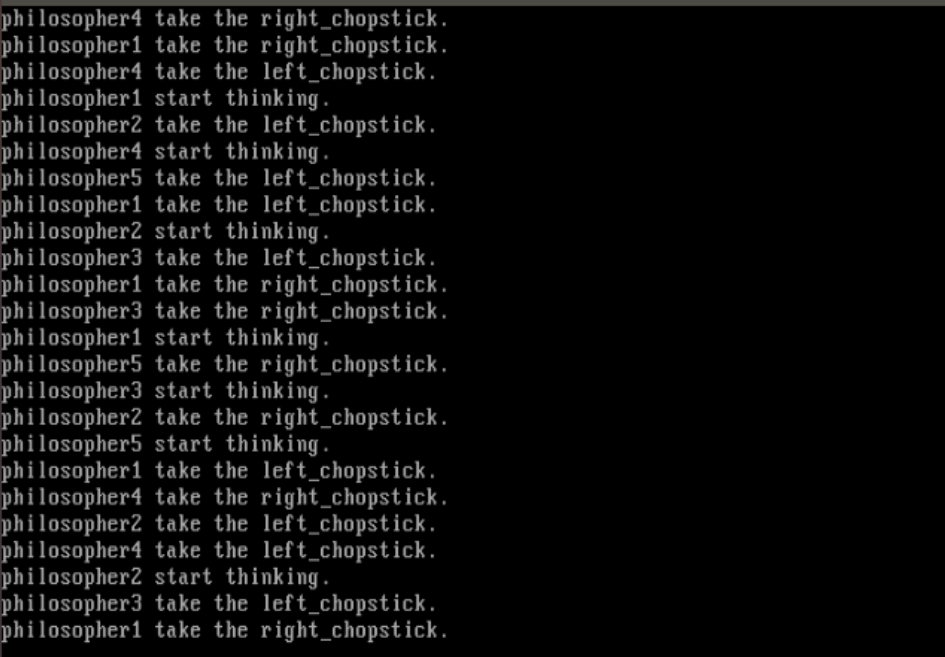


(3.2） 死锁现象及解决

死锁现象：



非对称策略解决后：

****

1. **总结**

对于哲学家问题的死锁呈现，因为一开始没把握住线程冲突的核心在于拿取筷子时的冲突，将延时设置在了吃饭和思考过程，无论如何加大延时都无法死锁，后和同学讨论后，在每一次拿取筷子后增加延时，达到效果。