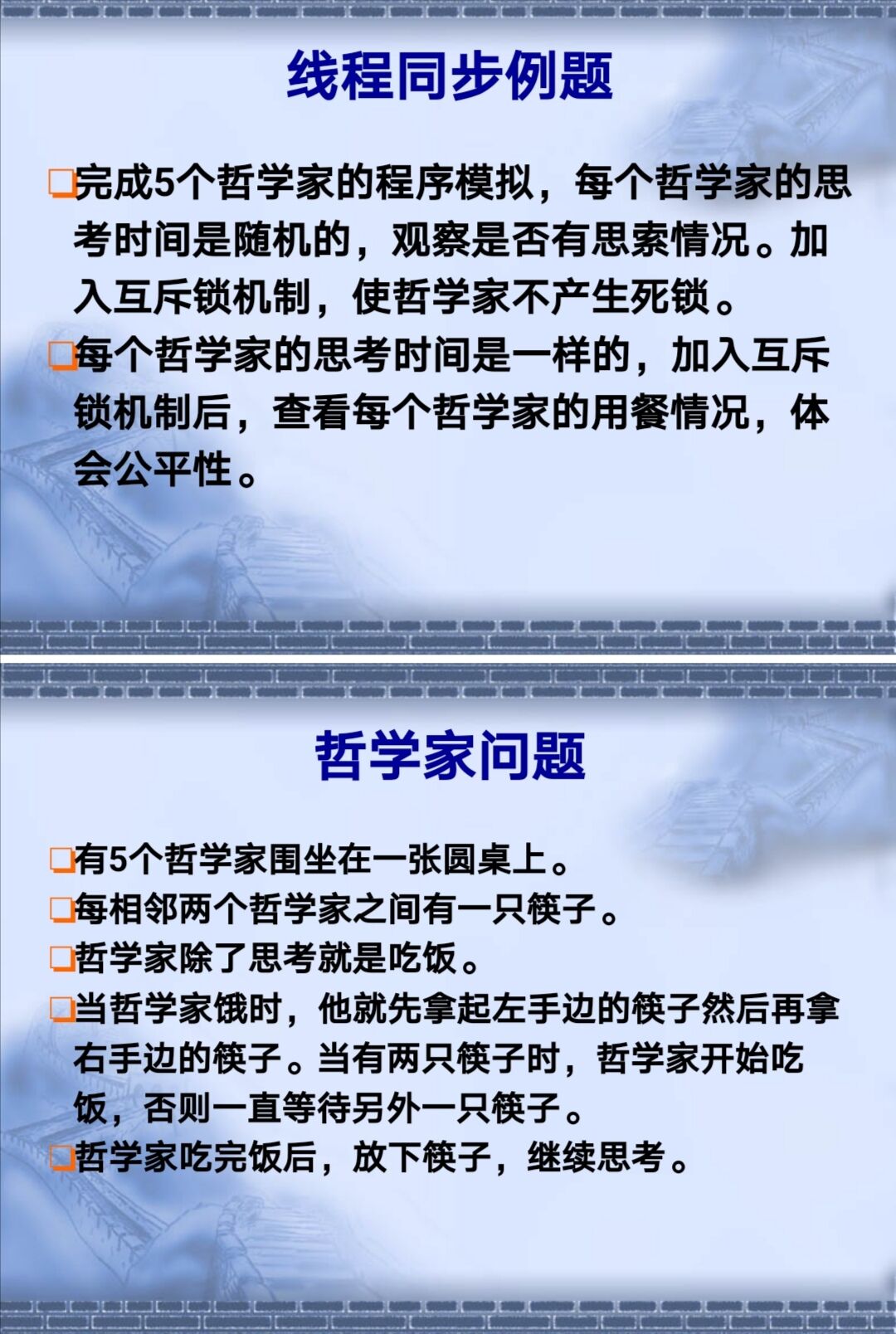
**问题描述：**

有五个哲学家，他们的生活方式是交替地进行思考和进餐。他们共用一张圆桌，分别坐在五张椅子上。在圆桌上有五个碗和五支筷子，平时一个哲学家进行思考，饥饿时便试图取用其左、右最靠近他的筷子，只有在他拿到两支筷子时才能进餐。进餐完毕，放下筷子又继续思考。



**约束条件**

(1)只有拿到两只筷子时，哲学家才能吃饭。

(2)如果筷子已被别人拿走，则必须等别人吃完之后才能拿到筷子。

(3)任一哲学家在自己未拿到两只筷子吃饭前，不会放下手中拿到的筷子。

**要解决“哲学家进餐问题”，就要明白以下几个名词的含义：**

（1）高并发：指的是是一种系统运行过程中遇到的一种“短时间内遇到大量操作请求”的情况,例如对资源CPU，寄存器的请求。

（2）多线程：简单来说，多线程就是多个执行序列。就是让cpu执行下这个序列，又执行下那个序列，不停地切换，正所谓一心二用。多线程是为了同步完成多项任务，不是为了提高运行效率，而是为了提高资源使用效率来提高系统的效率。

（3）多线程死锁：所谓死锁是指多个线程因竞争资源而造成的一种僵局（互相等待），若无外力作用，这些进程都将无法向前推进。

（4）信号量：多线程同步使用的；一个线程完成某个动作后通过信号告诉别的线程，别的线程才可以执行某些动作。（值可以为>=0的数）

（5）互斥量：多线程互斥使用的；一个线程占用某个资源，那么别的线程就无法访问，直到该线程离开，其他线程才可以访问该资源。(值只能为0和1)

**（6）那么如何解决高并发呢?**

答：如果使用单线程，那么程序就必须等待这些操作执行完成之后才能执行其他操作。可以使用多线程解决高并发，多线程在高并发问题中的作用就是充分利用计算机资源，使计算机的资源在每一时刻都能达到最大的利用率，不至于浪费计算机资源使其闲置。实际上是系统不断地在各个线程之间来回的切换（因为系统切换的速度非常的快，所以给我们在同时运行的错觉）。

**最直接的解决办法（引起死锁）**

最直接的解决办法是5只筷子分别设置为初始值为1的互斥信号量。这样可以保证不会有相邻的哲学家同时进餐，但是若5位哲学家同时拿起左筷子，都会因为拿不到右筷子而引起死锁。

**优化解决办法**

（1）

**方案一  
仅当哲学家的左，右两只筷子均可用时，才允许他拿起筷子进餐**

通过互斥信号量 mutex 对哲学家进餐之前取左侧和右侧筷子的操作进行保护，可以防止死锁的出现。（当第i个哲学家将左右筷子都拿到了才允许其他哲学家拿筷子）

如：哲学家4在拿起左筷子与右筷子中间没有其他的哲学家拿任何一个筷子，虽然他们也饿了，虽然他们要拿的筷子可能并没有被占用，所以这个会造成资源（筷子）得不到有效的利用。

（2）

**方案二**

**规定奇数号的哲学家先拿起他左边的筷子，然后再去拿他右边的筷子；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。按此规定，将是1、2号哲学家竞争1号筷子，3、4号哲学家竞争3号筷子。即五个哲学家都竞争奇数号筷子，获得后，再去竞争偶数号筷子，最后总会有一个哲学家能获得两支筷子而进餐**

规定奇数号哲学家先拿左筷子再拿右筷子，而偶数号哲学家相反。所以将是 2,3 号哲学家竞争 3 号筷子，4,5 号哲学家竞争 5 号筷子。1 号哲学家不需要竞争。最后总会有一个哲学家能获得两支筷子而进餐。

例如当哲学家4放下4号筷子时，4号筷子被释放，3号哲学家得到4号筷子（右筷子）进行进餐。

（3）

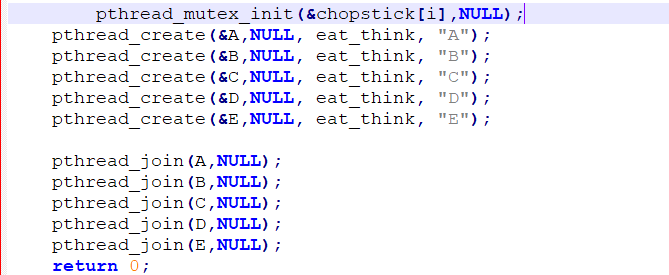
**方案三**  
**至多允许有四位哲学家同时去拿左边的筷子，然后在允许拿右边的筷子，最终能保证至少有一位哲学家能够进餐，并在用毕时能同时释放他用过的两只筷子，从而使更多的哲学家能够进餐**

至多只允许四位哲学家同时去拿左筷子，最终能保证至少有一位哲学家能进餐，并在用完后释放两只筷子供他人使用。此方法能使资源得到有效的利用。

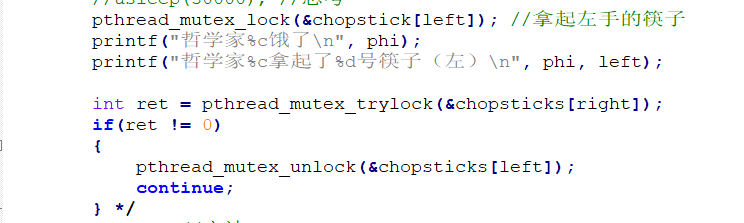
pthread-philosophers2-2.c程序代码

**实验过程**

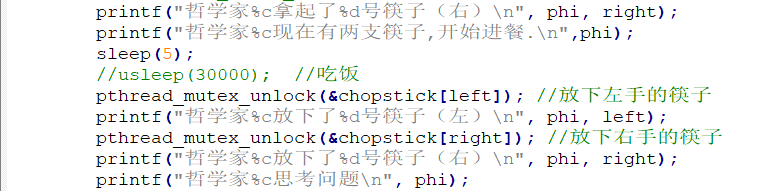
1. 筷子作为mutex，定义互斥锁
2. 通过无限循环保证实验的不断进行
3. 在主函数上进行互斥量的初始化，线程线程的创建，使用pthread\_join函数将线程挂起，并**不获取线程的终止状态**

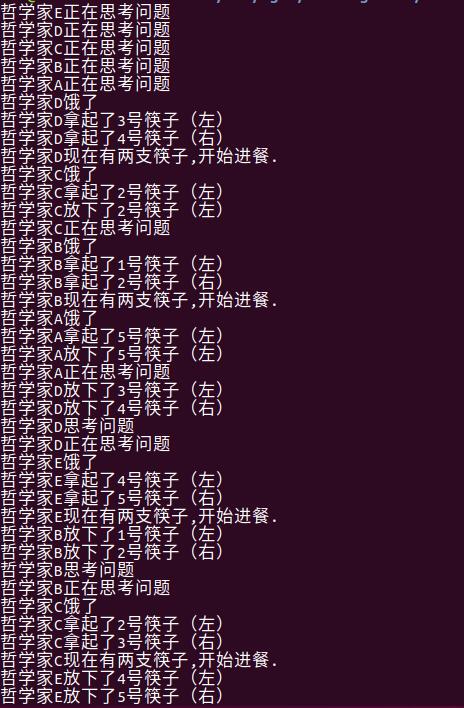


1. 拿筷子的顺序，是先左后右，每次拿筷子都进行加锁，并进行判断是否右边筷子被加锁，如果加锁，则放下资源，让其他线程使用



5、如果没有，就拿起来筷子，开始吃饭；一段时间之后在放下筷子，进行思考

实验结果



1. gcc编译时，加上-lphread选项，例：gcc .c -o 1,1 -lpthread

（gcc编译时，加上-phread选项，例：gcc .c -o 1,1 -pthread）

2.pthread\_create的第四个参数是void \*类型，我一开始传一个char类型，即'A',就会发生段错误。但改成char \*，即“A”就没问题了。

3.usleep是毫秒级的阻塞

**gcc中-pthread和-lpthread的区别**

一般情况下，我们在链接一个（文件名为libxxx.so或libxxx.a等的）库时，会使用-lxxx的方式；在Linux中要用到多线程时，需要链接pthread库，按照惯例，我们应该使用-lpthread的方式来进行链接；但是，通过日常的观察，我发现很多开源代码都是使用了-pthread参数，而非使用-lpthread

**参考链接：**

<https://blog.csdn.net/weixin_39656575/article/details/80719113?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-1&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-1>

<https://blog.csdn.net/kongzhp/article/details/7487150?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-4&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-4>

<https://blog.csdn.net/weixin_39531549/article/details/100857716>