**操作系统实验**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab4：

格式化输入函数scanf、信号量相关系统调用、

用户程序实现非死锁、高并行的哲学家问题

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

院系：人工智能学院

姓名：石睿

学号：211300024

班级：操作系统-2023春季学期

邮箱：[211300024@smail.nju.edu.cn](mailto:211300024@smail.nju.edu.cn)

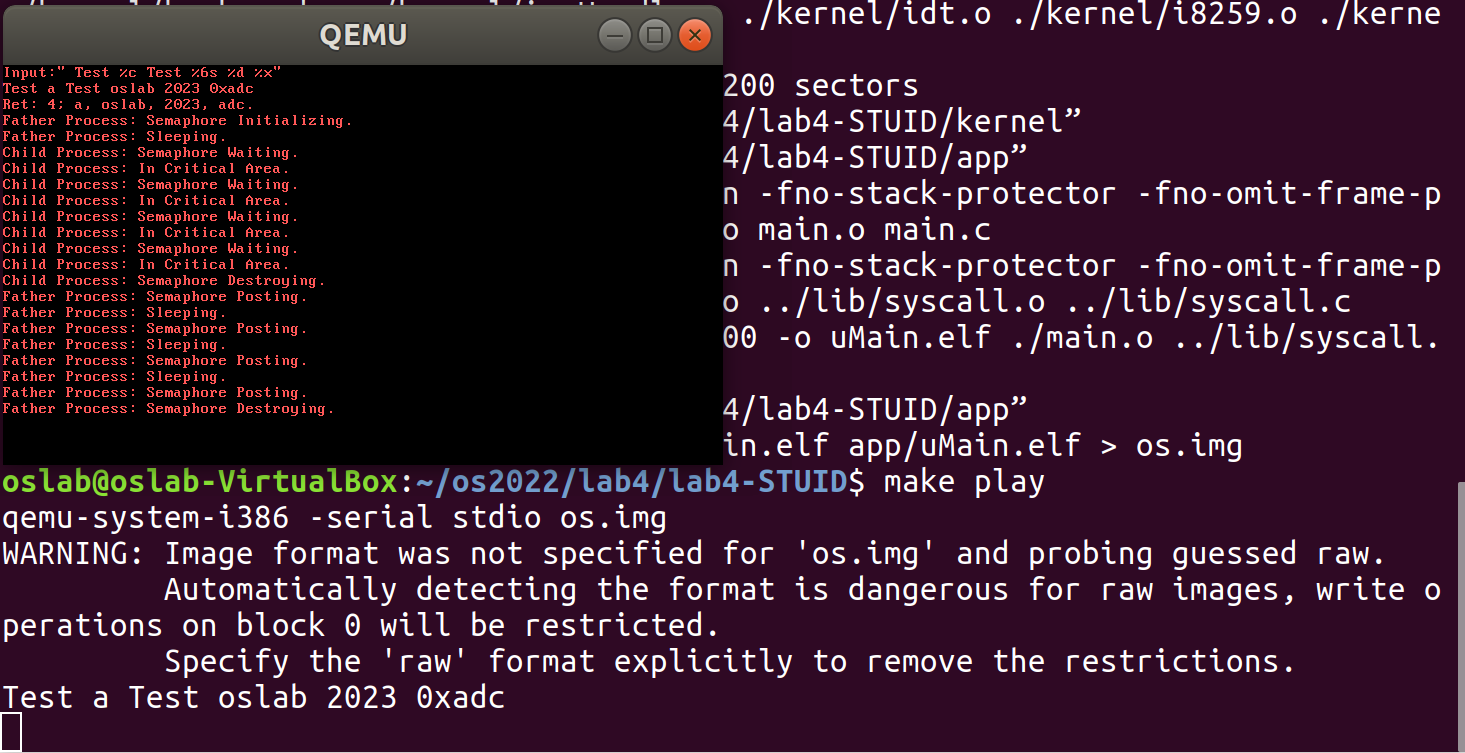
实验时间：2022.5.18

1. **实验进度**

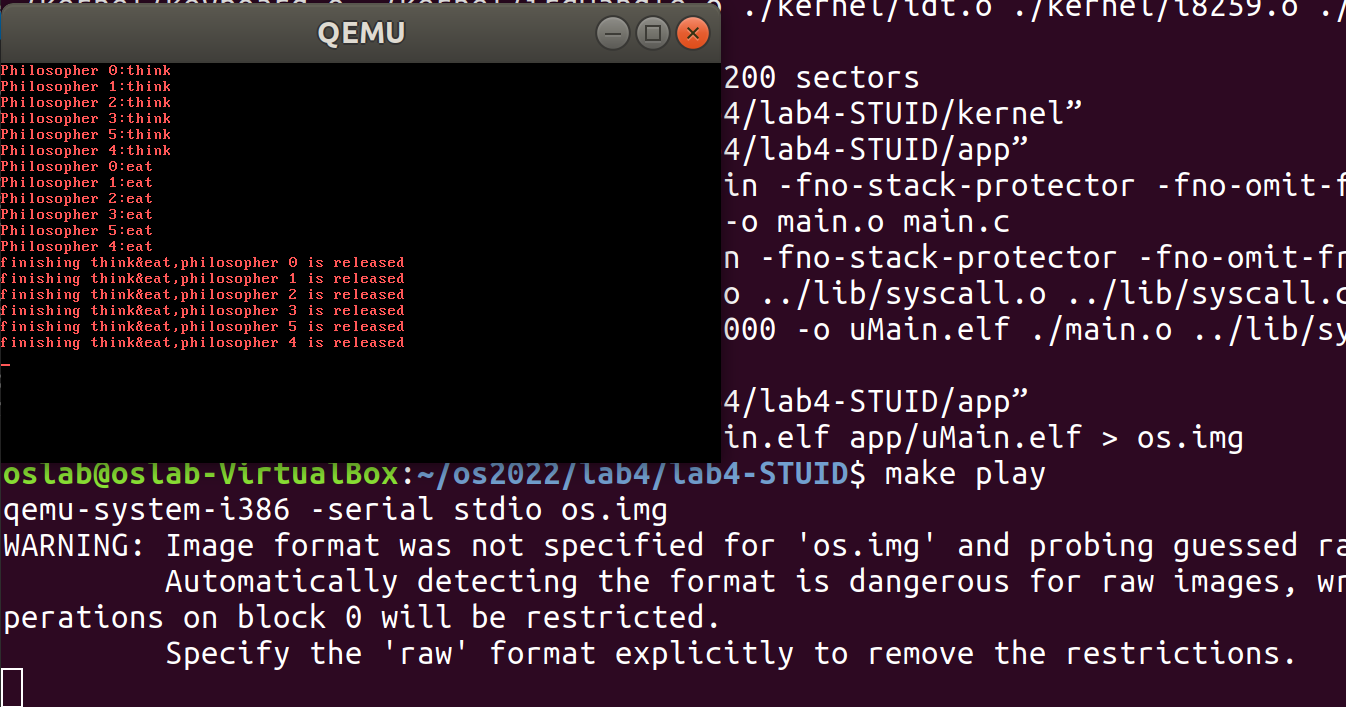
我已经完成了Lab4的所有内容。

1. **实验结果**

结果如下图所示，具体实现过程在第三部分介绍。



**完成格式化输入函数相关中断处理例程后结果**

****

**只跑一轮（for循环i=1）、哲学家初始化（now\_state数组）为Thinking时的运行结果**

1. **实验修改的代码位置**
   1. **格式化输入函数scanf**

Syscall.c中有关用户程序可以使用的scanf已经实现（调用了许多系统调用），只需要完成中断处理程序分发函数irqHandle.c中对应处理用户键盘输入的中断处理例程keyboardHandle和syscallReadStdIn两个函数。

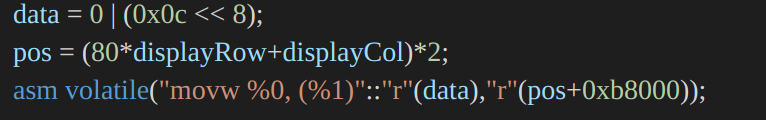
* 1. **实现keyboardHandle函数 = 写keybuffer**

本函数应完成的功能为[1]和[2]，以下对完成两个功能分别介绍：

[1]按下键盘中断后，把本字符所对应的keycode放到keybuffer之中。

按下键盘中断后，应该同时处理qemu的显示（显存）以及串口（serial）的显示，具体代码和lab2中一毛一样啦！需要处理：非法字符、退格符、回车符、正常字符的处理，如果合法，就放入keybuffer之中。放入keybuffer中的代码如下所示。

完成放入keybuffer之后，再滚屏（scrollscreen），并更新屏幕updateCursor即可。



[2]执行信号量V()操作，唤醒dev[STD\_IN]阻塞列表中的某一个进程。

使用内核初始化initSem中定义的Device信号量中的dev[STD\_IN]，在键盘中断将输入存入缓冲区后再让用户程序读取，而非一直监听键盘中断进行输入浪费资源。

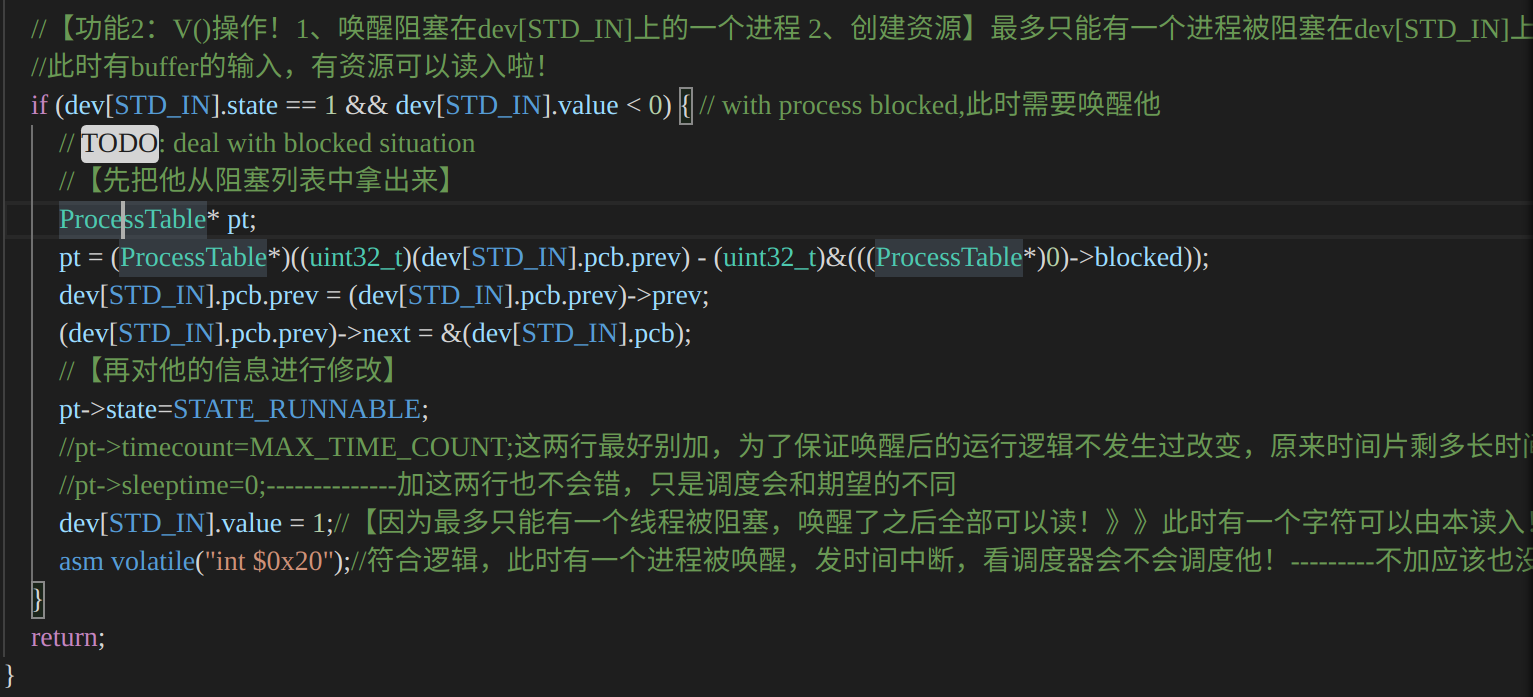
由规定可知，**最多只能有一个进程被阻塞到dev[STD\_IN]上**，而**如果有进程可以读keybuffer，那么他应该可以把keybuffer中的内容完全读入**！故此时dev[STD\_IN].value可以理解为（和传统信号量不太一样）：

1、当value<0的时候，表示阻塞在dev[STD\_IN]标准输入任务上的进程数量，且只能是-1。

2、当value>0的时候，表示当前在keybuffer中没有读入的字符个数。如果当前有进程从dev[STD\_IN]阻塞列表中唤醒，那么就应该全部读完，即让dev[STD\_IN]=0！

完成对dev[STD\_IN].value的更新后，就应该对唤醒的进程状态（processtable）的状态进行更新，并最后产生时间中断，由调度器完成下一次的进程调度。

具体代码实现如下。



* 1. **实现syscallReadStdIn函数 = 读keybuffer**

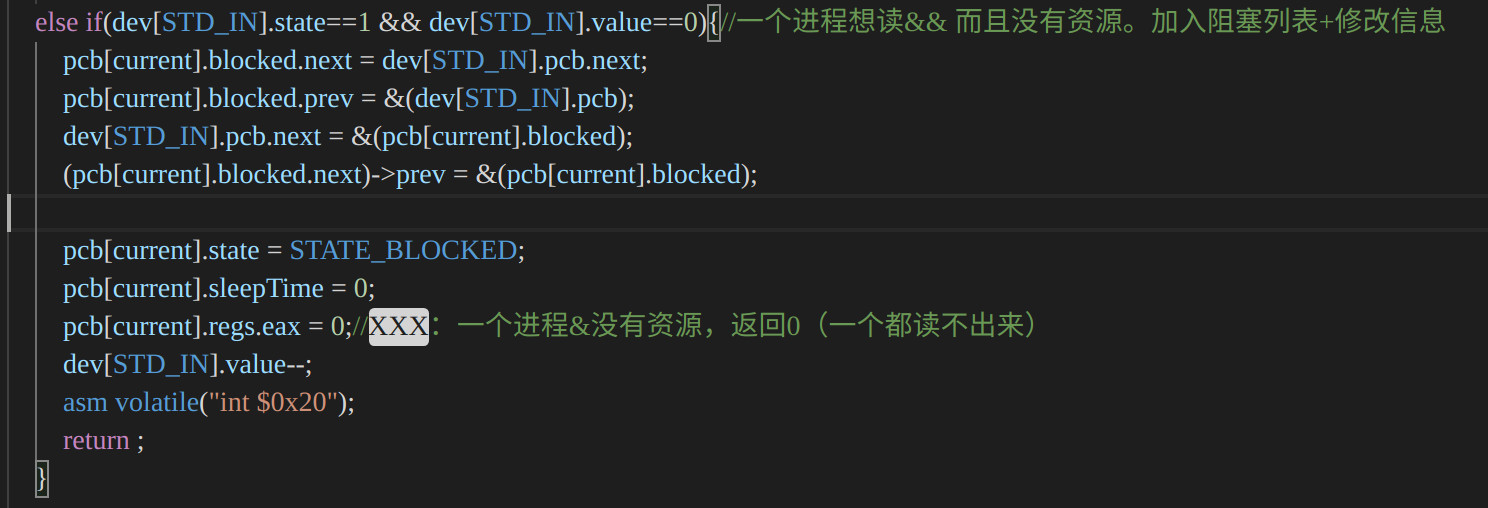
本函数应该完成的功能为[1]和[2]，以下分别进行介绍：

首先，注意到规定：“如果多个进程想读，那么后来的进程直接返回-1”。并且结合“最多只能有一个进程被阻塞在dev[STD\_IN]上”，故不把新进程加入阻塞列表中，直接pcb[current].regs.eax = -1;就可以啦！

[1]如果dev[STD\_IN].value=0（没有多余字符可读&没有其他阻塞进程），当前进程阻塞。

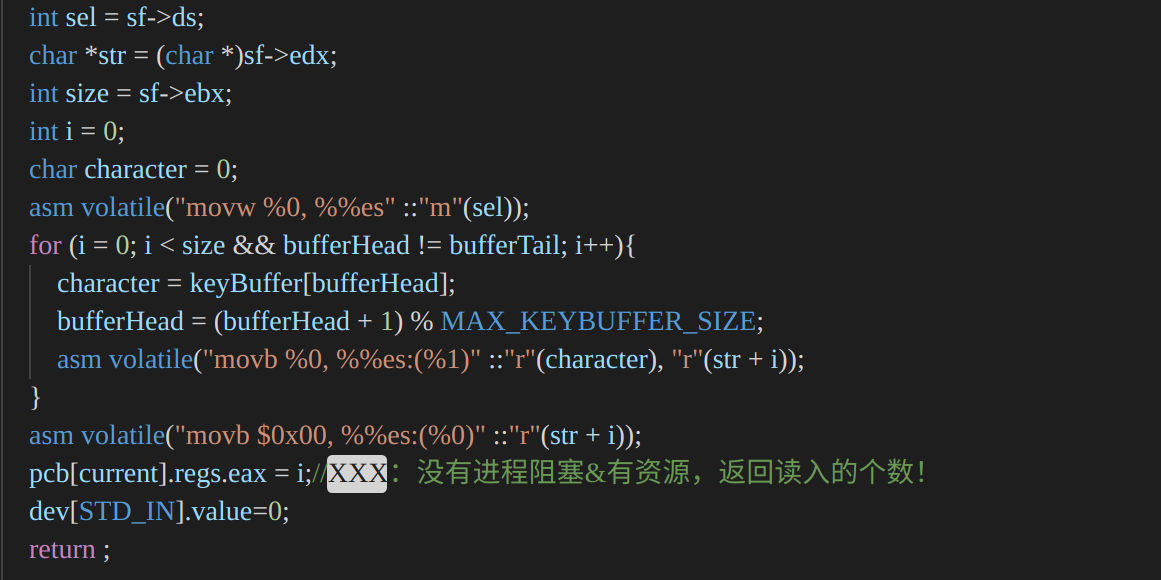
基本和V()的处理思路相同：加入到dev[STD\_IN]阻塞列表中，更新当前进程pcb[current]的状态，更新dev[STD\_IN].value的值（自减其实就相当于把value的值直接赋值为-1啦！）。

值得注意的是，此时调用scanf的返回值应该为0（当前进程一个字符也读不出来），最后调用时间中断，由调度器完成调度。代码实现如下



[2]进程被唤醒，读取keybuffer中所有的数据

此时只需要额外留意scanf的返回值，以及当前被唤醒的进程可以把keybuffer上面的字符全部读完，应显示赋值为0！（自减是不对的呢！）



* 1. **信号量相关系统调用**

注意到app/main.c中printf了“In Critical Area”，所以以下所有实现之前均进行了关中断操作，并在完成状态更新后开中断。

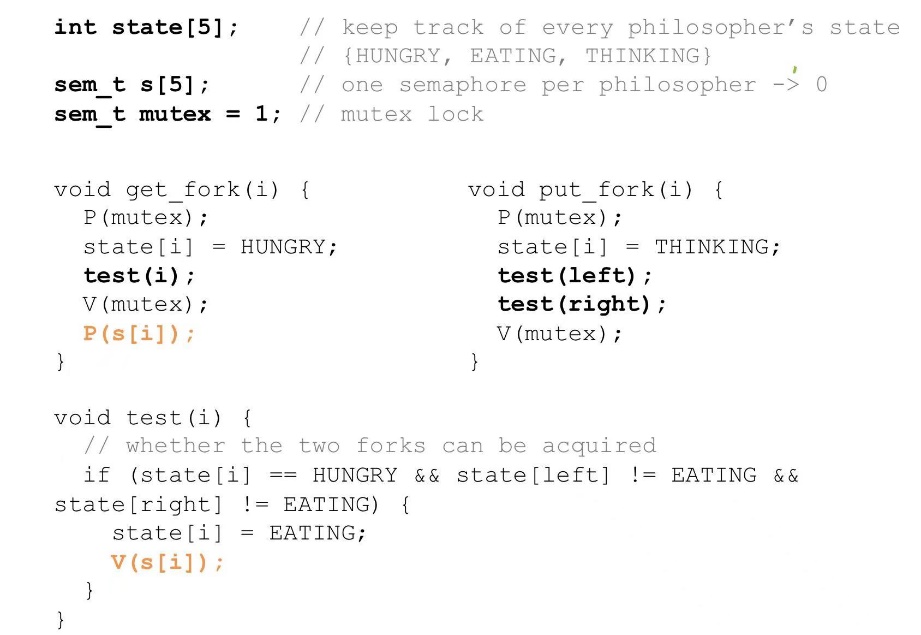
有以下值得注意的要点：

1. syscall中参数存放的位置 eax(系统调用号) ecx（参数1） edx（参数2），参数存放的位置是StackFrame在doIrq.S中压栈后形成的数据结构中的位置决定的。
2. 需要考虑输入不合法的情况，如到MAX\_SEM\_NUM后，没有位置给新信号量初始化、信号量数组的下标非法（小于0，大于最大值,,,）
3. 同第一步的实现，需要在P操作syscallSemWait和V操作syscallSemPost的最后引发时钟中断，由调度器决定接下来该谁执行啦！

其他实现依据实验手册逐步完成即可啦！

* 1. **哲学家就餐问题**

为了保证哲学家问题不出现死锁，且保证较高的并行度，并尽可能让最多的哲学家拿到筷子进行吃饭，对手册上的提到的可以不死锁的算法进行更改

****

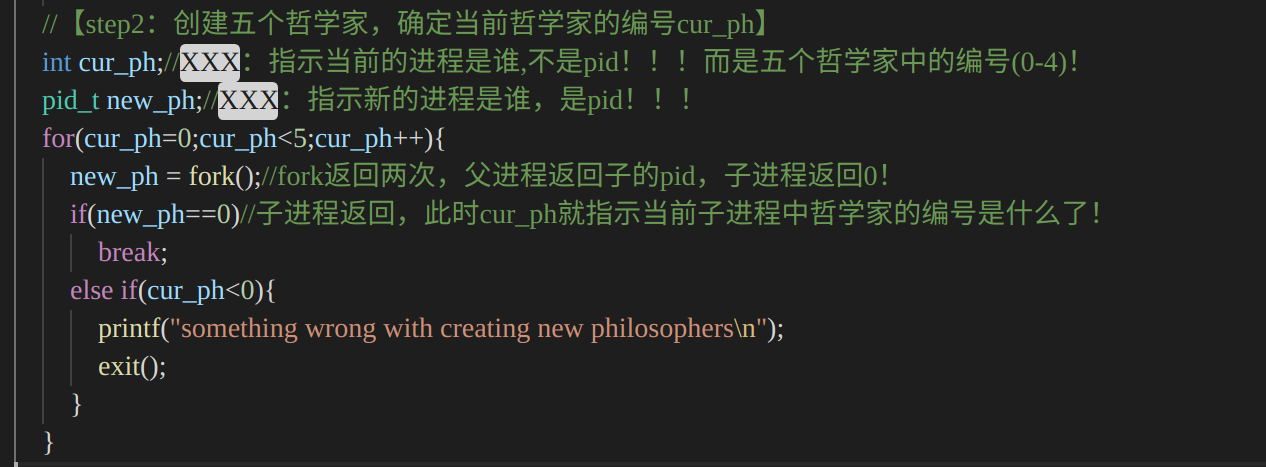
思路如下所述：

1、想吃饭的时候：只有在这哲学家同时具有左右两支筷子的时候才会拿起筷子并吃饭。

2、吃完饭的时候：放下筷子，同时检测左右两个相邻的哲学家有没有吃饭的需求，检测他们是否可以拿起筷子吃饭（左右两筷子都空闲）。

在这两个过程中，都用互斥锁放入临界区之中，只需要用信号量把初始值value设置成1。

在实现的过程中值得一提的是，**创建哲学家的过程中需要确认当前哲学家的编号**。因为在interleave的过程中，哲学家是交织执行的，只知道哲学家的pid，但不知道哲学家的具体编号。可以通过fork函数的两次返回值指示当前哲学家的编号！fork第一次在子进程中返回0，第二次在父进程中返回子进程的pid，所以可以通过fork的返回值是否为0，判断当前给哲学家分配的编号是几！

****

可以通过在在**代码标记step3的位置修改给定for循环的轮次**、**修改初始化now\_state数组、（哲学家的初始状态）**、**修改forks数组（哲学家当前可不可吃饭）**来增加测试用例。

**Tip：提交代码的时候，已把哲学家问题在app.main.c中屏蔽掉啦。如要测试，还请把lab4.1和lab4.2的内容屏蔽，并把对lab4.3的屏蔽内容删除。**