



实验课程：操作系统

实验名称：并发与锁机制

专业名称：计算机科学与技术

学生姓名：吴臻

学生学号：21307371

实验地点：实验中心大楼D栋501

实验成绩：

报告时间：2023/5/3

- 1. 实验要求
- 2. 预备知识与实验环境
- 3. 实验任务
- 4. 实验步骤/关键代码/实验结果

- **Assignment 1 自旋锁与信号量的实现**
 - 子任务1—利用自旋锁和信号量实现同步
 - 自旋锁
 - 信号量
 - 子任务2—自实现锁机制
- **Assignment 2 生产者-消费者问题(抽烟者与供应商)**
 - 子任务1—线程的竞争与冲突
 - 子任务2—利用信号量解决问题
- **Assignment 3 哲学家就餐问题**
 - 子任务1—简单解决方法
 - 子任务2—死锁应对策略
 - 死锁复现
 - 死锁解决
- 5. 总结(对实验过程中遇到的问题进行总结，可以提出对实验设置的改进意见)
- 6. 参考资料清单

1. 实验要求

在本次实验中，我们首先使用硬件支持的原子指令来实现自旋锁SpinLock，自旋锁将成为实现线程互斥的有力工具。接着，我们使用SpinLock来实现信号量，最后我们使用SpinLock和信号量来给出两个实现线程互斥的解决方案。

2. 预备知识与实验环境

3. 实验任务

1. 自旋锁与信号量的实现

- (1) 利用自旋锁和信号量实现同步
- (2) 自实现锁机制

2. 生产者-消费者问题

- (1) 线程的竞争与冲突
- (2) 利用信号量解决问题

3. 哲学家就餐问题

(1) 简单解决方法

(2) 死锁应对策略

4. 实验步骤/关键代码/实验结果

Assignment 1 自旋锁与信号量的实现

子任务1—利用自旋锁和信号量实现同步

任务要求：在实验6中，我们实现了自旋锁和信号量机制。现在，请同学们分别利用指导书中实现的自旋锁和信号量方法，解决实验6指导书中的“消失的芝士汉堡”问题，保存结果截图并说说你的总体思路。

注意：请将你姓名的英文缩写包含在某个线程的输出信息中（比如，代替母亲或者儿子），用作结果截图中的个人信息表征。

自旋锁

实验步骤

1. 定义一个自旋锁的类（为0时可以进入临界区，为1时不断循环等待）
2. 创建两个线程，观察两个线程是否出现竞争现象（改变共有变量）

关键代码（完整代码放置在src/Assignment 1）

自旋锁类的定义

```
class SpinLock
{
private:
    uint32 bolt;
public:
    SpinLock();
    void initialize();
    void lock();
    void unlock();
};
```

开锁及解锁

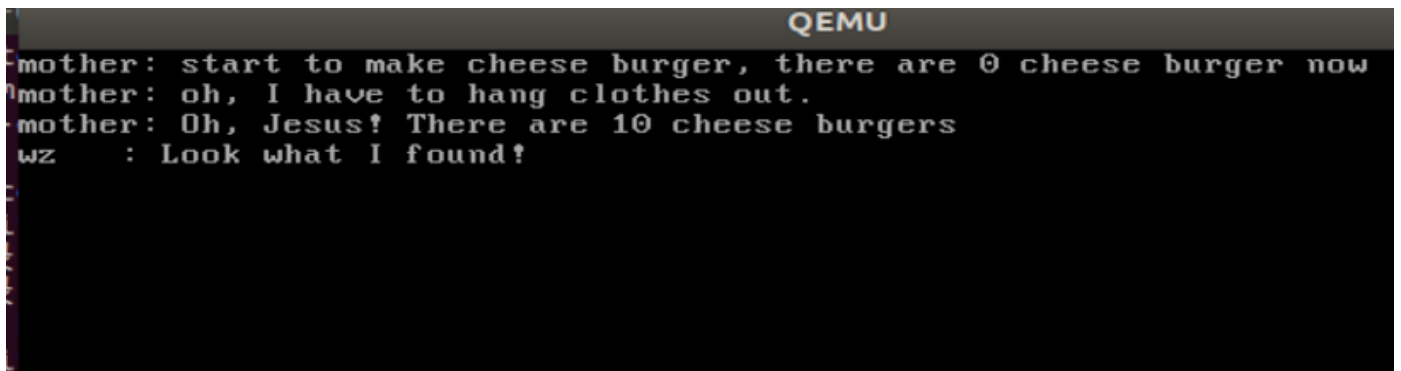
```
void SpinLock::lock()
{
    uint32 key = 1;

    do
    {
        asm_atomic_exchange(&key, &bolt);
        //printf("pid: %d\n", programManager.running->pid);
    } while (key);
}

void SpinLock::unlock()
{
    bolt = 0;
}
```

线程执行的函数请看src/2里的sync.cpp

实验结果



信号量

实验步骤

1. 定义信号量Semaphore
2. 实现P操作
3. 实现V操作

关键代码

信号量Semaphore

```
class Semaphore
{
private:
```

```

uint32 counter;
List waiting;
SpinLock semLock;

public:
    Semaphore();
    void initialize(uint32 counter);
    void P();
    void V();
};

```

P操作

```

void Semaphore::P()
{
    PCB *cur = nullptr;

    while (true)
    {
        semLock.lock();
        if (counter > 0)
        {
            --counter;
            semLock.unlock();
            return;
        }

        cur = programManager.running;
        waiting.push_back(&(cur->tagInGeneralList));
        cur->status = ProgramStatus::BLOCKED;

        semLock.unlock();
        programManager.schedule();
    }
}

```

V操作

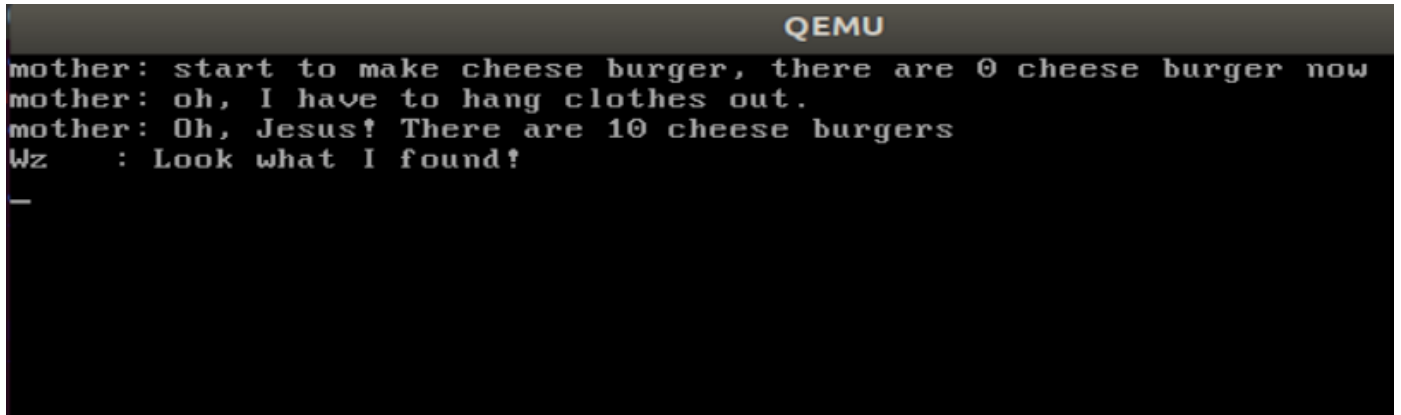
```

void Semaphore::V()
{
    semLock.lock();
    ++counter;
    if (waiting.size())
    {
        PCB *program = ListItem2PCB(waiting.front(), tagInGeneralList);
        waiting.pop_front();
        semLock.unlock();
        programManager.MESA_WakeUp(program);
    }
    else
    {

```

```
        semLock.unlock();
    }
}
```

实验结果



子任务2—自实现锁机制

实验步骤

只需要修改`asm_atomic_exchange()`函数即可，将其中原来的`xchg`原子指令修改成`lock bts`指令

关键代码（完整代码放置在src/Assignment 1）

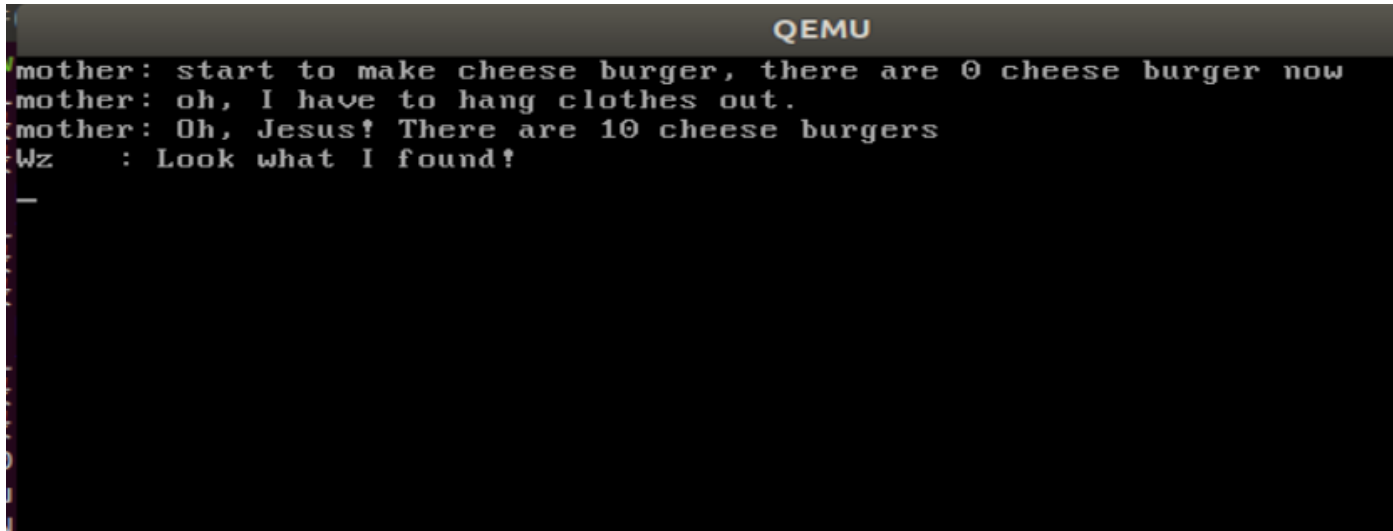
```
asm_atomic_exchange:
    push ebp
    mov ebp, esp
    pushad

    xor eax, eax
    mov ebx, [ebp + 4 * 3] ; memory
    lock bts [ebx], eax    ;将b0lt的值设为1, 将原来的值放入CF
    setc al                ;将进位标志 CF 的值设置给 al 寄存器

    mov ebx, [ebp + 4 * 2] ;
    mov [ebx], eax         ;

    popad
    pop ebp
    ret
```

实验结果



Assignment 2 生产者-消费者问题(抽烟者与供应商)

子任务1—线程的竞争与冲突

任务要求：在子任务1中，要求不使用任何实现同步/互斥的工具。因此，不同的线程之间可能会产生竞争/冲突，从而无法得到预期的运行结果。请同学们将线程竞争导致错误的场景呈现出来，保存相应的截图，并描述发生错误的场景。（提示：可通过输出共享变量的值进行观察）

实验步骤

1. 首先创建一个producer进程用来生产原材料，然后创建三个消费者进程用来表示是否可以得到一根完整的烟（共享变量为原材料）
2. 在判断能否生成烟时，结合题目要求，理想情况下我们只需判断目前吸烟者持有的原材料生产者是否有给出，有则无法得到烟，无则恰好生成烟，得到烟的同时，我们无需将原材料清零（跟我们的判断条件有关，清零会导致判断出错）

关键代码（完整代码放置在src/Assignment 2）

```
void producer(void *arg){
    // 不断生产
    while(1){
        //i表示哪个不生产
        for(int i = 0; i < 3; i++){
            for(int j = 0; j < 3; j++){
                if(i != j){
                    material[j] = 1;
                }else{
```

```

        material[j] = 0;
    }
}
}

void first_smoker(void *arg){
    while(1){
        for(int i = 0; i < 100000000; i++){
            int hold = 0;
            if(material[hold] == 0 && i % 40000000 == 0){
                printf("heyhey first_smoker (7371) got one %d %d %d \n
",material[0],material[1],material[2]);
            }
        }
    }
}

void second_smoker(void *arg){
    while(1){
        for(int i = 0; i < 100000000; i++){
            int hold = 1;
            if(material[hold] == 0 && i % 40000000 == 0){
                printf("heyhey second_smoker got one %d %d %d \n
",material[0],material[1],material[2]);
            }
        }
    }
}

void third_smoker(void *arg){
    while(1){
        for(int i = 0; i < 100000000; i++){
            int hold = 2;
            if(material[hold] == 0 && i % 40000000 == 0){
                printf("heyhey third_smoker got one %d %d %d \n
",material[0],material[1],material[2]);
            }
        }
    }
}

```

实验结果


```

heyhey second_smoker got one 1 0 1
heyhey second_smoker got one 1 0 1
heyhey second_smoker got one 1 0 1
heyhey second_smoker got one 1 0 1
heyhey first_smoker (7371) got one 0 1 1
heyhey first_smoker (7371) got one 0 1 1
heyhey second_smoker got one 1 0 1
heyhey second_smoker got one 1 0 1
heyhey third_smoker got one 1 1 0
heyhey third_smoker got one 1 1 0
heyhey second_smoker got one 1 0 1
heyhey second_smoker got one 1 0 1
heyhey first_smoker (7371) got one 0 1 0
heyhey first_smoker (7371) got one 0 1 0
heyhey third_smoker got one 0 1 0
heyhey third_smoker got one 0 1 0
heyhey second_smoker got one 1 0 1
heyhey second_smoker got one 1 0 1
heyhey first_smoker (7371) got one 0 1 0
heyhey first_smoker (7371) got one 0 1 0
heyhey third_smoker got one 0 1 0

```

分析：由图中的蓝框可以看到当第一种原材料和第三种原材料都缺少时，第一个抽烟者竟然合成了烟，原因就在于producer在重置原材料的途中被打断了（被第一个抽烟者的进程抢占了），导致实验结果出错

子任务2—利用信号量解决问题

任务要求：针对你选择的问题场景，简单描述该问题中各个线程间的互斥关系，并使用信号量机制实现线程的同步。说说你的实现方法，并保存能够证明你成功实现线程同步的结果截图。

大致思路

1. 在抽烟者与供应商这个问题中，导致输出与预期不符的原因在于抽烟者与生产者对原材料的竞争
2. 只需在生产者修改原材料的时候，使用信号量的P操作，修改完之后，再进行V操作即可，这样便可以避免抽烟者访问未完全重置的原材料，因为抽烟者未改变原材料，所以抽烟者之间可以不需要PV操作(无需修改)

关键代码（完整代码放置在src/Assignment 2）

```

void producer(void *arg){
    // 不断生产
    while(1){
        //i表示哪个不生产
        for(int i = 0; i < 3; i++){

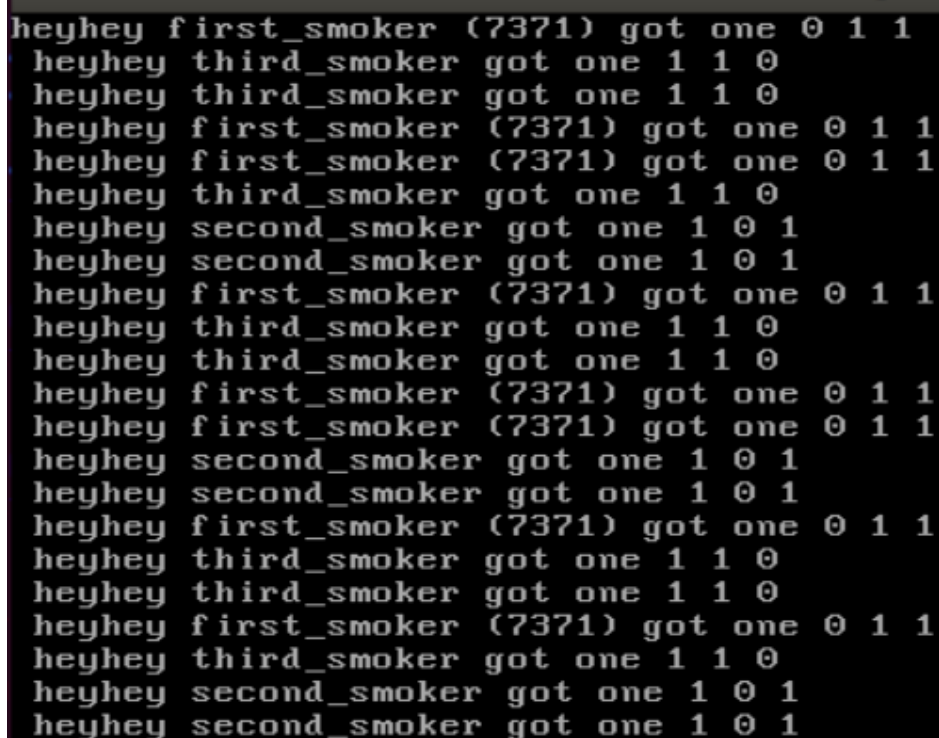
```

```

for(int cnt = 0; cnt < 80000000; cnt++){
semaphore.P();
for(int j = 0; j < 3; j++){
    if(i != j){
        material[j] = 1;
    }else{
        material[j] = 0;
    }
}
semaphore.V();
}
}
}

```

实验结果



```

heyhey first_smoker (7371) got one 0 1 1
heyhey third_smoker got one 1 1 0
heyhey third_smoker got one 1 1 0
heyhey first_smoker (7371) got one 0 1 1
heyhey first_smoker (7371) got one 0 1 1
heyhey third_smoker got one 1 1 0
heyhey second_smoker got one 1 0 1
heyhey second_smoker got one 1 0 1
heyhey first_smoker (7371) got one 0 1 1
heyhey third_smoker got one 1 1 0
heyhey third_smoker got one 1 1 0
heyhey first_smoker (7371) got one 0 1 1
heyhey first_smoker (7371) got one 0 1 1
heyhey second_smoker got one 1 0 1
heyhey second_smoker got one 1 0 1
heyhey first_smoker (7371) got one 0 1 1
heyhey third_smoker got one 1 1 0
heyhey third_smoker got one 1 1 0
heyhey first_smoker (7371) got one 0 1 1
heyhey third_smoker got one 1 1 0
heyhey second_smoker got one 1 0 1
heyhey second_smoker got one 1 0 1

```

分析：可以看到，上面的结果每一个抽烟者在合成烟时，原材料的信息都是正确的

Assignment 3 哲学家就餐问题

子任务1—简单解决方法

任务要求：同学们需要在实验6教程的代码环境下，创建多个线程来模拟哲学家就餐的场景。然后，同学们需要结合信号量来实现理论课教材（参见《操作系统概念》中文第9版第187页的内容）中给出的关于

哲学家就餐问题的简单解决方法。最后，保存结果截图并说说你是怎么做的

注意：

1. 截图结果中不能出现饥饿/死锁，可以通过输出不同哲学家的状态信息，验证你使用教材的方法确实能解决哲学家就餐问题。
2. 请将你学号的后4位包含在其中一个哲学家线程的输出信息中，用作结果截图中的个人信息表征。

实验步骤

1. 创建五个哲学家的线程，设置五个信号量分别表示筷子是否可以获得，哲学家对属于他的两个信号量（筷子）进行P操作，如果都可获得，则输出**eating**，如果其中任意一个得不到，则进行等待，**eating**之后会进行V操作（放下筷子）
2. 为了展示互斥的效果，第一个哲学家的函数与其他略有不同

关键代码（完整代码放置在src/Assignment 3-1）

第一个哲学家

```
void first_philosopher(void* arg){
    while(1){
        for(int cnt = 0; cnt < 200000000; cnt++){
            chopstick[0].P();
            chopstick[4].P();

            printf("The first_philosopher(7371) is eating \n");
            //延长吃饭时间
            for(int cnt = 0; cnt < 100000000; cnt++){
                chopstick[0].V();
                chopstick[4].V();
            }
            printf("The first_philosopher(7371) is full \n");
        }
    }
}
```

其他哲学家

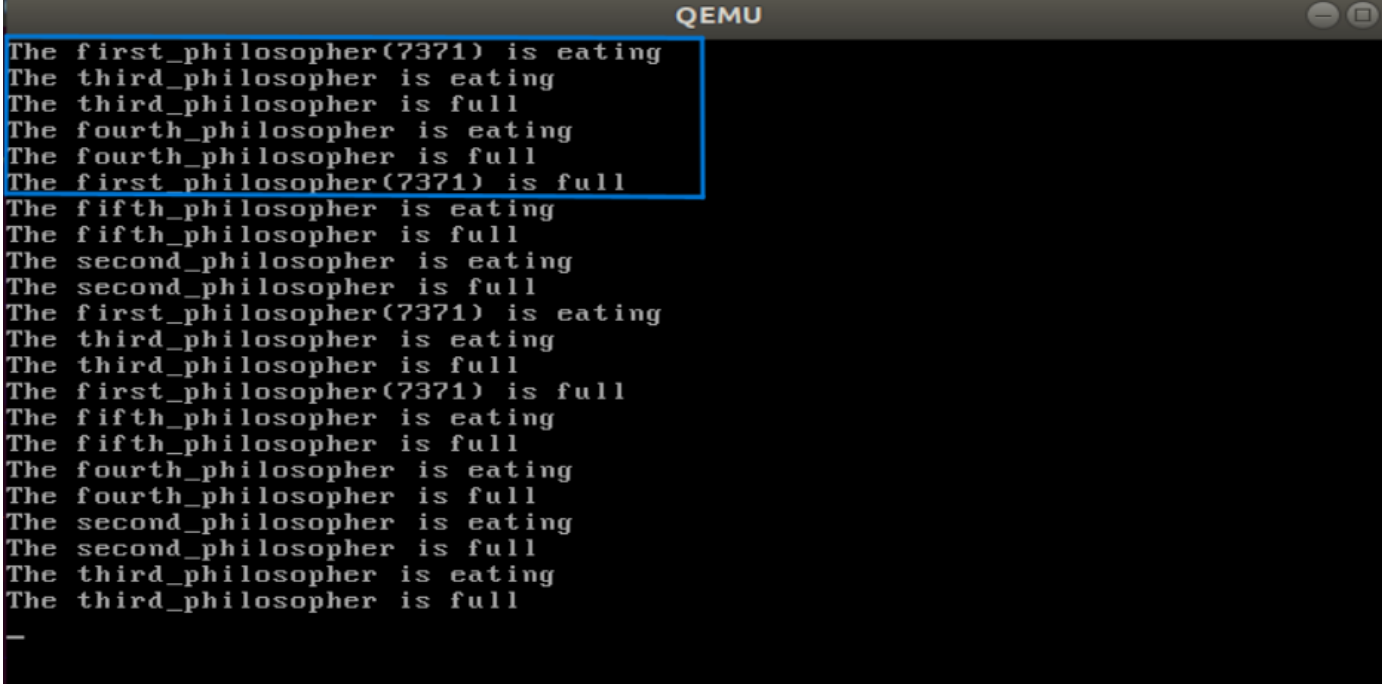
```
void second_philosopher(void* arg){
    while(1){
        for(int cnt = 0; cnt < 200000000; cnt++){
            chopstick[0].P();
            chopstick[1].P();

            printf("The second_philosopher is eating \n");

            chopstick[0].V();
```

```
        chopstick[1].V();  
        printf("The second_philosopher is full \n");  
    }  
}
```

实验结果



```
QEMU  
The first_philosopher(7371) is eating  
The third_philosopher is eating  
The third_philosopher is full  
The fourth_philosopher is eating  
The fourth_philosopher is full  
The first_philosopher(7371) is full  
The fifth_philosopher is eating  
The fifth_philosopher is full  
The second_philosopher is eating  
The second_philosopher is full  
The first_philosopher(7371) is eating  
The third_philosopher is eating  
The third_philosopher is full  
The first_philosopher(7371) is full  
The fifth_philosopher is eating  
The fifth_philosopher is full  
The fourth_philosopher is eating  
The fourth_philosopher is full  
The second_philosopher is eating  
The second_philosopher is full  
The third_philosopher is eating  
The third_philosopher is full  
—
```

分析：从图中可以看到，当第一个哲学家在吃饭时，只有三四哲学家可以吃，第二和第五个哲学家需要等第一个哲学家吃完才可以吃

子任务2—死锁应对策略

任务要求：子任务1的解决方案保证两个相邻的哲学家不能同时进食，但是这种方案可能导致死锁。请同学们描述子任务1解决方法中会导致死锁的场景，并将其复现出来。进一步地，请同学们在下述4种策略中选择1种，解决子任务1中的死锁问题，并在代码中实现。最后，保存结果截图并说说你的实现思路。

死锁复现

实验步骤

只要增大哲学家在拿左右筷子的间隔便可以造成死锁现象的发生

关键代码（完整代码放置在src/Assignment 3-2）

```

void first_philosopher(void* arg){
    while(1){
        for(int cnt = 0; cnt < 200000000; cnt++){
            chopstick[4].P();
            printf("The first_philosopher(7371) got 4 \n");
            for(int cnt = 0; cnt < 200000000; cnt++){
                chopstick[0].P();
                printf("The first_philosopher(7371) got 0 \n");

                printf("The first_philosopher(7371) is eating \n");
                //延长吃饭时间
                for(int cnt = 0; cnt < 100000000; cnt++){
                    chopstick[4].V();
                    chopstick[0].V();
                    printf("The first_philosopher(7371) is full \n");
                }
            }
        }
    }
}

```

实验结果

```

QEMU
Create first_philosopher
Create second_philosopher
Create third_philosopher
Create fourth_philosopher
Create fifth_philosopher
The first_philosopher(7371) got 4
The second_philosopher got 0
The third_philosopher got 1
The fourth_philosopher got 2
The fifth_philosopher got 3

```

分析：从图中可以看到每个哲学家都只拿到了自己左边的筷子，并且都在等待右边筷子，从而造成了死锁。

死锁解决

任务要求：子任务1的解决方案保证两个相邻的哲学家不能同时进食，但是这种方案可能导致死锁。请同学们描述子任务1解决方法中会导致死锁的场景，并将其复现出来。进一步地，请同学们在下述4种策略中选择1种，解决子任务1中的死锁问题，并在代码中实现。最后，保存结果截图并说说你的实现思路。

实验步骤

1. 使用非对称的解决方案，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
2. 一号哲学家：4 0 （拿筷子的顺序） 二号哲学家：1 0 三号哲学家：1 2 四号哲学家：3 2 五号哲学家：3 4

关键代码（完整代码放置在src/Assignment 3-3）

只需修改每个哲学家拿筷子的顺序即可(下面是一个例子)

```
void second_philosopher(void* arg){
    while(1){
        for(int cnt = 0; cnt < 100000000; cnt++){
            chopstick[1].P();
            printf("The second_philosopher got 1 \n");

            for(int cnt = 0; cnt < 100000000; cnt++){
                chopstick[0].P();
                printf("The second_philosopher got 0 \n");

                printf("The second_philosopher is eating \n");

                chopstick[1].V();
                chopstick[0].V();
                printf("The second_philosopher is full \n");
            }
        }
    }
}
```

实验结果

```
Create first_philosopher
Create second_philosopher
Create third_philosopher
Create fourth_philosopher
Create fifth_philosopher
The first_philosopher(7371) got 4
The second_philosopher got 1
The fourth_philosopher got 3
The first_philosopher(7371) got 0
The first_philosopher(7371) is eating
The fourth_philosopher got 2
The fourth_philosopher is eating
The fourth_philosopher is full
The fifth_philosopher got 3
The first_philosopher(7371) is full
The second_philosopher got 0
The second_philosopher is eating
The second_philosopher is full
The third_philosopher got 1
The fifth_philosopher got 4
The fifth_philosopher is eating
The fifth_philosopher is full
The fourth_philosopher got 3
```

分析：从图中可以看到，一号拿了筷子4，二号拿了筷子1，它们将共同竞争筷子0，三号与二号共同竞争筷子1，因为筷子1已经被二号拿走了，所以三号只能等待，此时三号并没有拿任何筷子，所以剩余的四名哲学家必然有一名可以拿到两只筷子，这便解决了死锁问题。

5. 总结(对实验过程中遇到的问题进行总结，可以提出对实验设置的改进意见)

总结：此次实验内容与课本内容联系紧密，理解起来简单许多，操作上也简单不少，不过在线程执行函数时还是遇到了比较棘手的问题，因为cpu执行的速度很快，所以函数里需要进行延迟操作，延迟的时间需要特别讲究，太短起不到效果，太长又会导致线程没有在这个时间片内执行完我需要它执行的指令，最终导致输出错误，我大致能得到：循环 $1e8 \sim 2e8$ 能使得线程在该时间片（ticks = 10）内只执行一次想要的指令。

6. 参考资料清单