



中山大學  
SUN YAT-SEN UNIVERSITY

## 本科生实验报告

实验课程: 操作系统原理实验

任课教师: 刘宁

实验题目: 实验6并发与锁机制

专业名称: 信息与计算科学

学生姓名: 罗弘杰

学生学号: 22336173

实验地点: 实验中心D503

实验时间: 2024/3/15

### Section 1 实验概述

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。

### Section 2 预备知识与实验环境

操作系统, 同步, 互斥, 死锁等理论知识

### Section 3 实验任务

#### 实验任务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

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。

子任务2-自实现锁机制

实验6教程中使用了原子指令 xchg 来实现自旋锁。但这种方法并不是唯一的。例如，x86指令中提供了另外一个原子指令 bts 和 lock 前缀等，这些指令也可以用来实现锁机制。现在，同学们需要结合自己所学的知识，实现一个与指导书实现方式不同的锁机制。最后，尝试用你实现的 锁机制解决“消失的芝士汉堡”问题，保存结果截图并说说你的总体思路。

实验任务2：

Assignment 2 生产者-消费者问题

总要求和任务场景

- 1. 同学们请在下述问题场景A或问题场景B中选择一个，然后在实验6教程的代码环境下创建多个线程来模拟你选择的问题场景。同学们需自行决定每个线程的执行次数，以方便观察临界资源变化为首要原则。
- 2. 请将你学号的后4位包含在其中一个线程的输出信息中，用作结果截图中的个人信息表征。例如，学号为 21319527 的同学选择问题A，并扮演服务生A，则服务生A放入1块蛋糕时，程序输出 "Waiter-A 9527 put a piece of matcha cake in the plate.

问题场景A :宴会蛋糕服务	问题场景B :抽烟者与供应商
<p><b>问题描述：</b>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，男性来宾等待享用抹茶蛋糕，女性来宾等待享用芒果蛋糕。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。</p>	<p><b>问题描述：</b>酒馆的吧台坐着3位抽烟者和1位供应商。每位抽烟者会不停地卷烟并抽掉，但是要卷起并抽掉一支烟，抽烟者需要三种材料：烟草，纸和胶水。三位抽烟者中，第1位拥有烟草，第2位有纸，第3位有胶水。供应商会源源不断地提供3种材料，每次他会将两种随机材料组合放在吧台的桌面上，拥有剩下那种材料的抽烟者，会立刻取走材料卷一根烟抽掉它，然后给供应商发一个完成信号，供应商就会放另外两种不同材料组合在桌面上，这样的过程一直重复。</p>

实验任务3：

问题场景 假设有5位哲学家，他们在就餐时只能思考或者就餐。这些哲学家公用一个圆桌，每个哲学家都坐在一把指定的椅子上。在桌子上放着5根筷子，每两根筷子之间都放着一碗葱油面。

下面 是一些约束条件：

- 1. 当一位哲学家处于思考状态时，他对其他哲学家不会产生影响 当一位哲学家感到饥饿时，他会试图拿起与他相邻的两根筷子
- 2. 一个哲学家一次只能拿起一根筷子，
- 3. 在拿到两根筷子之前不会放下手里的筷子
- 4. 如果筷子在其他哲学家手里，则需等待
- 5. 当一个饥饿的哲学家同时拥有两根筷子时，他会开始吃面
- 6. 吃碗面后的哲学家会同时放下两根筷子，并开始思考

## Section 4 实验步骤与实验结果

该节描述每个实验任务的具体的完成过程，包括思路分析、代码实现与执行、结果展示三个部分，实验任务之间的划分应当清晰明了，实验思路分析做到有逻辑、有条理。

### ----- 实验任务1-----

#### 任务要求：

#### 子任务1-利用自旋锁和信号量实现同步

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。

#### 思路分析：

根据实验资料的指示，分别使用自旋锁和信号量来实现同步

自旋锁：

自旋锁是一种简单的同步原语，用于在多线程环境下保护共享资源。其基本思想是当一个线程尝试获取一个已被其他线程持有的锁时，不是立即放弃CPU并进入等待状态（如睡眠），而是“自旋”在一个循环中，不断地检查锁是否已经释放。这种检查通常通过原子操作完成，确保了操作的完整性。一旦锁变为可用状态，自旋的线程就会立即获取锁并继续执行

缺点：

- 忙等待，消耗处理机时间。
- 可能饥饿。
- 可能死锁

#### 代码分析

原子交换的汇编函数，

```
; void asm_atomic_exchange(uint32 *register, uint32 *memory);
asm_atomic_exchange:
    push ebp
    mov ebp, esp
    pushad

    ;get random number

    mov ebx, [ebp + 4 * 2] ; register
    mov eax, [ebx]         ;
    mov ebx, [ebp + 4 * 3] ; memory
    xchg [ebx], eax        ;
    mov ebx, [ebp + 4 * 2] ; memory
    mov [ebx], eax         ;

    popad
    pop ebp
    ret
```

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

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

### 实验结果：

由于线程调度是轮转调度，所以ta线程会在未执行完就被学生线程抢占，但是由于自旋锁的存在，学生需要等待ta线程执行完成才能进入共享区域，**否则一直在循环检测进入条件（忙等待）**

```
rogers2@ubuntu: ~/os/lab6/src/2/build
File Edit View Search Terminal Help
1+0 records out
512 bytes copied, 0.00104908 s, 488 kB/s
dd if=bootloader.bin of=../run/hd.img bs=512 count=5 seek=
1 conv=notrunc
0+1 records in
0+1 records out
281 bytes copied, 0.000356783 s, 788 kB/s
dd if=kernel.bin of=../run/hd.img bs=512 count=145 seek=6
conv=notrunc
15+1 records in
15+1 records out
8112 bytes (8.1 kB, 7.9 KiB) copied, 0.000409692 s, 19.8 M
B/s
rogers2@ubuntu:~/os/lab6/src/2/build$ make run
qemu-system-i386 -hda ../run/hd.img -serial null -parallel
stdio -no-reboot
WARNING: Image format was not specified for '../run/hd.img
' and probing guessed raw.
Automatically detecting the format is dangerous f
or raw images, write operations on block 0 will be restric
ted.
Specify the 'raw' format explicitly to remove the
restrictions.
[
QEMU
TA: start to announce theory HW, there are 0 theory HW now
TA: oh, I have to go to WC.
TA: Oh, SHIT! There are 10 theory HW
22336173_LHJ_ROGERS: Look what I found!
22336173_LHJ_ROGERS: already finish OS HW. Time to have a break!
```

## 子任务1-利用信号量实现同步

### 思路分析：

共享变量为信号量，为正数的时候表示可用资源的数量，也就是可同时进入执行区的进程数量；

为负数的时候，表示正在阻塞队列里等待资源释放的进程数量。

**P 操作：**当一个进程想要访问一个受信号量保护的资源时，它需要执行 P 操作。这包括：检查信号量的值，如果大于零，则减一并允许进程继续；如果等于零，则进程必须等待（可能被阻塞）直到信号量的值大于零。

**V 操作：**当一个进程完成对资源的访问或释放资源时，它执行 V 操作。这会将信号量的值加一，并唤醒一个或多个（如果有多个进程在等待）等待该信号量的进程。

```

class Semaphore
{
private:
    uint32 counter;
    List waiting; //等待,是一个先进先出的队列结构
    SpinLock semLock;

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

```

```

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)); //放入对尾
        cur->status = ProgramStatus::BLOCKED;

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

```

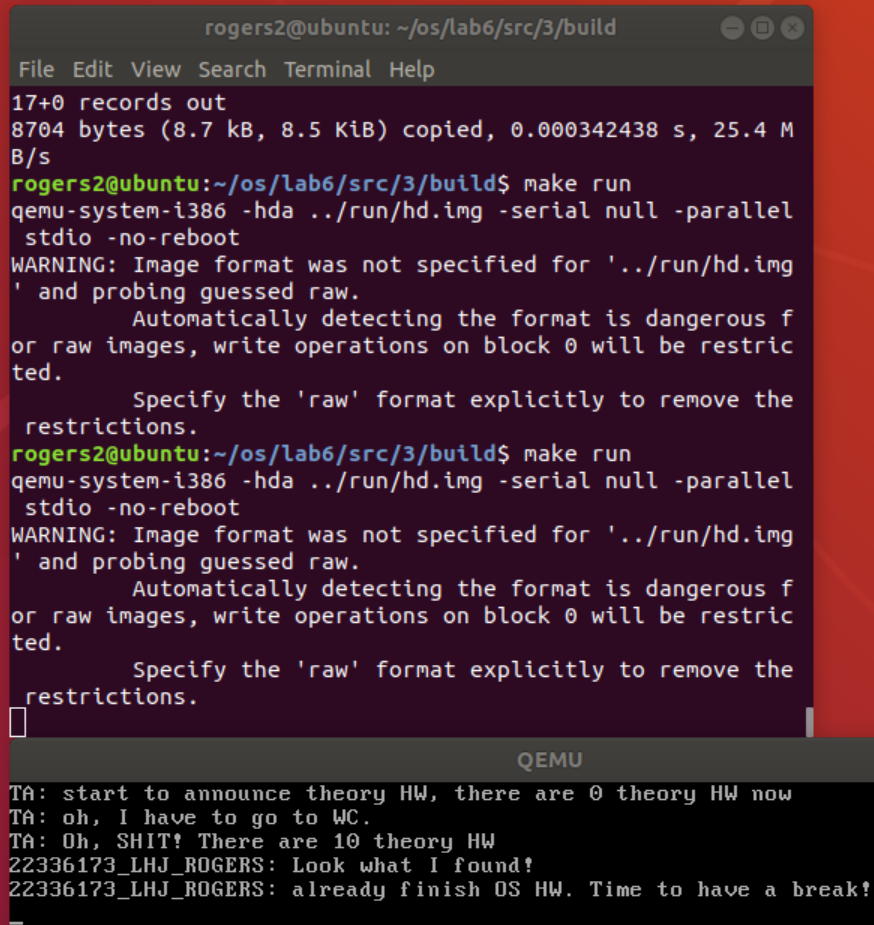
```

void Semaphore::V()
{
    semLock.lock();
    ++counter;
    if (waiting.size())
    {
        PCB *program = ListItem2PCB(waiting.front(), tagInGeneralList);
        waiting.pop_front();
        semLock.unlock();
        programManager.MESA_wakeup(program); //唤醒阻塞进程
    }
    else
    {
        semLock.unlock();
    }
}

```

```
}
```

## 实验结果:



```
rogers2@ubuntu: ~/os/lab6/src/3/build
File Edit View Search Terminal Help
17+0 records out
8704 bytes (8.7 kB, 8.5 KiB) copied, 0.000342438 s, 25.4 MB/s
rogers2@ubuntu:~/os/lab6/src/3/build$ make run
qemu-system-i386 -hda ../run/hd.img -serial null -parallel
stdio -no-reboot
WARNING: Image format was not specified for '../run/hd.img'
and probing guessed raw.
Automatically detecting the format is dangerous for
raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the
restrictions.
rogers2@ubuntu:~/os/lab6/src/3/build$ make run
qemu-system-i386 -hda ../run/hd.img -serial null -parallel
stdio -no-reboot
WARNING: Image format was not specified for '../run/hd.img'
and probing guessed raw.
Automatically detecting the format is dangerous for
raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the
restrictions.
[ ]

QEMU
TA: start to announce theory HW, there are 0 theory HW now
TA: oh, I have to go to WC.
TA: Oh, SHIT! There are 10 theory HW
22336173_LHJ_ROGERS: Look what I found!
22336173_LHJ_ROGERS: already finish OS HW. Time to have a break!
-
```

## 子任务2-自实现锁机制

实验6教程中使用了原子指令 `xchg` 来实现自旋锁。但这种方法并不是唯一的。例如，x86指令中提供了另外一个原子指令 `bts` 和 `lock` 前缀等，这些指令也可以用来实现锁机制。现在，同学们需要结合自己所学的知识，实现一个与指导书实现方式不同的锁机制。最后，尝试用你实现的锁机制解决“消失的芝士汉堡”问题，保存结果截图并说说你的总体思路。

### 代码修改

实际上，`key`和`bolt`不是交换的关系，是`bolt`若为0，则线程可以打开锁进入执行区域；同时`key`总是为1，否则就不会进入循环遇到交换的汇编函数；于是代码逻辑为：**使用`lock`前缀的`bts`指令，判断内存中的值是否为1，如果为1则CF设置为1,并将内存中的值设置为1，否则cf为0**。同时将`cf`的值存入`key`，那么若`bolt`为0，则`key`为0，`bolt`改为1，实现了互斥的功能。

```
; void asm_atomic_exchange(uint32 *register, uint32 *memeory); 检测memeor
asm_atomic_exchange:
    push ebp
    mov ebp, esp
    pushad

    xor eax, eax
    mov ebx, [ebp + 4 * 3] ; memory
    ;使用lock前缀的bts指令，判断内存中的值是否为0，如果为0则CF设置为1,并将内存中的值设置为1
    lock bts [ebx], 0
    ;把cf的值存入eax
    setc al
    ;将eax的值存入ebx指向的内存中
    mov ebx, [ebp + 4 * 2] ; memory
    mov [ebx], eax

    popad
    pop ebp
    ret
```

### 实验结果：

类似之前的结果



```
rogers2@ubuntu: ~/os/lab6/src/2/build
File Edit View Search Terminal Help
1+0 records out
512 bytes copied, 0.000213769 s, 2.4 MB/s
dd if=bootloader.bin of=../run/hd.img bs=512 count=5 seek=
1 conv=notrunc
0+1 records in
0+1 records out
281 bytes copied, 0.000149629 s, 1.9 MB/s
dd if=kernel.bin of=../run/hd.img bs=512 count=145 seek=6
conv=notrunc
15+1 records in
15+1 records out
8116 bytes (8.1 kB, 7.9 KiB) copied, 0.000541882 s, 15.0 M
B/s
rogers2@ubuntu:~/os/lab6/src/2/build$ make run
qemu-system-i386 -hda ../run/hd.img -serial null -parallel
stdio -no-reboot
WARNING: Image format was not specified for '../run/hd.img
' and probing guessed raw.
Automatically detecting the format is dangerous f
or raw images, write operations on block 0 will be restric
ted.
Specify the 'raw' format explicitly to remove the
restrictions.
```

```
QEMU
TA: start to announce theory HW, there are 0 theory HW now
TA: oh, I have to go to WC.
TA: Oh, SHIT! There are 10 theory HW
22336173_LHJ_ROGERS: Look what I found!
22336173_LHJ_ROGERS: already finish OS HW. Time to have a break!
```

## ----- 实验任务2 -----

任务要求:

## 问题场景B :抽烟者与供应商

**问题描述：** 酒馆的吧台坐着3位抽烟者和1位供应商。每位抽烟者会不停地卷烟并抽掉，但是要卷起并抽掉一支烟，抽烟者需要三种材料：烟草，纸和胶水。三位抽烟者中，第1位拥有烟草，第2位有纸，第3位有胶水。供应商会源源不断地提供3种材料，每次他会将两种随机材料组合放在吧台的桌面上，拥有剩下那种材料的抽烟者，会立刻取走材料卷一根烟抽掉它，然后给供应商发一个完成信号，供应商就会放另外两种不同材料组合在桌面上，这样的过程一直重复。

### 子任务1

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，从而无法达到预期的运行结果。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）

#### 思路分析：

```
/*
    如果不使用互斥手段来处理，当其中一个线程贪婪地抢夺不属于自己的资源，容易导致死锁
    . But, if either of them does wake up and consume a resource, that will
    prevent the third thread from begin able to smoke and thus also prevent the agent
    from waking up to deliver additional resources. If this happens, the system is
    deadlocked; no thread will be able to make further progress.
    --from https://github.com/Yuanjiezhao/Cigarette-Smokers-Problem
*/
void producer(void *arg)
{
    int rand = 0;
    //static const int choices[3] = {TOBACCO|MATCH, PAPER|TOBACCO, MATCH|PAPER};
    //static const int matching[3] = {PAPER, TOBACCO, MATCH};
    while(1){
        //printf("producer thread running!\n");
        while(SIGNAL){
```

```

int rand = (rand + 1) % 3;

if (rand == 0){
    TOBACCO_NUM++;
    MATCH_NUM++;
    printf("offer material for smoker_have_paper %d\n", rand+1);
}else if (rand == 1){
    PAPER_NUM++;
    TOBACCO_NUM++;
    printf("offer material for smoker_have_match%d\n", rand+1);
}else{
    MATCH_NUM++;
    PAPER_NUM++;
    printf("offer material for smoker_have_tobacco %d\n", rand+1);
}
}
}

void smoker1(void *arg)
{
    bool TOBACCO_FLAG = false;
    bool MATCH_FLAG = false;
    while(1){
        //printf("smoker1 thread running!\n");
        if (TOBACCO_NUM){
            TOBACCO_NUM--;
            TOBACCO_FLAG = true;
            printf("smoker_have_paper: GET TOBACCO!\n");
        }
        if (MATCH_NUM){
            MATCH_NUM--;
            MATCH_FLAG = true;
            printf("smoker_have_paper: GET MATCH!\n");
        }
        if (TOBACCO_FLAG && MATCH_FLAG){
            printf("smoker_have_paper: I am smoking!\n");
            SIGNAL++;
            //printf("waiting for producer!\n");
            TOBACCO_FLAG = false;
            MATCH_FLAG = false;
        }
    }
}
}

```

如果每个抽烟者一见到有自己所需要的资源就获得该资源，而不是凑齐了才上的话，就会出现竞争与死锁。

### 实验结果

如图，在结尾处，两个线程分别获得一个资源，造成死锁

```
SeaBIOS (version 1.10.2-1ubuntu1)

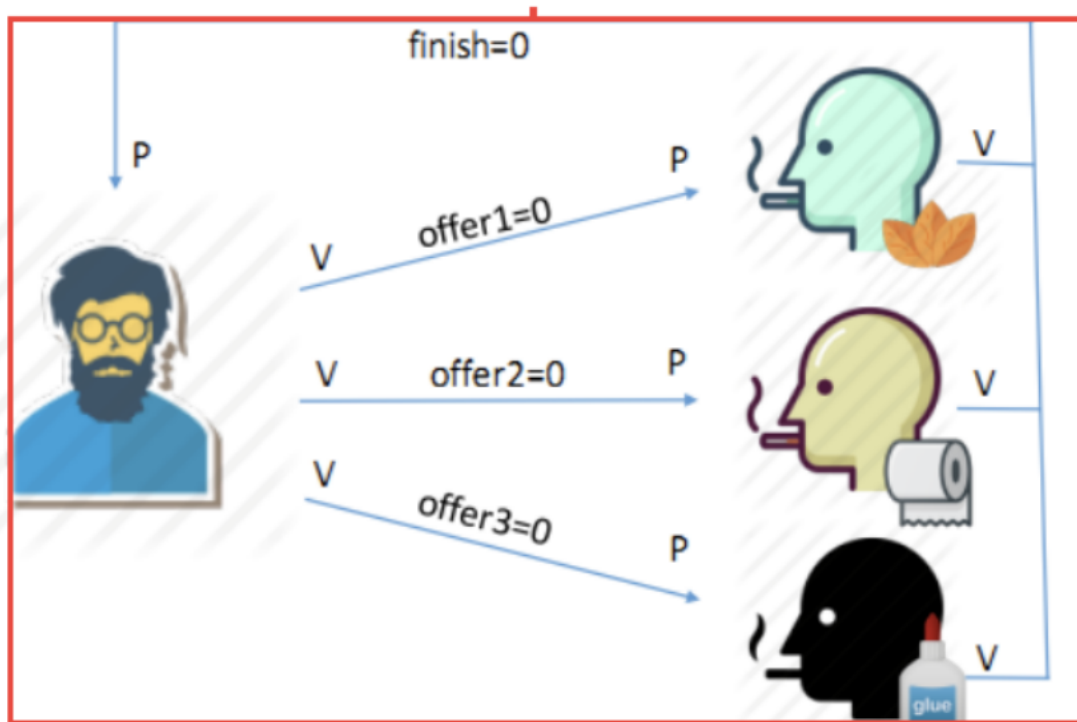
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD+07ECDDDD C980

Booting from Hard Disk...
22336173LHJ offer material for smoker_have_paper 1
smoker_have_paper: GET TOBACCO!
smoker_have_paper: GET MATCH!
smoker_have_paper: I am smoking!
22336173LHJ offer material for smoker_have_match2
smoker_have_paper: GET TOBACCO!
smoker_have_match: GET PAPER!
```

## 子任务2:

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。

供应者与三个抽烟者分别是同步关系。由于抽烟者无法同时满足两个或以上的抽烟者，三个抽烟者对抽烟这个动作互斥（或由三个抽烟者轮流抽烟得知）



## 思路和代码分析

使用4个信号量来实现这个同步问题，其中三个是为了抽烟者设定，一个是为了供应商设定。

```
Semaphore SMOKER_MATCH;
Semaphore SMOKER_PAPER;
Semaphore SMOKER_TOBACCO;
Semaphore AGENT; //供应商的信号量，会被初始化为1，表示可以开始放物资

void producer(void *arg)
{
    AGENT.initialize(1); // 1 means the agent is available
    int rand = 0;
    while(1){
        //printf("producer thread running!\n");
        AGENT.P();
        int rand = (rand + 1) % 3;
        printf("offer material %d\n", rand + 1);
        if (rand == 0){
            SMOKER_MATCH.V(); //smoker1
        } else if (rand == 1){
            SMOKER_PAPER.V(); //smoker2
        } else{
            SMOKER_TOBACCO.V(); //smoker3
        }
    }
}

void smoker1(void *arg)
{
    while(1){
        SMOKER_MATCH.P(); //smoker1
        printf("smoker1: I am smoking!\n");
        AGENT.V();
    }
}

void smoker2(void *arg)
{
    while(1){
        //printf("smoker2 thread running!\n");
        SMOKER_PAPER.P(); //smoker2
        printf("smoker2: I am smoking!\n");
        AGENT.V();
    }
}

void smoker3(void *arg)
{
    while(1){
        //printf("smoker3 thread running!\n");
        SMOKER_TOBACCO.P(); //smoker3
    }
}
```

```

        printf("smoker3: I am smoking!\n");
        AGENT.V();
    }
}

```

## 成果展示：

可以看到抽烟和供应的循环可以一直继续下去。

```

QEMU
smoker3: I am smoking!
22336173ΓCöΓCöLHJ offer material 1
smoker1: I am smoking!
22336173ΓCöΓCöLHJ offer material 2
smoker2: I am smoking!
22336173ΓCöΓCöLHJ offer material 3
smoker3: I am smoking!
22336173ΓCöΓCöLHJ offer material 1
smoker1: I am smoking!
22336173ΓCöΓCöLHJ offer material 2
smoker2: I am smoking!
22336173ΓCöΓCöLHJ offer material 3
smoker3: I am smoking!
22336173ΓCöΓCöLHJ offer material 1
smoker1: I am smoking!
22336173ΓCöΓCöLHJ offer material 2
smoker2: I am smoking!
22336173ΓCöΓCöLHJ offer material 3
smoker3: I am smoking!
22336173ΓCöΓCöLHJ offer material 1
smoker1: I am smoking!
22336173ΓCöΓCöLHJ offer material 2
smoker2: I am smoking!
22336173ΓCöΓCöLHJ offer material 3

```

## ----- 实验任务3 -----

### 子任务1：

#### 子任务1-简单解决方法

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教材（参见《操作系统概念》中文第9版187页）中给出的关于哲学家就餐的简单解决办法。最后，保存结果截图并说说你是怎么做的。

注意：

可以通过输出不同哲学家的状态信息，验证你使用教程的方法确实能解决哲学家就餐问题。

请将你的学号的后四位包含在其中一个哲学家线程的输出信息中，用作结果截图的个人信息表征

### 代码分析

根据教科书，可以使用5个信号量来表示5根筷子。

```

Semaphore CH1; //5个筷子信号量，将会被初始化为1
Semaphore CH2;
Semaphore CH3;
Semaphore CH4;
Semaphore CH5;

```

```

//模拟5个哲学家，5根筷子吃饭问题,使用5个信号量对应5根筷子，可能会发生死锁
void PH1(void *arg)
{
    int time = 0;
    while (1)
    {
        CH1.P();
        CH2.P();
        int wait = 0xffffffff; //吃饭时间
        printf("22336173_LHJ is eating: %d\n", ++time); //输出个人id
        while (wait--)
            ;
        CH1.V();
        CH2.V();
        wait = 0xffffffff; //思考时间
        //printf("22336173_LHJ is thinking\n");
        while (wait--)
            ;
    }
}

void PH2(void *arg)
{
    int time = 0;
    while (1)
    {
        CH2.P();
        CH3.P();
        printf("PH2 is eating: %d\n", ++time);
        int wait = 0xffffffff;
        while (wait--)
            ;
        CH2.V();
        CH3.V();
        //printf("PH2 is thinking\n");
        wait = 0xffffffff;
        while (wait--)
            ;
    }
}

```

### 结果展示:

在这里我的学号表示第一个哲学家，最后的数字表示吃饭的次数。可以看到基本上所有哲学家都可以吃到事物，但是第一个拿起筷子的哲学家可能会**吃更多次**，最后稳定在其他人的两倍

```
Booting from Hard Disk...
22336173_LHJ is eating: 1
PH3 is eating: 1
22336173_LHJ is eating: 2
22336173_LHJ is eating: 3
PH5 is eating: 1
PH4 is eating: 1
PH3 is eating: 2
PH2 is eating: 1
22336173_LHJ is eating: 4
PH5 is eating: 2
PH4 is eating: 2
PH3 is eating: 3
PH2 is eating: 2
22336173_LHJ is eating: 5
22336173_LHJ is eating: 6
PH5 is eating: 3
PH4 is eating: 3
PH3 is eating: 4
PH2 is eating: 3
22336173_LHJ is eating: 7
22336173_LHJ is eating: 8
PH5 is eating: 4
PH4 is eating: 4
```

过了一段时间:

```
H2 is eating: 7
2336173_LHJ is eating: 15
2336173_LHJ is eating: 16
H5 is eating: 8
H4 is eating: 8
H3 is eating: 9
H2 is eating: 8
2336173_LHJ is eating: 17
H5 is eating: 9
H4 is eating: 9
H3 is eating: 10
H2 is eating: 9
2336173_LHJ is eating: 18
2336173_LHJ is eating: 19
H5 is eating: 10
H4 is eating: 10
H3 is eating: 11
H2 is eating: 10
2336173_LHJ is eating: 20
H5 is eating: 11
H4 is eating: 11
H3 is eating: 12
```

## 子任务2-死锁应对策略（选做）

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是怎么做的。

- 策略1：利用抽屉原理（鸽笼原理）。即允许最多4个哲学家同时做在桌子上，保证至少有1位哲学家能够吃到面。
- 策略2：利用AND信号量机制或信号量保护机制。仅当哲学家的左右两支筷子都可用时，才允许他拿起筷子就餐（或者说哲学家必须在临界区内拿起两根筷子）。
- 策略3：使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。
- 策略4：基于管程的解决方法。参加《操作系统概念》中文第9版190-191页，采用类似策略2的思路，定义管程来控制筷子的分布，控制哲学家拿起筷子和放下筷子的顺序，确保两个相邻的哲学家不会同时就餐。



## 死锁模拟

死锁情况：每一个哲学家同时拿起左边的筷子，这时筷子没有剩余，产生循环等待，造成死锁。

代码分析：

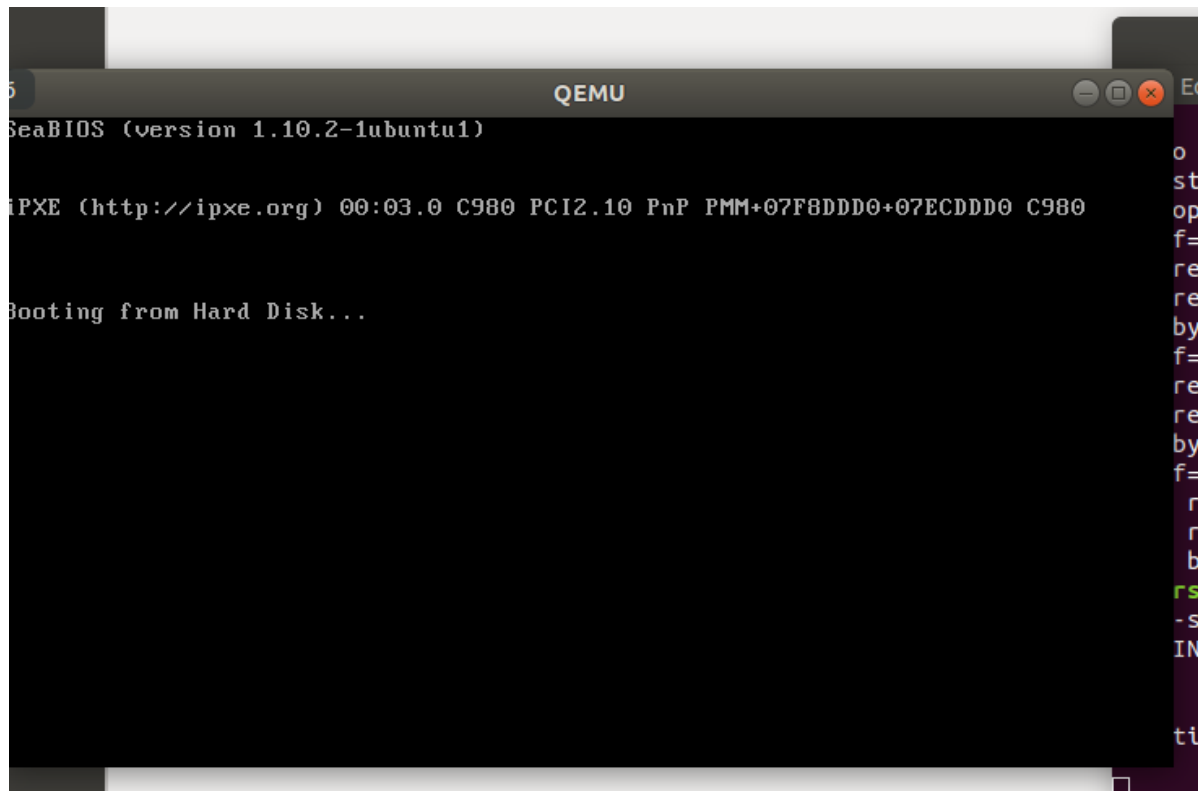
在时间片轮转调度模拟进程切换比较困难，在这里拿起左边筷子就等待来模拟切换，最终导致循环等待产生死锁，每个哲学家手里都有一个筷子。

```
//模拟5个哲学家，5根筷子吃饭问题,使用5个信号量对应5根筷子，可能会发生死锁
void PH1(void *arg)
{
    int time = 0;
    while (1)
    {
        CH1.P();
        int wait = 0xffffffff; //拿起左边筷子等待，当进程切换后，会发现右边筷子被别人拿走了，

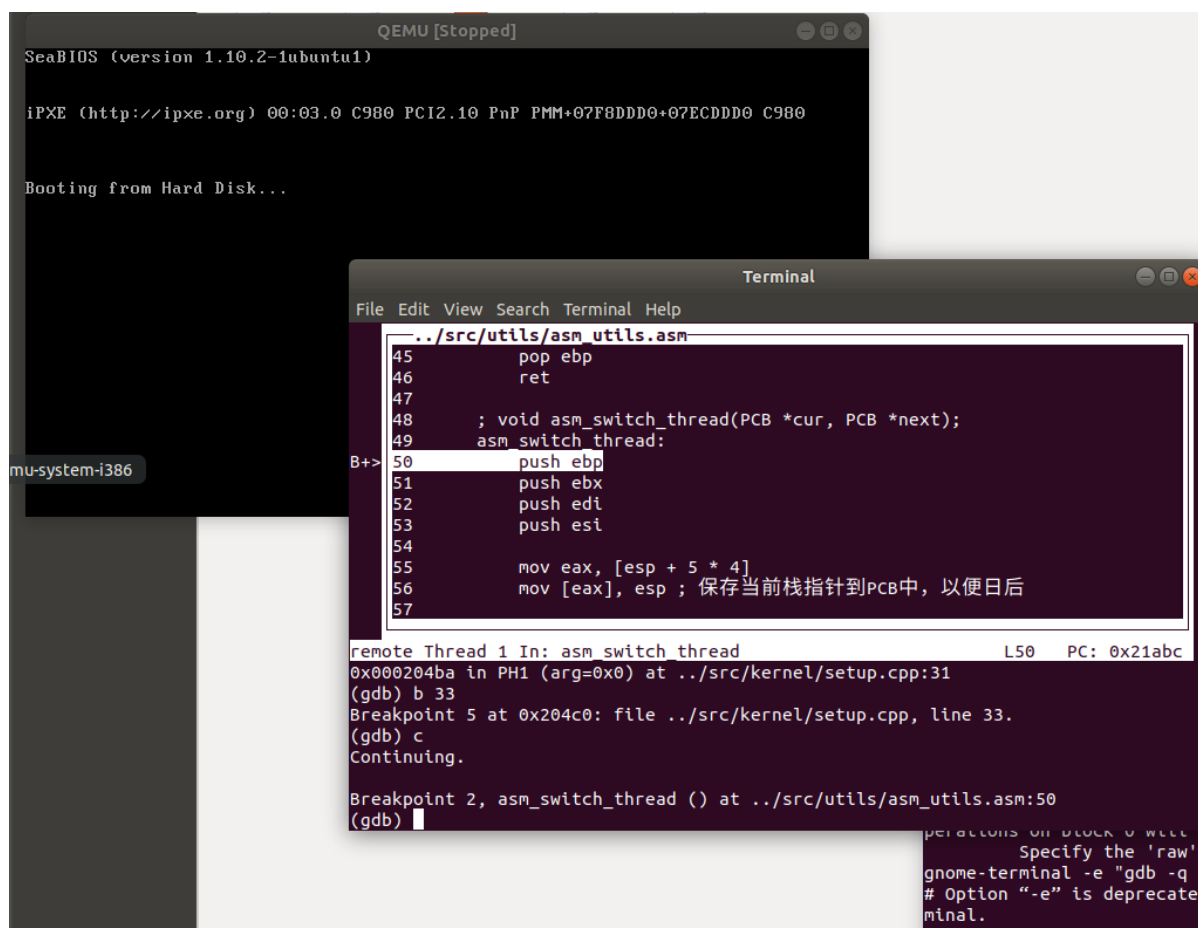
        while (wait--)
            ;
        CH2.P();
        printf("22336173_LHJ is eating: %d\n", ++time);
        while (wait--)
            ;
        CH1.V();
        CH2.V();
        wait = 0xffffffff;
        while (wait--)
            ;
        //printf("22336173_LHJ is thinking\n");
    }
}
```

结果展示：

可以看到没有人能吃饭



进入调试，发现，在拿起左筷子进入循环后，会产生时间片用完导致的进程切换，最终发生死锁。



## 死锁解决策略

### 思路和代码分析:

使用鸽笼原理，限制拿起筷子的只有4个人，保证不会产生循环等待：

具体地，可以使用一个EATING\_NUM的信号量，初始化为4，来表示只能同时4个人拿筷子；

```
//使用鸽笼原理解决哲学家吃饭问题，避免产生死锁

Semaphore EATING_NUM;
EATING_NUM.initialize(4); //最多只能有4个哲学家同时吃饭

//模拟5个哲学家，5根筷子吃饭问题，使用5个信号量对应5根筷子
void PH1(void *arg)
{
    int time = 0;
    while (1)
    {
        EATING_NUM.P();
        CH1.P();
        int wait = 0xffffffff;
        while (wait--)
            ;
        CH2.P();
        printf("22336173_LHJ is eating: %d\n", ++time);
        while (wait--)
            ;
        CH1.V();
        CH2.V();
        EATING_NUM.V();
        wait = 0xffffffff;
        while (wait--)
            ;
        //printf("22336173_LHJ is thinking\n");
    }
}
```

结果展示:

