



本科生实验报告

实验课程：_____操作系统原理实验_____

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：_____

学生学号：_____

实验成绩：_____

报告时间：_____2025 年 5 月 8 日_____

1. 实验要求

Assignment 1 代码复现题

1.1 代码复现

在本章中，我们已经实现了自旋锁和信号量机制。现在，同学们需要复现教程中的自旋锁和信号量的实现方法，并用分别使用二者解决一个同步互斥问题，如消失的芝士汉堡问题。最后，将结果截图并说说你是怎么做的。

1.2 锁机制的实现

我们使用了原子指令 `xchg` 来实现自旋锁。但是，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，实现一个与本教程的实现方式不完全相同的锁机制。最后，测试你实现的锁机制，将结果截图并说说你是怎么做的。

Assignment 2 生产者-消费者问题

2.1 Race Condition

同学们可以任取一个生产者-消费者问题，然后在本教程的代码环境下创建多个线程来模拟这个问题。在 2.1 中，我们不会使用任何同步互斥的工具。因此，这些线程可能会产生冲突，进而无法产生我们预期的结果。此时，同学们需要将这个产生错误的场景呈现出来。最后，将结果截图并说说你是怎么做的。

2.2 信号量解决方法

使用信号量解决上述你提出的生产者-消费者问题。最后，将结果截图并说说你是怎么做的。

Assignment 3 哲学家就餐问题

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。

3.1 初步解决方法

同学们需要在本教程的代码环境下，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中给出的关于哲学家就餐问题的方法。最后，将结果截图并说说你是怎么做的。

3.2 死锁解决方法

虽然 3.1 的解决方案保证两个邻居不能同时进食，但是它可能导致死锁。现在，同学们需要想办法将死锁的场景演示出来。然后，提出一种解决死锁的方法并实现之。最后，将结果截图并说说你是怎么做的。

2. 实验过程

Assignment 1 代码复现题

自旋锁是一种忙等待的同步机制，当线程尝试获取锁时，如果锁已被占用，线程会循环检查锁的状态，直到锁被释放。自旋锁的实现是通过一系列特殊的 CPU 指令，这类指令在执行时不会被中断，从而具有原子性质。

信号量是一种更通用的同步工具，其在本次实验的基础上，通过 P 操作和 V 操作来控制资源的访问，可以避免忙等待。

根据实验指导书，可以实现自旋锁的数据结构和相关函数，包括锁的初始化、获取和释放，并在消失的芝士汉堡问题上得到正确性验证。

信号量也可以根据实验指导书实现，在消失的芝士汉堡问题上改为：妈妈只有在晾完衣服后才将信号量进行 V 操作，儿子对信号量进行 P 操作进行等待。

自旋锁也可以用 lock bts 指令实现，该指令原子性检查内存字节的某一位，若为 0 则将其置位并将 cf 置 1，只需反复调用该指令并检查 cf 即可实现 lock。

Assignment 2 生产者-消费者问题

采用 2 生产者+1 消费者的模式，生产者每隔一段时间生产一个产品，并将其放到空的货柜上。生产者将在产品放到货柜上之前会先将货柜对应位置标记，表示消费者可以取走商品，且其他生产者不能将商品放到该货柜上。在没有互斥锁和信号量的情况下，可能出现：

1. 多个生产者将产品放到同一个货柜上，因为有可能在一个生产者修改标记之前，另一个生产者也检测到该货柜为空并尝试写入数据，导致数据覆盖。
2. 消费者在生产者将数据写入之前就取出数据，可能取出空数据。
3. 消费者忙等待，消耗 CPU 资源。

为了解决这种情况，考虑引入互斥锁和信号量，互斥锁用于保护货柜的读写和访问，确保同一时刻只有

一个生产者或消费者能够操作货柜，防止数据竞争和覆盖。信号量用于同步生产者和消费者之间的操作，协调生产和消费的速度。

Assignment 3 哲学家就餐问题

初步解决方法是为每根筷子设置一个信号量，哲学家用餐前首先等待左侧筷子，再等待右侧筷子，用餐后再依次将他们释放。

这样做的问题在于如果每个哲学家都拿到了左手边的筷子，此时每个人都在等待右边的筷子被释放，这样就造成了死锁，其中一种解决方式是限制最大就餐人数为总人数-1，即为 4 人。添加一个就餐人数限制的信号量即可解决。

3. 关键代码

Assignment 1 代码复现题

用互斥锁实现：

```
SpinLock aLock;
int cheese_burger = 0;

void a_mother(void* arg) {
    aLock.lock();
    int delay = 0;

    printf("mother: start to make cheese burger, there are %d cheese burger now\n",
cheese_burger);
    // make 10 cheese_burger
    cheese_burger += 10;

    printf("mother: oh, I have to hang clothes out.\n");
    // hanging clothes out
    delay = 0xffffffff;
    while (delay)
        --delay;
    // done

    printf("mother: Oh, Jesus! There are %d cheese burgers\n", cheese_burger);
    aLock.unlock();
    asm_halt();
}
```

```

}

void a_naughty_boy(void*) {
    aLock.lock();
    printf("boy   : Look what I found!\n");
    // eat all cheese_burgers out secretly
    cheese_burger -= 10;
    // run away as fast as possible
    aLock.unlock();
}

```

用信号量实现：

```

Semaphore s;

void a_mother(void* arg) {
    // make burger and hang out clothes
    s.V();
    asm_halt();
}

void a_naughty_boy(void*) {
    s.P();
    printf("boy   : Look what I found!\n");
    cheese_burger -= 10;
}

```

用 bts 和 lock 前缀实现锁：

```

; void asm_acquire_lock(uint32 *lock);
asm_acquire_lock:
    push ebp
    mov ebp, esp
    pushad

    mov ebx, [ebp + 8] ; 获取 lock 参数指针
.try_lock:
    lock bts dword [ebx], 0 ; 原子地测试并设置锁的第 0 位
    jc .try_lock           ; 如果进位标志被设置，表示锁已被占用，继续尝试

    popad
    pop ebp
    ret

; void asm_release_lock(uint32 *lock);
asm_release_lock:
    push ebp
    mov ebp, esp
    pushad

```

```
mov ebx, [ebp + 8] ; 获取 lock 参数指针
btr dword [ebx], 0 ; 重置锁的第 0 位, 释放锁

popad
pop ebp
ret
```

对应的 lock 调用方法:

```
void SpinLock::lock() {
    asm_acquire_lock(&bolt);
}

void SpinLock::unlock() {
    asm_release_lock(&bolt);
}
```

Assignment 2 生产者-消费者问题

在没有互斥锁和信号量的情况下, 会出现线程冲突:

```
int item[10];
bool available[10];

void producer(void* arg) {
    int producer_id = *(int*)arg;
    int item_id = 0;
    while (true) {
        for (int i = 0; i < 0xffffffff; i++)
            ; // 生产一件商品
        bool found = false;
        while (!found) {
            for (int i = 0; i < 10; i++) {
                if (!available[i]) {
                    available[i] = true;
                    for (int j = 0; j < 0xffffffff; j++)
                        ; // 把商品放到货柜
                    item[i] = ++item_id;
                    printf("producer %d: put item %d on %d\n", producer_id, item[i], i);
                    found = true;
                    break;
                }
            }
        }
    }
}

void consumer(void*) {
```

```

while (true) {
    for (int i = 0; i < 10; i++) {
        if (available[i]) {
            printf("consumer: get item %d\n", item[i]);
            available[i] = false;
            item[i] = 0;
        }
    }
}
}
}

```

引入互斥锁和信号量

```

int item[10];
bool available[10];
SpinLock aLock[10];
Semaphore item_count, empty_count;

void producer(void* arg) {
    int producer_id = *(int*)arg;
    int item_id = 0;
    while (true) {
        for (int i = 0; i < 0xffffffff; i++)
            ; // 生产一件商品
        bool found = false;
        empty_count.P();
        for (int i = 0; i < 10; i++) {
            if (!available[i]) {
                aLock[i].lock();
                available[i] = true;
                for (int j = 0; j < 0xffffffff; j++)
                    ; // 把商品放到货柜
                item[i] = ++item_id;
                printf("producer %d: put item %d on %d\n", producer_id, item[i], i);
                aLock[i].unlock();
                item_count.V();
                found = true;
                break;
            }
        }
    }
}

void consumer(void*) {
    while (true) {
        item_count.P();
        for (int i = 0; i < 10; i++) {
            if (available[i]) {
                aLock[i].lock();

```



```

        printf("consumer: get item %d\n", item[i]);
        available[i] = false;
        item[i] = 0;
        aLock[i].unlock();
    }
}
empty_count.V();
}
}

```

Assignment 3 哲学家就餐问题

没有限制最大就餐人数，可能出现死锁：

```

Semaphore chopstick[5];

void philosopher(void* arg) {
    int philosopher_id = *(int*)arg;
    while (true) {
        printf("philosopher %d: think\n", philosopher_id);
        for (int i = 0; i < 0xffffffff; i++)
            ; // 思考
        chopstick[philosopher_id].P();
        printf("philosopher %d: pick up left chopstick\n", philosopher_id);
        for (int i = 0; i < 0xffffffff; i++)
            ; // 模拟延迟以复现死锁
        chopstick[(philosopher_id + 1) % 5].P();
        printf("philosopher %d: pick up right chopstick\n", philosopher_id);
        printf("philosopher %d: eat\n", philosopher_id);
        for (int i = 0; i < 0xffffffff; i++)
            ; // 吃饭完毕
        chopstick[philosopher_id].V();
        chopstick[(philosopher_id + 1) % 5].V();
        printf("philosopher %d: put down chopsticks\n", philosopher_id);
    }
}

```

加入限制最大就餐人数的信号量：

```

Semaphore chopstick[5];
Semaphore seat;

void philosopher(void* arg) {
    int philosopher_id = *(int*)arg;
    while (true) {
        printf("philosopher %d: think\n", philosopher_id);
        for (int i = 0; i < 0xffffffff; i++)
            ; // 思考
    }
}

```

```

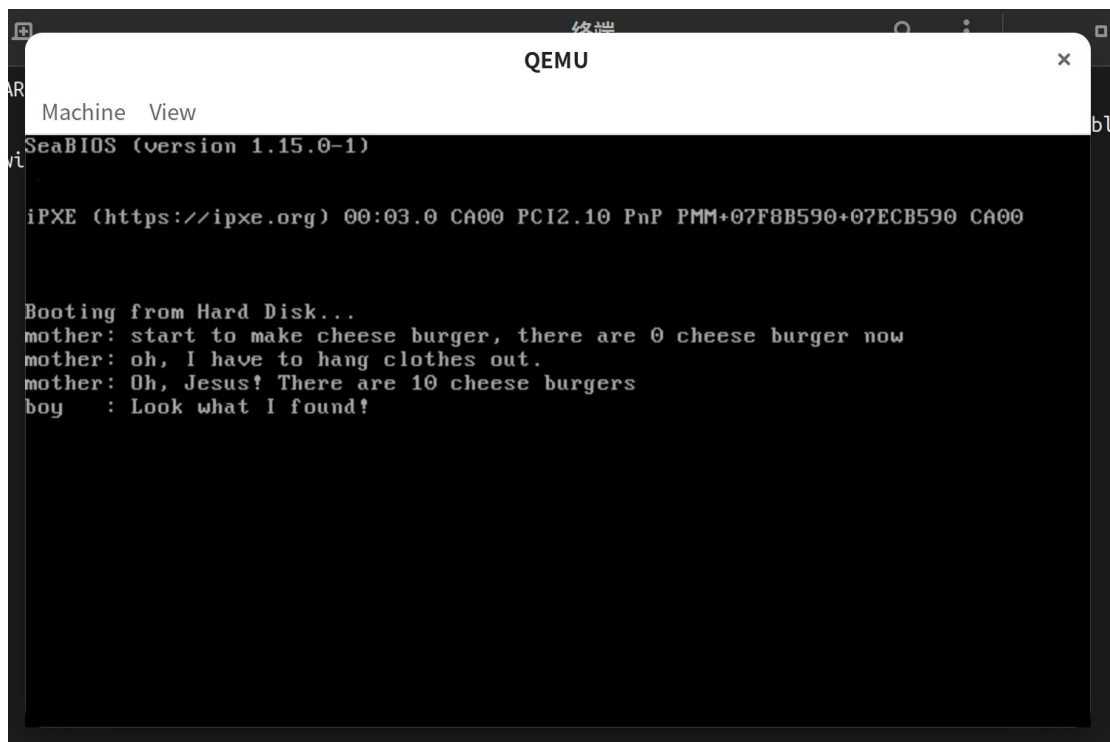
seat.P();
printf("philosopher %d: get a seat\n", philosopher_id);
chopstick[philosopher_id].P();
printf("philosopher %d: pick up left chopstick\n", philosopher_id);
for (int i = 0; i < 0xffffffff; i++)
    ; // 模拟延迟以复现死锁
chopstick[(philosopher_id + 1) % 5].P();
printf("philosopher %d: pick up right chopstick\n", philosopher_id);
printf("philosopher %d: eat\n", philosopher_id);
for (int i = 0; i < 0xffffffff; i++)
    ; // 吃饭完毕
chopstick[philosopher_id].V();
chopstick[(philosopher_id + 1) % 5].V();
printf("philosopher %d: put down chopsticks\n", philosopher_id);
seat.V();
printf("philosopher %d: leave the table\n", philosopher_id);
}
}

```

不会出现死锁。

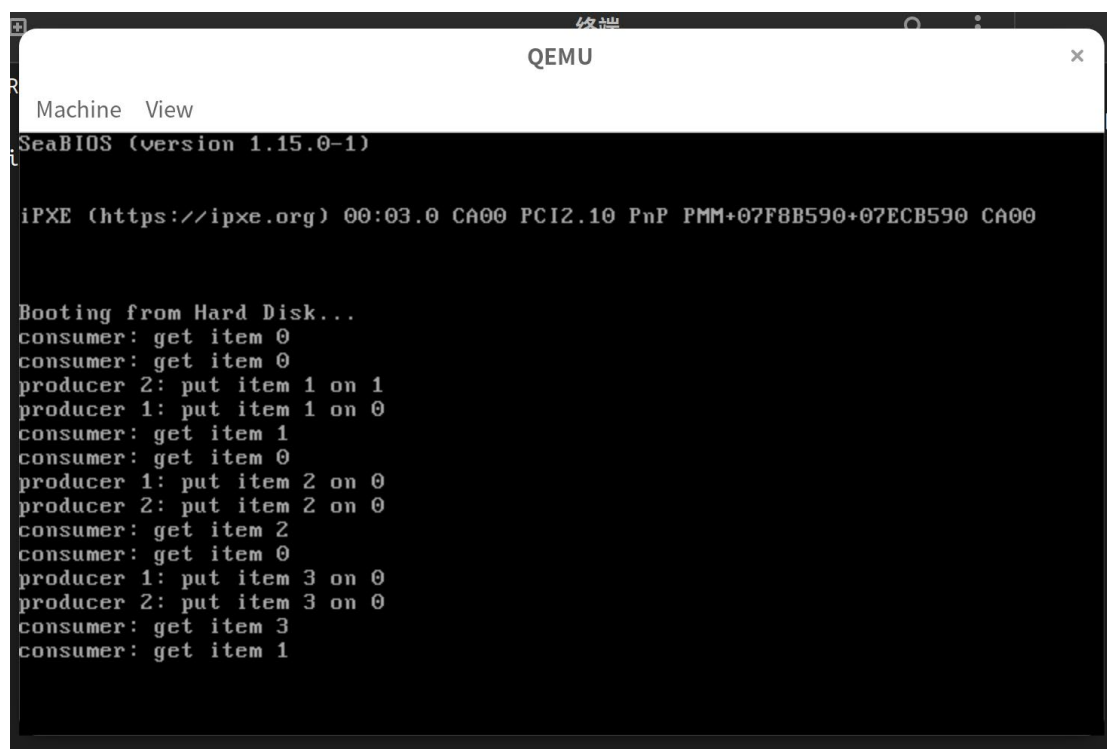
4. 实验结果

Assignment 1 代码复现题



Assignment 2 生产者-消费者问题

没有加入互斥锁和信号量时，多个生产者将货物放到相同的货柜、消费者在货物放到货柜上之前就请求：



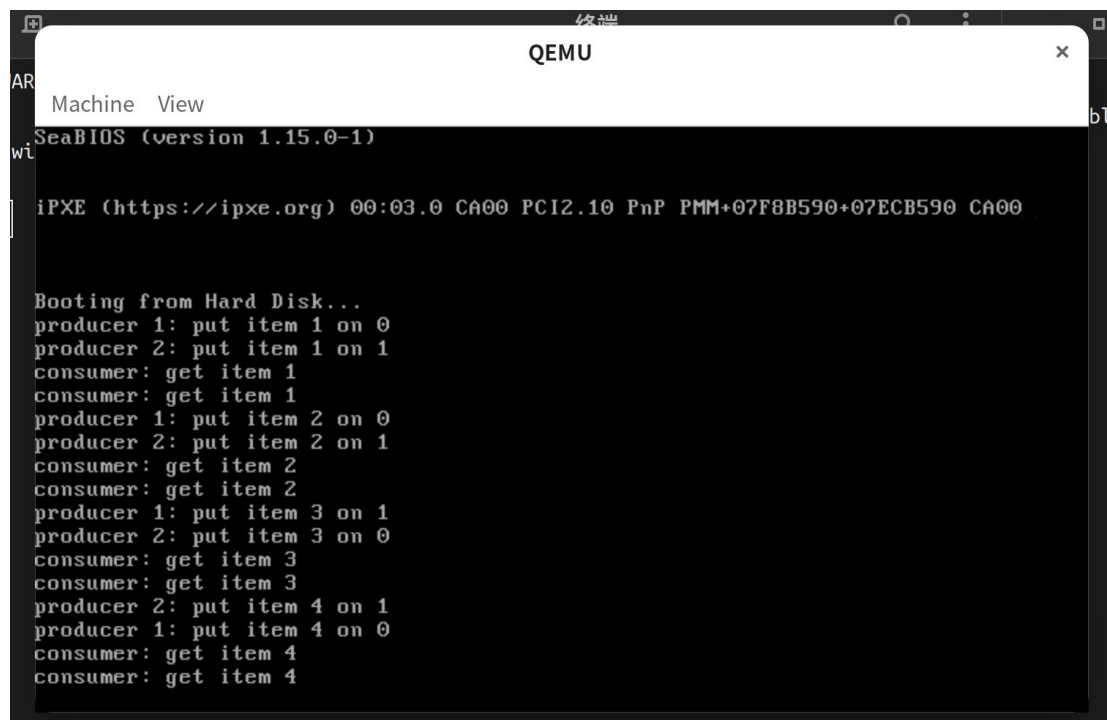
```
Machine View
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B590+07ECB590 CA00

Booting from Hard Disk...
consumer: get item 0
consumer: get item 0
producer 2: put item 1 on 1
producer 1: put item 1 on 0
consumer: get item 1
consumer: get item 0
producer 1: put item 2 on 0
producer 2: put item 2 on 0
consumer: get item 2
consumer: get item 0
producer 1: put item 3 on 0
producer 2: put item 3 on 0
consumer: get item 3
consumer: get item 1
```

发现消费者取到了非法的商品编号，两个生产者同时把商品放到了相同的货架上。

加入互斥锁和信号量，冲突消失，执行流程正常：



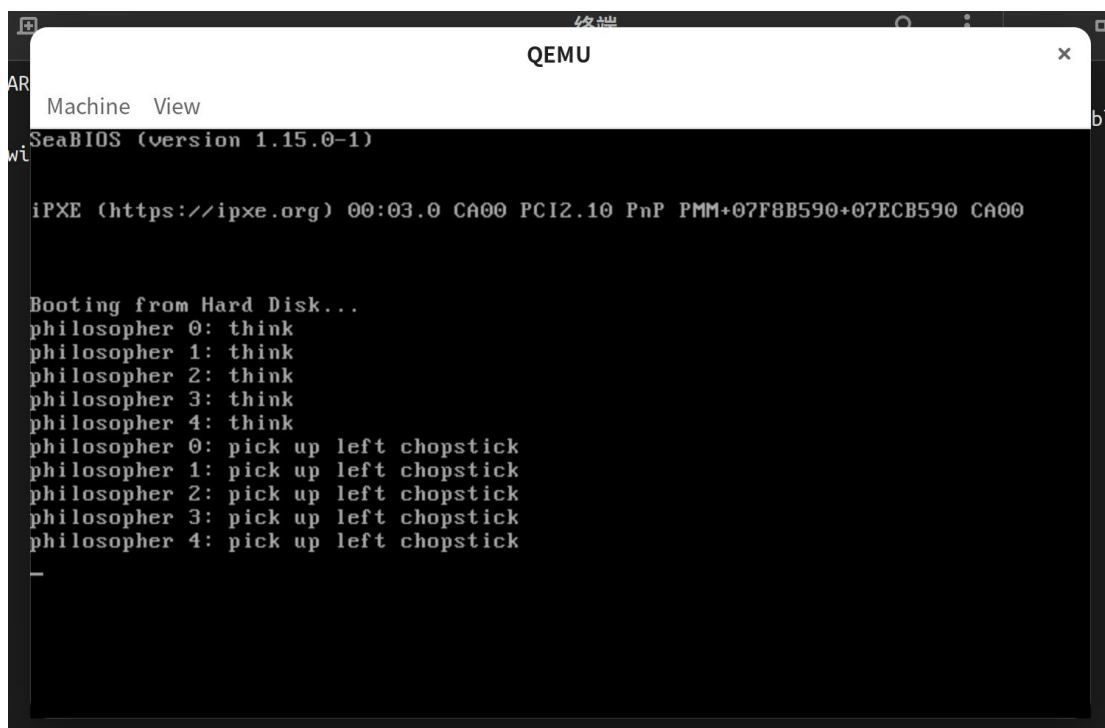
```
Machine View
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B590+07ECB590 CA00

Booting from Hard Disk...
producer 1: put item 1 on 0
producer 2: put item 1 on 1
consumer: get item 1
consumer: get item 1
producer 1: put item 2 on 0
producer 2: put item 2 on 1
consumer: get item 2
consumer: get item 2
producer 1: put item 3 on 1
producer 2: put item 3 on 0
consumer: get item 3
consumer: get item 3
producer 2: put item 4 on 1
producer 1: put item 4 on 0
consumer: get item 4
consumer: get item 4
```

Assignment 3 哲学家就餐问题

没有限制最大就餐人数，出现死锁：



```
QEMU
Machine View
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B590+07ECB590 CA00

Booting from Hard Disk...
philosopher 0: think
philosopher 1: think
philosopher 2: think
philosopher 3: think
philosopher 4: think
philosopher 0: pick up left chopstick
philosopher 1: pick up left chopstick
philosopher 2: pick up left chopstick
philosopher 3: pick up left chopstick
philosopher 4: pick up left chopstick
```

加入信号量限制最大同时就餐人数，死锁解除：



```
QEMU
Machine View
philosopher 2: eat
philosopher 4: get a seat
philosopher 4: pick up left chopstick
philosopher 2: put down chopsticks
philosopher 2: leave the table
philosopher 2: think
philosopher 3: get a seat
philosopher 3: pick up left chopstick
philosopher 1: pick up right chopstick
philosopher 1: eat
philosopher 1: put down chopsticks
philosopher 1: leave the table
philosopher 1: think
philosopher 0: pick up right chopstick
philosopher 0: eat
philosopher 2: get a seat
philosopher 2: pick up left chopstick
philosopher 0: put down chopsticks
philosopher 0: leave the table
philosopher 0: think
philosopher 4: pick up right chopstick
philosopher 4: eat
philosopher 1: get a seat
philosopher 1: pick up left chopstick
```

5. 总结

本次实验通过原子指令实现了自旋锁，并基于锁构建信号量，解决了消失的芝士汉堡、生产者-消费者及哲学家就餐问题。实验复现了不同锁机制（如 xchg 与 bts 指令），对比了无同步时的竞态条件与信号量

方案的稳定性。针对哲学家就餐问题，演示了死锁场景并找到了有效避免死锁的方法。实验深化了对同步原语的理解，并实践了多线程问题的分析与解决能力。