LAB6 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：自旋锁和信号量的实现

专业名称：计算机科学与技术

学生姓名：马福泉

学生学号：　23336179

实验地点：　实验中心 B202

实验时间：　2025.5.18

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

- 预备知识：汇编语言基础，计算机体系结构，操作系统基础，调试工具的使用（如 gdb）等。
- 实验环境：
  - 虚拟机版本/处理器型号：VMware-Ubuntu22.04.5/i386（32 位）
  - 代码编辑环境： 编辑器：VSCode
    插件：C/C++插件，汇编插件
  - 代码编译工具： 编译器：gcc
    工具链：binutils、make、qemu、nasm 等
    调试工具：gdb
  - 重要三方库信息：GNU binutils：工具集（如 as、ld）
    gdb：调试工具
    QEMU、Bochs：模拟器用于测试 等

Section 3 实验任务

实验任务 1：自旋锁与信号量的实现

- 子任务 1.1-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 1.2-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：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
问题描述：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。	**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 2.1-线程的竞争与冲突

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）

- 子任务 2.2-利用信号量解决问题

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。

3. 实验任务 3：哲学家就餐问题
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：
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
一个哲学家一次只能拿起一根筷子，在拿到两根筷子之前不会放下手里的筷子
如果筷子在其他哲学家手里，则需等待
当一个饥饿的哲学家同时拥有两根筷子时，他会开始吃面
吃完面后的哲学家会同时放下两根筷子，并开始思考

- 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：

1. 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。
2. 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。

- 子任务 2-死锁应对策略（选做）

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。

Section 4 实验步骤与实验结果

---------- 实验任务 1 ----------

任务要求；自旋锁与信号量的实现

- 子任务 1.1-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 1.2-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。

- 思路分析：

1.1 复制代码文件到本地，稍加修改加入个人身份复现即可

1.2 在原代码基础上

- 实验步骤：

1.1.1 复制代码到本地，自旋锁方法解决

1.1.2 复制代码到本地，信号量方法解决

```
Make clean
Make build
Make run
```

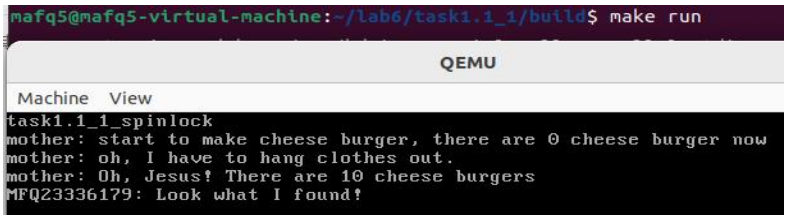
1.2 自实现锁机制

基于 lock bts 指令的原子操作，设置内存地址中特定位（第 0 位）并返回该位的原始值。与 xchg 指令相比，lock bts 只针对内存中的第 0 位进行操作，而 xchg 实现的是完整的 32 位原子交换。lock bts 通过 setc 指令从 CPU 寄存器 CF 中获取被操作比特位的原始值，而 xchg 通过 eax 寄存器传回内存地址中原始的 32 位值。bts 需要显式使用 lock 前缀保证原子性，而 xchg 指令具有隐式原子性。

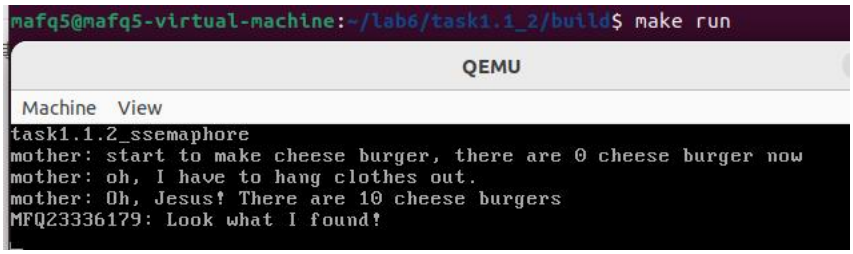
```
asm_atomic_exchange:            ; 定义一个名为 asm_atomic_exchange 的函数
    push ebp                    ; 保存基指针寄存器 ebp 的值到堆栈中
    mov ebp, esp                ; 将栈指针 esp 的值移动到基指针寄存器 ebp 中，建立新的栈帧
    pushad                      ; 将所有通用寄存器（eax, ebx, ecx, edx, esi, edi）的值压栈保存

    mov ebx, [ebp + 4 * 3]       ; 将函数的第三个参数（memory地址）加载到ebx寄存器中
    xor eax, eax                ; 将eax寄存器置0，准备进行位测试
    lock bts dword [ebx], eax   ; 原子操作：在ebx指向的内存地址上测试并设置第0位（使用lock前缀保证原子性）
    setc al                     ; 根据进位标志（CF）设置al寄存器的值（0或1）
    movzx eax, al                ; 将al寄存器的值零扩展到eax寄存器中（将8位值扩展到32位）
    mov ebx, [ebp + 4 * 2]       ; 将函数的第二个参数（register地址）加载到ebx寄存器中
    mov [ebx], eax              ; 将eax寄存器的值存储到ebx指向的内存地址中，即更新register变量的值

    popad                       ; 从堆栈中恢复所有通用寄存器的值
    pop ebp                     ; 从堆栈中恢复基指针寄存器 ebp 的值
    ret                         ; 返回调用者
```

- 实验结果展示：

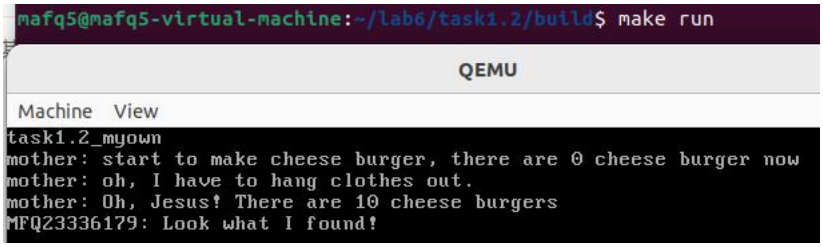
1.1.1 复现自旋锁方法解决



1.1.2 复现信号量方法解决



1.2 自实现锁机制



---------- 实验任务 2 ----------

- 任务要求：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
问题描述：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。	**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 2.1-线程的竞争与冲突

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）

子任务 2.2-利用信号量解决问题

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。

- 思路分析：篇幅原因，只展示伪代码如下

```
    1.全局变量和信号量定义
    蛋糕数量:matcha_cakes 和 mango_cakes 分别表示抹茶蛋糕和芒果蛋糕的数量。
    最大蛋糕数量:MAX_CAKES 定义了点心盘最多可以容纳的蛋糕数量。
    信号量:
    mutex:用于互斥访问共享资源(蛋糕数量)。
    matcha_request 和 mango_request:分别表示抹茶蛋糕和芒果蛋糕的请求信号。
    matcha_ready 和 mango_ready:分别表示抹茶蛋糕和芒果蛋糕准备好的信号。
    2. 函数定义
    delay_function:一个简单的延迟函数,用于模拟时间延迟。
    waiterA 和 waiterB:分别代表服务生 A 和 B 的线程函数,负责添加抹茶蛋糕和芒果蛋糕。
    male_guest 和 female_guest:分别代表男性和女性顾客的线程函数,负责消费抹茶蛋糕和芒果蛋糕。
    3. 线程函数逻辑
    服务生线程(waiterA 和 waiterB):
    等待蛋糕请求信号。
    获取互斥锁,检查盘子是否已满。
    如果未满,添加蛋糕并更新数量。
    释放互斥锁,发送蛋糕准备好的信号。
    模拟准备下一个蛋糕的时间。
    顾客线程(male_guest 和 female_guest):
    获取互斥锁,检查是否有对应口味的蛋糕。
    如果有,取走蛋糕并更新数量。
    如果没有,发送蛋糕请求信号,等待蛋糕准备好的信号。
    再次获取互斥锁,取走蛋糕。
    模拟食用时间。
    4. 主线程逻辑
    初始化信号量和蛋糕数量。
    创建服务生和顾客线程。
    模拟蛋糕服务过程:
    服务生添加蛋糕。
    顾客请求蛋糕。
    处理蛋糕请求和供应。
    监控和输出蛋糕服务状态:
    持续监控蛋糕数量和状态。
    输出当前蛋糕数量和状态。
```

- 实验步骤：

按思路编写代码, 篇幅原因，伪代码见上文，具体代码见附件

编写测试样例，为充分体现各个场景，编写样例如下：

创建两个服务生线程

服务员 A 添加一个抹茶蛋糕

服务员 B 添加一个芒果蛋糕

男人 0 请求（找到抹茶蛋糕并拿走）

男人 1 请求（找不到抹茶蛋糕，通知服务员 A）

服务员 A 添加一个抹茶蛋糕

女人 0 请求（找到芒果蛋糕并拿走）

女人 1 请求（找不到芒果蛋糕，通知服务员 B）

服务员 B 添加一个芒果蛋糕

男人 2 请求（应该能拿到服务员 A 新放的抹茶蛋糕）

女人 2 请求（应该能拿到服务员 B 新放的芒果蛋糕）

服务员 A 添加 1 个抹茶蛋糕，连续进行五次，

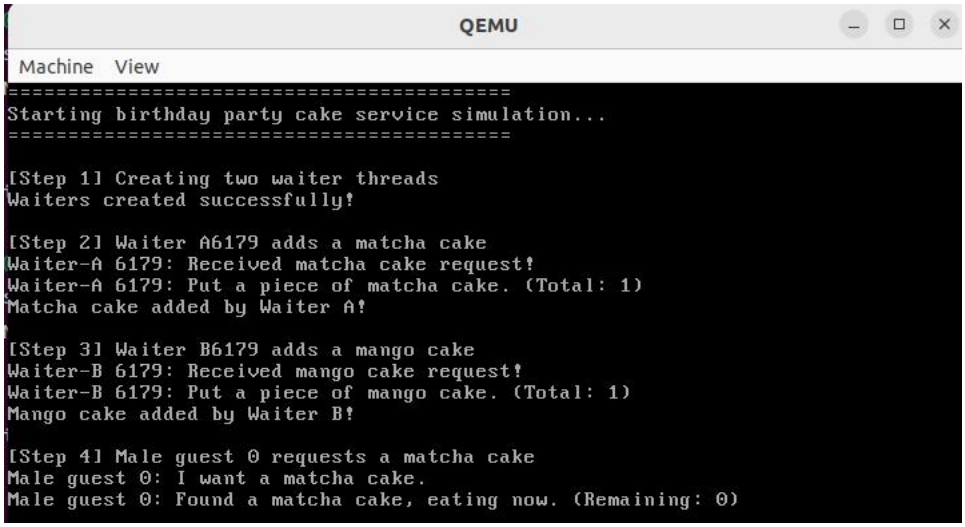
女人 3 请求（找不到芒果蛋糕，通知服务员 B）

服务员 B 尝试添加一个芒果蛋糕，但是盘子已满，失败。

结束

```
make clean
make build
make run
```

- 实验结果展示：



```
[Step 5] Male guest 1 requests (will notify Waiter A)
Male guest 1: I want a matcha cake.
Male guest 1: No matcha cake available, requesting one...
Waiter-A 6179: Received matcha cake request!
Waiter-A 6179: Put a piece of matcha cake. (Total: 1)
Male guest 1: Got my matcha cake, eating now. (Remaining: 0)
Male guest 1 has requested a cake from Waiter A!

[Step 6] Waiter A is adding a matcha cake
Waiter A has added a matcha cake!

[Step 7] Female guest 0 requests a mango cake
Female guest 0: I want a mango cake.
Female guest 0: Found a mango cake, eating now. (Remaining: 0)
Female guest 0 has taken her cake!

[Step 8] Female guest 1 requests (will notify Waiter B)
Female guest 1: I want a mango cake.
Female guest 1: No mango cake available, requesting one...
Waiter-B 6179: Received mango cake request!
Waiter-B 6179: Put a piece of mango cake. (Total: 1)
Female guest 1: Got my mango cake, eating now. (Remaining: 0)
```

```
[Step 8] Female guest 1 requests (will notify Waiter B)
Female guest 1: I want a mango cake.
Female guest 1: No mango cake available, requesting one...
Waiter-B 6179: Received mango cake request!
Waiter-B 6179: Put a piece of mango cake. (Total: 1)
Female guest 1: Got my mango cake, eating now. (Remaining: 0)
Female guest 1 has requested a cake from Waiter B!

[Step 9] Waiter B is adding a mango cake
Waiter B has added a mango cake!

[Step 10] Male guest 2 requests (should get the new matcha cake)
Male guest 2: I want a matcha cake.
Male guest 2: No matcha cake available, requesting one...
Waiter-A 6179: Received matcha cake request!
Waiter-A 6179: Put a piece of matcha cake. (Total: 1)
Male guest 2: Got my matcha cake, eating now. (Remaining: 0)
Male guest 2 has taken his cake!
```

```
[Step 11] Female guest 2 requests (should get the new mango cake)
Female guest 2: I want a mango cake.
Female guest 2: No mango cake available, requesting one...
Waiter-B 6179: Received mango cake request!
Waiter-B 6179: Put a piece of mango cake. (Total: 1)
Female guest 2: Got my mango cake, eating now. (Remaining: 0)
Female guest 2 has taken her cake!

[Step 12] Waiter A adds 5 matcha cakes consecutively
Waiter A is adding matcha cake #1
Waiter-A 6179: Received matcha cake request!
Waiter-A 6179: Put a piece of matcha cake. (Total: 1)
Waiter A is adding matcha cake #2
Waiter-A 6179: Received matcha cake request!
Waiter-A 6179: Put a piece of matcha cake. (Total: 2)
Waiter A is adding matcha cake #3
Waiter-A 6179: Received matcha cake request!
Waiter-A 6179: Put a piece of matcha cake. (Total: 3)
Waiter A is adding matcha cake #4
Waiter-A 6179: Received matcha cake request!
Waiter-A 6179: Put a piece of matcha cake. (Total: 4)
```

```
Waiter-A 6179: Put a piece of matcha cake. (Total: 3)
Waiter A is adding matcha cake #4
Waiter-A 6179: Received matcha cake request!
Waiter-A 6179: Put a piece of matcha cake. (Total: 4)
Waiter A is adding matcha cake #5
Waiter-A 6179: Received matcha cake request!
Waiter-A 6179: Put a piece of matcha cake. (Total: 5)
Waiter A has added 5 matcha cakes!

[Step 13] Female guest 3 requests (will notify Waiter B)
Female guest 3: I want a mango cake.
Female guest 3: No mango cake available, requesting one...
Waiter-B 6179: Received mango cake request!
Waiter-B 6179: Plate is full, cannot add more cakes.
Female guest 3: Strange, there is still no mango cake!
Female guest 3 has requested a cake from Waiter B!

[Step 14] Waiter B tries to add a mango cake but the plate is full
Waiter B failed to add a mango cake because the plate is full!

========================================
Cake service simulation completed!
========================================
Current status - Matcha: 5, Mango: 0, Total: 5/5
```

如上图所示：

服务员 A 添加一个抹茶蛋糕

服务员 B 添加一个芒果蛋糕

男人 0 请求（找到抹茶蛋糕并拿走）

男人 1 请求（找不到抹茶蛋糕，通知服务员 A）

服务员 A 添加一个抹茶蛋糕

女人 0 请求（找到芒果蛋糕并拿走）

女人 1 请求（找不到芒果蛋糕，通知服务员 B）

服务员 B 添加一个芒果蛋糕

男人 2 请求（应该能拿到服务员 A 新放的抹茶蛋糕）

女人 2 请求（应该能拿到服务员 B 新放的芒果蛋糕）

服务员 A 添加 1 个抹茶蛋糕，连续进行五次，

女人 3 请求（找不到芒果蛋糕，通知服务员 B）

服务员 B 尝试添加一个芒果蛋糕，但是盘子已满，失败。

---------- 实验任务 3 ----------

- 任务要求：哲学家就餐问题

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：
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
一个哲学家一次只能拿起一根筷子，在拿到两根筷子之前不会放下手里的筷子
如果筷子在其他哲学家手里，则需等待当一个饥饿的哲学家同时拥有两根筷子时，他会开始吃面吃完面后的哲学家会同时放下两根筷子，并开始思考
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：
1. 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。
2. 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。

- 思路分析：查阅资料，可以的解决方法有：

(1) 限制同时拿筷子的哲学家数量：这种方法相对简单且直接，通过设置一个信号量来限制并发访问筷子的哲学家数量。它不需要复杂的逻辑或额外的同步机制。

(2) 奇偶编号规则：这种方法也比较简单，只需要规定哲学家拿筷子的顺序即可。然而，它要求哲学家必须按照固定的顺序拿筷子，这可能会增加一些复杂性。

(3) 双筷子同步策略：这种方法相对复杂，因为它要求哲学家在拿起筷子前必须同时检查两侧的筷子是否都可用。这可能需要额外的同步机制或逻辑来确保哲学家在尝试获取筷子时不会造成死锁。

- 实验步骤：

  1.编写一线程，执行观察到确实会发生死锁。

```
void philosopher(void *arg) {
    int id = *(int *)arg;
    int left_chop = id;
    int right_chop = (id + 1) % PHILOSOPHER_NUM;

    while (true) {
        // 思考
        printf("Philosopher %d is thinking...\n", id);
        delay(0x1ffffff);  // 思考时间

        // 尝试就餐
        printf("Philosopher %d is hungry, trying to get chops...\n", id);

        chops[left_chop].P();  // 拿左叉子
        printf("Philosopher %d picked up left chop %d\n", id, left_chop);
        delay(0xffffff);  // 延迟一段时间，增加死锁概率

        programManager.schedule();

        chops[right_chop].P();  // 拿右叉子
        printf("Philosopher %d picked up right chop %d\n", id, right_chop);

        // 就餐
        printf("Philosopher %d is eating...\n", id);
        delay(0x3ffffff);  // 就餐时间

        // 放下叉子
        chops[right_chop].V();
        printf("Philosopher %d put down right chop %d\n", id, right_chop);

        chops[left_chop].V();
        printf("Philosopher %d put down left chop %d\n", id, left_chop);

        printf("Philosopher %d finished eating and starts thinking again\n", id);
    }
}
```

```
void first_thread(void *arg) {
    // 清屏
    stdio.moveCursor(0);
    for (int i = 0; i < 25 * 80; ++i) {
        stdio.print(' ');
    }
    stdio.moveCursor(0);

    // 初始化信号量
    for (int i = 0; i < PHILOSOPHER_NUM; ++i) {
        chops[i].initialize(1);  // 每个叉子初始可用
    }

    // 创建哲学家线程
    int ids[PHILOSOPHER_NUM];
    for (int i = 0; i < PHILOSOPHER_NUM; ++i) {
        ids[i] = i;
        programManager.executeThread(philosopher, &ids[i], "philosopher", 1);

    }


    asm_halt();
}
```

```
mafq5@mafq5-virtual-machine:~/lab6/task3_1/build$ make run
                                QEMU
Machine  View
Philosopher 0 is thinking...
Philosopher 0 is hungry, trying to get chops...
Philosopher 0 picked up left chop 0
Philosopher 1 is thinking...
Philosopher 1 is hungry, trying to get chops...
Philosopher 1 picked up left chop 1
Philosopher 2 is thinking...
Philosopher 2 is hungry, trying to get chops...
Philosopher 2 picked up left chop 2
Philosopher 3 is thinking...
Philosopher 3 is hungry, trying to get chops...
Philosopher 3 picked up left chop 3
Philosopher 4 is thinking...
Philosopher 4 is hungry, trying to get chops...
Philosopher 4 picked up left chop 4
```

2. 按以下思路编写代码，采用**奇偶编号规则**解决死锁，篇幅原因详细代码见附件。

(1) 初始化和设置

全局变量：定义了哲学家的数量（PHILOSOPHER_NUM）和用于控制访问筷子的信号量数组（chops）。

中断管理器：初始化中断管理器，启用时间中断，为模拟时间流逝和线程调度提供基础。

输出管理器：初始化屏幕 IO 处理器，用于在屏幕上显示哲学家的状态。

程序/线程管理器：初始化程序管理器，用于创建和管理哲学家线程。

(2) 哲学家线程的创建和执行

清屏：在屏幕上清除所有内容，准备显示哲学家的状态。

初始化信号量：对每支筷子对应的信号量进行初始化，初始值设为 1，表示筷子是可用的。

创建哲学家线程：为每位哲学家创建一个线程，每个线程都调用 philosopher 函数，并将哲学家的 ID 作为参数传递。

(3) 哲学家线程的执行流程

思考：哲学家开始思考，并打印思考的状态信息，然后等待一段时间（模拟思考时间）。

尝试就餐：哲学家感到饥饿，尝试获取筷子。首先打印尝试获取筷子的状态信息。

获取筷子：根据哲学家的编号（奇数或偶数），决定获取筷子的顺序：

偶数编号的哲学家先尝试拿起右边的筷子，再拿起左边的筷子。

奇数编号的哲学家先尝试拿起左边的筷子，再拿起右边的筷子。

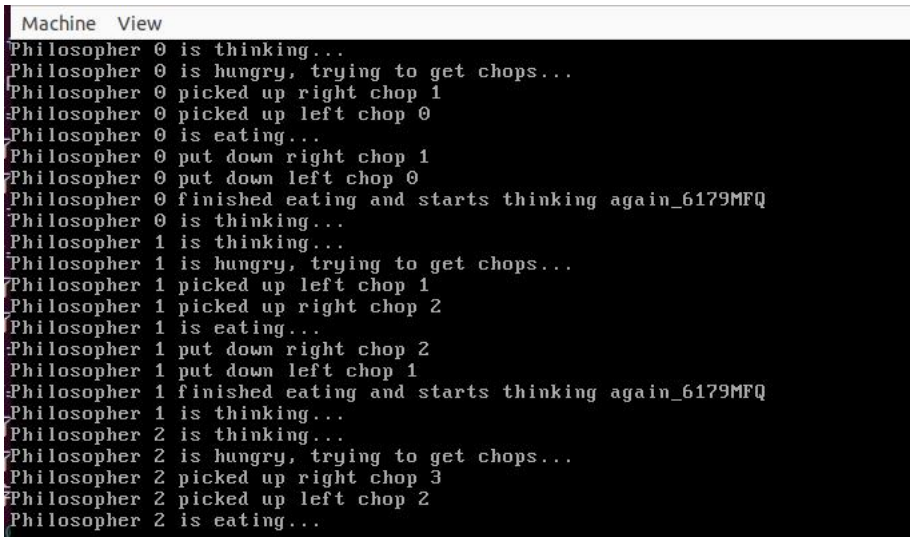
使用信号量的 P() 操作尝试获取筷子。如果筷子可用（信号量值大于 0），则哲学家获取筷子，信号量值减 1；如果筷子不可用，则哲学家等待。

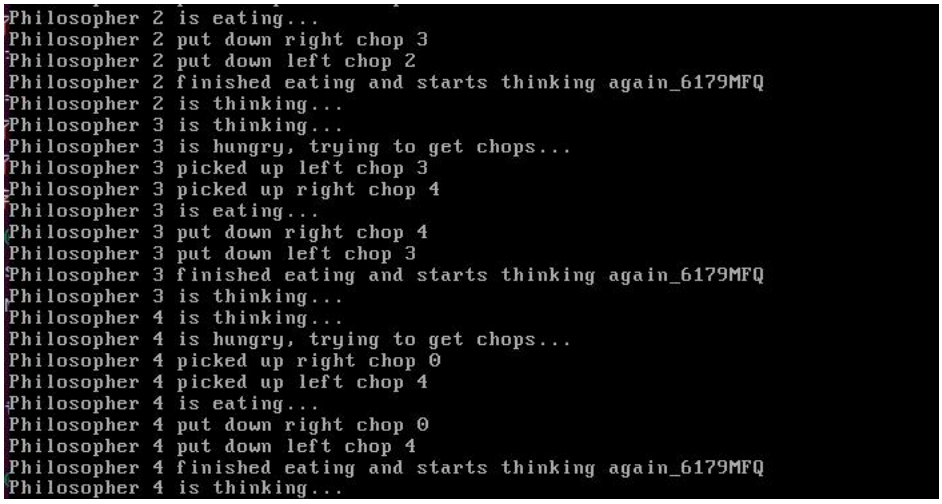
就餐：成功获取两支筷子后，哲学家开始就餐，并打印就餐的状态信息，然后等待一段时间（模拟就餐时间）。

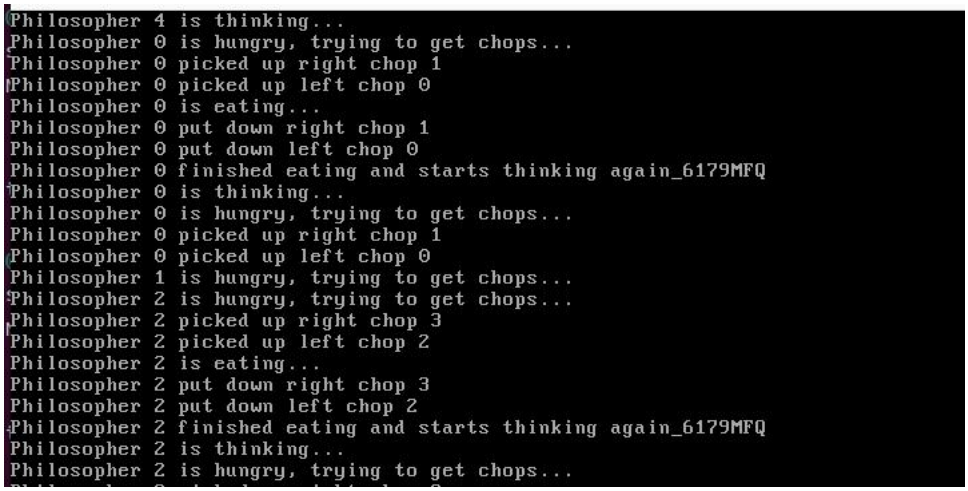
放下筷子：就餐结束后，哲学家放下筷子，使用信号量的 V() 操作释放筷子，信号量值加 1，表示筷子再次可用。

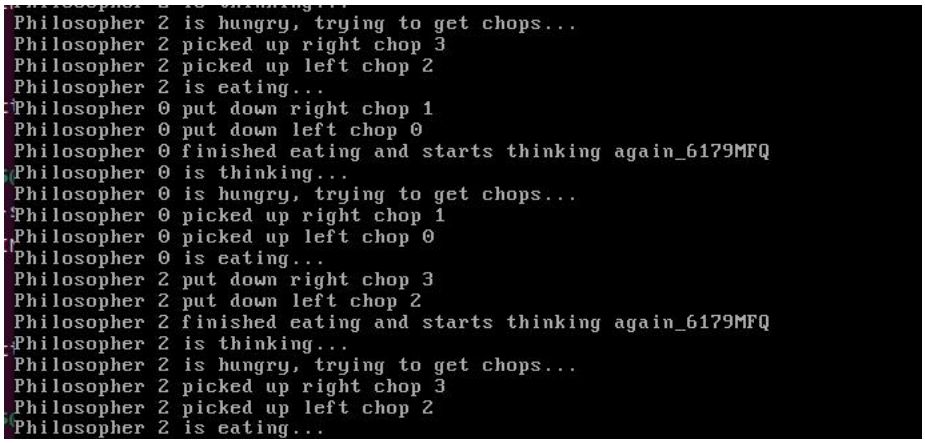
循环：哲学家线程无限循环，交替进行思考和就餐。

- 实验结果展示：









以上是已经经过一轮用餐循环连续截图，死锁问题解决，输出 6179MFQ 是本人个人信息。

Section 5 实验总结与心得体会

- 同步机制的重要性：通过实验，我深刻体会到了同步机制对于保证程序正确性和性能的重要性。不同的同步工具有其特定的用途和最佳实践场景，我学会了如何根据具体问题选择合适的同步工具。

- 死锁的避免与处理：实验让我体会到了死锁对系统性能的严重影响。我学会了一些避免死锁的策略，这些策略对于设计可靠的并发程序至关重要。
- 资源管理的复杂性：在处理共享资源时，我遇到了资源分配和回收的复杂性。我发现并发程序的调试比单线程程序更具挑战性。问题可能只在特定的线程执行顺序下出现，暂时没有探究清楚什么原因。

Section 6 附录：参考资料清单

1. 部分代码 debug 与指令除了参考实验指导书，还参考了大语言模型
2. 参考文章，博客部分如下：

[https://blog.csdn.net/brainkick/article/details/7583727]
哲学家进餐问题--C 语言 PV 操作_哲学家就餐问题 pv 操作-CSDN 博客