Assignment 5 - Semaphore

2025 年 5 月 18 日

1 Background

在本次作业中，你需要在 xv6 的内核态中实现 Semaphore。Semaphore 是一种 **带计数器的互斥锁**。

2 Specifictions

2.1 Semaphore API

`sem_init()` initializes the semaphore at the address pointed to by sem. The value argument specifies the initial value for the semaphore.

`sem_wait()` decrements (locks) the semaphore pointed to by sem. If the semaphore's value is greater than zero, then the decrement proceeds, and the function returns, immediately. If the semaphore currently has the value zero, then the call blocks until it becomes possible to perform the decrement (i.e., the semaphore value rises above zero).

`sem_post()` increments (unlocks) the semaphore pointed to by sem. If the semaphore's value consequently becomes greater than zero, then another process or thread blocked in a `sem_wait()` call will be woken up and proceed to lock the semaphore.

2.2 Dining Philosophers Problem

Five philosophers dine together at the same table. Each philosopher has their own plate at the table. There is a fork between each plate. The dish served is a kind of spaghetti which has to be eaten with two forks. Each philosopher can only alternately think and eat. Moreover, a philosopher can only eat their spaghetti when they have both a left and right fork. Thus two forks will only be available when their two nearest neighbors are thinking, not eating. After an individual philosopher finishes eating, they will put down both forks. The

Build commit hash: 9f104d5

problem is how to design a concurrent algorithm such that any philosopher will not starve; i.e., each can forever continue to alternate between eating and thinking, assuming that **no philosopher can know when others may want to eat or think.**

The problem was designed to illustrate the challenges of avoiding deadlock, a system state in which no progress is possible. To see that a proper solution to this problem is not obvious, consider a proposal in which each philosopher is instructed to behave as follows:

- think unless the left fork is available; when it is, pick it up;
- think unless the right fork is available; when it is, pick it up;
- when both forks are held, eat for a fixed amount of time;
- put the left fork down;
- put the right fork down;
- repeat from the beginning.

With these instructions, the situation may arise where each philosopher holds the fork to their left; in that situation, they will all be stuck forever, waiting for the other fork to be available: it is a deadlock.

哲学家就餐问题可以这样表述，假设有五位哲学家围坐在一张圆形餐桌旁，做以下两件事情之一：吃饭，或者思考。吃东西的时候，他们就停止思考，思考的时候也停止吃东西。餐桌上有五碗意大利面，每位哲学家之间各有一只餐叉。因为用一只餐叉很难吃到意大利面，所以假设哲学家必须用两只餐叉吃东西。他们只能使用自己左右手边的那两只餐叉。哲学家就餐问题有时也用米饭和五根筷子而不是意大利面和餐叉来描述，因为吃米饭必须用两根筷子。

问题在于如何设计一套规则，使得在哲学家们在完全不交谈，也就是无法知道其他人可能在什么时候要吃饭或者思考的情况下，可以在这两种状态下永远交替下去。

这个问题旨在说明避免死锁的挑战，死锁是一种程序无法继续运行的状态。要更好理解这个问题，假设我们要求哲学家遵守以下规则：

- 哲学家在左边的叉子可用（没有其他人拿起）之前处于思考状态。如果左边的叉子可用，就拿起来。
- 哲学家等待右边的叉子可用。如果右边的叉子可用，就拿起来。
- 如果两个叉子都已经拿起来，开始吃意大利面，每次吃面都花费同样的时间。
- 吃完后先放下左边的叉子。
- 然后放下右边的叉子。
- 开始思考（进入一个循环）。

这个解法是失败的，当每个哲学家都拿起左侧的叉子，等待右侧的叉子可用时，就会进入死锁状态，每个哲学家将永远都在等待（右边的）另一个哲学家放下叉子。

3 Hints

请先全局搜索关键词 `TODO` 和 `Assignment`。

关于 xv6 内核中的同步机制，你可以参考 Lab 课件 *互斥与同步 Mutual Exclusion & Synchronization* 和 *同步 2 Synchronization 2*。

你应该先在 `lock.h` 和 `lock.c` 中实现 Semaphore API。前者包含 `struct semaphore` 结构体的定义，后者包含 `sem_init()`、`sem_wait()` 和 `sem_post()` 的实现。然后，你应该在 `a5/checkpoint3.c` 中实现哲学家吃饭问题。你可以在该文档中找到哲学家吃饭问题的描述。

4 Checkpoints

本次作业一共 4 分，有 3 个 Checkpoint 和一份报告。3 个 Checkpoint 各 1 分、2 分、1 分。报告不占分，但是强制要求写。你的报告应该符合以下要求：

- PDF 文件格式的报告。
- 每个 checkpoint 运行成功的截图。
- 记录完成这个作业一共花了多少时间。
- （可选）描述你在完成这个作业时遇到的困难，或者描述你认为本次作业还需要哪些指引。
- （可选）你可以在报告中分享你完成这个作业的思路。
- 越简洁越好。

要求 你应该使用 `make runsmp` 启动**多核心**的 xv6。

本次作业没有用户模式代码,在内核启动后它会直接开始运行位于 *os/a5/sync_main.c* 中的 `synclab_main` 函数，它会分别调用 3 个 checkpoint 的测试函数。如果你想测试单个 checkpoint，可以在该函数中注释掉其他的函数调用。

Checkpoint 说明 第一个 Checkpoint 是初始化一个值为 1 的 Semaphore，它的语义和单一的互斥锁是一致的。

第二个 Checkpoint 是测试一个值不为 1 的 Semaphore。第三个 Checkpoint 要求你实现哲学家吃饭问题并正确避免死锁，你只能使用 Semaphore 来解决它。

全部通过提示 如果通过了所有测试点，你应该能看到类似下面的输出：

```
[INFO  0,-1] synclab_init: sync lab init
[INFO  0,1] checkpoint1: -> checkpoint 1 starts
[INFO  3,1] checkpoint1: -> checkpoint 1 passed
[INFO  3,1] checkpoint2: -> checkpoint 2 starts
[INFO  1,1] checkpoint2: -> checkpoint 2 passed
[INFO  1,1] checkpoint3: -> checkpoint 3 starts
```

```
[INFO  0,1] checkpoint3: -> checkpoint 3 passed
[INFO  0,1] synclab_main: all checks passed
```

5 提交

将你的报告重命名为 report.pdf，放到与该 PDF 文件的同目录下，运行 `make handin`，这会生成一个 handin.zip 压缩包。上传该文件到 Blackboard 即可.