Computer System Engineering Lab2

18302010018 俞哲轩

Exercise 1

semaphore的用途是**调度线程**。一些线程生产、另一些线程消费，semaphore让生产和消费保持合乎逻辑的执行顺序，保证线程执行的**有序**性。

mutex的用途是**保护共享资源**。Mutex保证使用资源线程的唯一性和排他性，但无法限制资源释放后其他线程的申请顺序，是**无序**的。

当semaphore的值初始化为1时，它被称为**二元信号量**（*binary semaphore*），以**提供互斥**为目的的二元信号量也可以被成为**互斥锁**，即mutex，此时semaphore和mutex的**功能相同**；但semaphore还可以**调度对共享资源的访问**，即**调度线程**，一个被用作一组可用资源的计数器的信号量被称为**计数信号量**（*counting semaphore*）。

以**生产者-消费者问题**举例：生产者和消费者线程共享一个有n个槽的有限缓冲区，生产者线程不断生成新的项目加入缓冲区，消费者线程不断从缓冲区中取出项目消费。①因为加入和取出项目都涉及更新共享资源，所以必须保证对缓冲区的访问是**互斥**的，此时需要使用**mutex**来**保护共享资源**。②我们还需**调度对缓冲区的访问**：如果缓冲区满了，生产者进程就必须等待，直至有槽可以使用；反之，如果缓冲区空了，消费者进程就必须等待，直至有项目可以消费，此时需要**semaphore**来**调度线程**。

在**读者-写者问题（读者优先）**中同理，mutex用于**保护共享资源**（记录读者数量的变量）的更新，semaphore用于**调度线程**（只有没有读者时才能执行写者），不过此时的semaphore为1。

Exercise 2

1. 在何种情况下程序会产生死锁?

foo进程执行到x = x + 2或者P(lock2)且**bar进程**执行到y = y + 1或者P(lock1)的时候，会产生死锁。

2. 在死锁状态下，x，y，z的最终值可能是多少?

x，y，z的最终值可能是：

x	y	z
2	1	2

3. 如果程序正常终止，x，y，z的最终值可能是多少？（提示：对z的赋值操作不是原子性的）

x，y，z的最终值可能是：

x	y	z
3	3	3
3	3	2
3	3	1

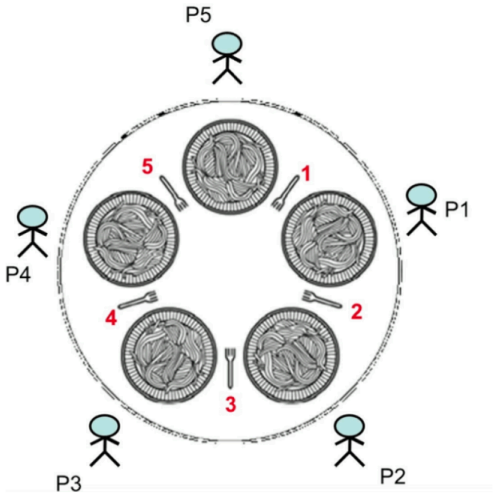
Exercise 3

```
int free = U;
sem lock_1 = 1, lock_2 = 1;

# <await (free > 0) free = free - 1;>
request() {
  P(lock_1);
  while(free <= 0) {
    /* wait until free > 0 */
  }
  P(lock_2);
  /* code: get one resource */
  free = free - 1;
  V(lock_2);
  P(lock_1);
}

# <free = free + number;>
release(int number) {
  P(lock_2);
  /* code: return several resources */
  free = free + number;
  P(lock_2);
}
```

Exercise 4



防止死锁的办法：每位哲学家都**先拿偶数号叉子**，即**偶数号**哲学家**先拿右手边的叉子**，**奇数号**哲学家**先拿左手边的叉子**。

`Class Philosopher`

```
// pseudo code
public class Philosopher implements Runnable {
  private final Object leftFork;
  private final Object rightFork;

  @Override
  public void run() {
    while (true) {
      think();
      // all the philosophers pick up the even number fork first
      if (id % 2 == 0) {
        rightFork.pickUpFork();
        leftFork.pickUpFork();
      } else {
        leftFork.pickUpFork(id, "left");
        rightFork.pickUpFork(id, "right");
      }
      eat();
      // all the philosophers put down the even number fork first
      if (id % 2 == 0) {
        rightFork.putDownFork();
        leftFork.putDownFork();
      } else {
        leftFork.putDownFork();
        rightFork.putDownFork();
      }
    }
  }
}
```

`Class Fork`

```
// pseudo code
public class Fork {
  private boolean isUsing = false;

  synchronized void pickUpFork() {
    while (isUsing) {
        wait();
    }
    isUsing = true;
  }

  synchronized void putDownFork() {
    isUsing = false;
    notifyAll();
  }
}
```

运行结果：