CSELAB2

by 黄子豪 18302010034

Exercise1

Mutex考虑的是限制资源，通常服务于共享资源；

Semaphore考虑的是调度线程，服务于多个线程间的执行的逻辑顺序。

以老师上课讲过的经典读者和写者问题为例：

reader和writer是不同的进程，为了避免读写的不一致，当访问同一个共享文件（或者buffer块）时，我们会需要给这个文件上锁：设置一个mutex，当有reader读的时候锁上它（mutex设为false），当结束读的时候释放它，当有writer试图写的时候同理。这样通过一个mutex单纯二元值，实现了对共享资源的限制。

现在我们假设程序的目的是writer每次向一个buffer区写入一个字节，然后reader可以读走这个字节，存在多个writer和reader，这样就有线程之间的逻辑限制了：在第N个reader读之前，必须至少有N个writer写了。这是我们设置一个信号量Semaphore，每次writer写了就执行原子性的函数semaphore_increase()，每次reader读了就执行原子性的函数semaphore_decrease()，通过信号量我们调度了线程之间的执行关系。

以上例子表明了Semaphore和Mutex的不同，然而假设在第二个例子中我们把semaphore的最大值和初始值设为1，就会发现这完全对应于只允许同时只有一个reader或者writer在工作，即实现了对这个buffer区的锁。所以在限制资源时Semaphore和Mutex其实可以达到相同的效果，Mutex此时相当于Semaphore值为1，但是这并不是一个好习惯，因为它把资源和进程混为一谈，二者应该是完全不同的：需要限制资源使用mutex，需要调度线程时使用semaphore。

Exercise2

1、foo先申请lock1，再申请lock2，再释放lock1，再释放lock2；而bar先申请lock2，再申请lock1，再释放lock1，再释放lock2。因此当foo申请了lock1正要申请lock2，bar申请了lock2正要申请lock1时会产生死锁。

2、只可能是x=2,y=1,z=2。

3、可能是x=3,y=3,z=1；x=3,y=3,z=2；x=3,y=3,z=3。

Exercise3

```
int free = U;
# your variables
semaphore mutex=1;
# <await (free > 0) free = free - 1;>
request(){
 # your code
    P(mutex);
    P(free);
    V(mutex);
}
# <free = free + number;>
release(int number){
```

```
 # your code
    P(mutex);
    free = free + number;
    V(mutex);
}
```

Exercise4

为了防止每只叉子同时被多个人拿到，给每只叉子设置了锁*available*，每个fork对象有自己的pickup()和putdown()方法，并且pickup()使用关键字*synchronized*限制，确保同时只能有一个进程尝试拿起这个叉子。

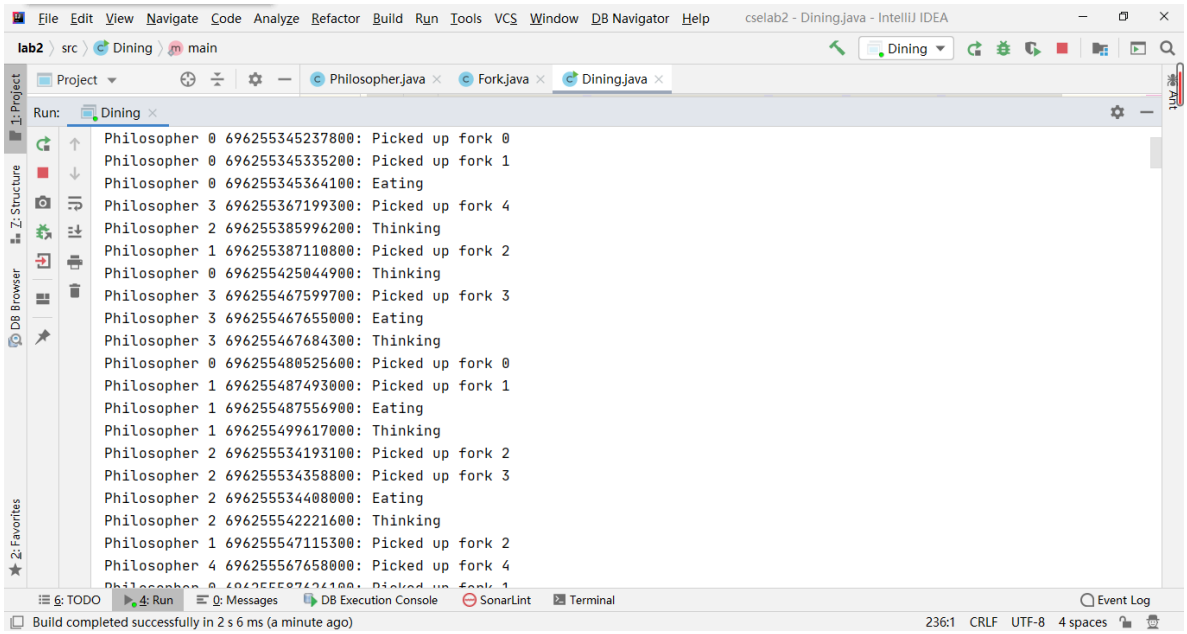
```
public class Fork {
    private final int index;
    private boolean available;

    public Fork(int index) {
        this.index = index;
        this.available = true;
    }

    public int getIndex() {
        return index;
    }

    public synchronized void pickUp(int name) throws InterruptedException {
        while (!available) {
            Thread.sleep(100);
        }
        System.out.println("Philosopher " + name + " " + System.nanoTime() + ":
Picked up fork " + index);
        available = false;
    }

    public void putDown() {
        available = true;
    }
}
```

而哲学家类拿叉子的顺序必须注意，否则可能会产生死锁，因此我采用了这样的策略：**每个哲学家都优先拿偶数号的叉子，这会使得偶数哲学家拿左边的叉子，奇数哲学家拿右边的叉子。**由此就避免了例如所有人都拿了左手的叉子带来的死锁。

```
public class Philosopher implements Runnable {
    private final Fork evenFork;
    private final Fork oddFork;
    private final int name;

    Philosopher(int name, Fork left, Fork right) {
        this.name = name;
        if (left.getIndex() % 2 == 0) {
            evenFork = left;
            oddFork = right;
        } else {
            evenFork = right;
```

```
            oddFork = left;
        }
    }

    private void doAction(String action) throws InterruptedException {
//        System.out.println(Thread.currentThread().getName() + " " +
//                action);
        System.out.println("Philosopher " + this.name + " " + action);
        Thread.sleep(((int) (Math.random() * 100)));
    }

    @Override
    public void run() {
        try {
            while (true) {
                doAction(System.nanoTime() + ": Thinking"); // thinking
                // your code
                pickupevenFork();
                pickupoddFork();
                doAction(System.nanoTime() + ": Eating");
                putdownevenfork();//保持拿叉子时一样的顺序
                putdownoddfork();
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    private void pickupevenFork() throws InterruptedException {
        evenFork.pickUp(name);
    }

    private void pickupoddFork() throws InterruptedException {
        oddFork.pickUp(name);
    }

    private void putdownevenfork() {
        evenFork.putDown();
    }

    private void putdownoddfork() {
        oddFork.putDown();
    }
}
```

最终程序运行正常，能够一直吃饭、思考，并不会因为死锁而停止：


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window DB Navigator Help
cselab2 - Dining.java - IntelliJ IDEA
lab2 src Dining main
Dining
Project
Philosopher.java
Fork.java
Dining.java
Run: Dining
Philosopher 0 696255345237800: Picked up fork 0
Philosopher 0 696255345335200: Picked up fork 1
Philosopher 0 696255345364100: Eating
Philosopher 3 696255367199300: Picked up fork 4
Philosopher 2 696255385996200: Thinking
Philosopher 1 696255387110800: Picked up fork 2
Philosopher 0 696255425044900: Thinking
Philosopher 3 696255467599700: Picked up fork 3
Philosopher 3 696255467655000: Eating
Philosopher 3 696255467684300: Thinking
Philosopher 0 696255480525600: Picked up fork 0
Philosopher 1 696255487493000: Picked up fork 1
Philosopher 1 696255487556900: Eating
Philosopher 1 696255499617000: Thinking
Philosopher 2 696255534193100: Picked up fork 2
Philosopher 2 696255534358800: Picked up fork 3
Philosopher 2 696255534408000: Eating
Philosopher 2 696255542221600: Thinking
Philosopher 1 696255547115300: Picked up fork 2
Philosopher 4 696255567658000: Picked up fork 4
1: Project
Z: Structure
DB Browser
2: Favorites
Ant
6: TODO
4: Run
0: Messages
DB Execution Console
SonarLint
Terminal
Event Log
Build completed successfully in 2 s 6 ms (a minute ago)
236:1 CRLF UTF-8 4 spaces