现代操作系统实验报告

——生产者消费者问题

实验编号 二
实验名称 生产者消费者问题
实验室名称 软件学院实验室
班级 软件学院 14 级二班
学号-姓名 刘嘉洋-1412620
实验日期 2016 年 10 月 27 日
至 2016 年 11 月 8 日

评分教师_____ 实验报告成绩_____
评分日期_____年_____月_____日

一、实验目的：

- 掌握信号量及相关知识
- 理解临界区及保护操作原理

二、实验环境

Java
Eclipse Mars Release (4.5.0)

三、实验报告

1. 运行结果：

```
-  Stock is 1
Producer 2
-  Stock is 2
Producer 1
-  Stock is 3
Producer 3
-  Stock is 4
Producer 4
+  Stock is 3
+  Stock is 2
Consumer 2
+  Stock is 1
Consumer 1
+  Stock is 0
Consumer 4
Consumer 3
-  Stock is 1
Producer 2
-  Stock is 2
Producer 3
-  Stock is 3
Producer 1
-  Stock is 4
Producer 4
-  Stock is 5
Producer 1
-  Stock is 6
Producer 4
-  Stock is 7
Producer 3
-  Stock is 8
Producer 2
-  Stock is 9
Producer 3
-  Stock is 10
Producer 4
+  Stock is 9
Consumer 2
+  Stock is 8
Consumer 3
```

说明：程序停止的设定为生产及操作共进行 20 次。

2. 关键代码说明：

a. 创建线程及 buffer：

```
        P1 = new Thread(new Producer());
        P2 = new Thread(new Producer());
        P3 = new Thread(new Producer());
        P4 = new Thread(new Producer());
        P1.start();
        P2.start();
        P3.start();
        P4.start();
        C1 = new Thread(new Consumer());
        C2 = new Thread(new Consumer());
        C3 = new Thread(new Consumer());
        C4 = new Thread(new Consumer());
        C1.start();
        C2.start();
        C3.start();
        C4.start();
    }

    static int time = 0;

    static Thread P1;
    static Thread P2;
    static Thread P3;
    static Thread P4;
    static Thread C1;
    static Thread C2;
    static Thread C3;
    static Thread C4;

    static Warehouse buffer = new Warehouse();
```

分别创建生产者与消费者各 4 个，并创建 buffer 以用于产品数量的存储与处理。

b. 生产者类:

```
// producer
static class Producer implements Runnable {
    static int num = 1;
    @Override
    public void run() {
        int n = num++; // every time a producer run, stock + 1
        while (true) {
            try {
                buffer.put(n);
                System.out.println("Producer " + n);
                // fast, sleep for 10 ms
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

类定义中包含了每次 run 中进行的产品数量的增一操作。

c. 消费者类:

```
// consumer
static class Consumer implements Runnable {
    @Override
    public void run() {
        while (true) {
            try {
                System.out.println("Consumer " + buffer.take());
                // slow, sleep for 1000 ms
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

类定义中包含了每次 run 中进行的产品数量的减一操作。

d. 仓库类（核心）：

首先实现三种信号量：

```
// Lock_NotFull
final Semaphore notFull = new Semaphore(10);
// Lock_NotEmpty
final Semaphore notEmpty = new Semaphore(0);
// Lock_Mutex(core)
final Semaphore mutex = new Semaphore(1);
```

每次生产操作的实现：

```
/**
 * put products into warehouse
 *
 * @param x
 * @throws InterruptedException
 */
public void put(Object x) throws InterruptedException {
    // guarantee NotFull
    notFull.acquire();  // Semaphore.acquire() -- get a permission
    // guarantee NotConflict
    mutex.acquire();
    try {
        // add stock
        items[putptr] = x;
        if (++putptr == items.length)
            putptr = 0;
        ++count;
        time++;
        if (time > 20) {
            P1.stop();
            P2.stop();
            P3.stop();
            P4.stop();
        }

        System.out.println("-  Stock is " + count);
    } finally {
        // quit from core
        mutex.release();    // Semaphore.release() -- liberate a premission
        // add NotEmpty Semaphore, there for allowing to take Item out
        notEmpty.release();
    }
}
```

每次消费操作的实现：

```java
/**
 * get products from warehouse
 *
 * @return
 * @throws InterruptedException
 */
public Object take() throws InterruptedException {
    // guarantee NotFull
    notEmpty.acquire();
    // guarantee NotConflict
    mutex.acquire();
    try {
        // reduce stock
        Object x = items[takeptr];
        if (++takeptr == items.length)
            takeptr = 0;
        --count;
        time++;
        if (time > 20) {
            C1.stop();
            C2.stop();
            C3.stop();
            C4.stop();
        }
        System.out.println("+  Stock is " + count);
        return x;
    } finally {
        // quit from core
        mutex.release();
        // add NotEmpty Semaphore, there for allowing to put Item in
        notFull.release();
    }
}
```

四、写在最后

感谢李旭东老师对此次实验的辛勤指导和悉心解答。