利用管程概念求解哲学家进餐问题

詹劲松
(福建师范大学福清分校 电子与信息工程学院,福建 福清 350300)

摘 要:介绍了利用管程概念求解哲学家进餐问题的一种方法,并和我们以前的方法进行了比较,结果表明,引入管程概念使程序的模块化程度和可读性有了很大的提高。

关键词:管程;哲学家进餐问题;高级别并发对象
中图分类号:TP316 **文献标志码**:A **文章编号**:1009-3907(2015)12-0034-04

0 引言

哲学家进餐问题是操作系统中经典的同步问题,它需要在多个进程或线程之间分配多个资源,使进程(线程)能向前推进。我们的目的是要在这种情况下采用某种策略,预防死锁的发生。管程是一种高级的同步构造,利用 Java 高级别并发对象可以实现管程概念。通过管程可以方便地实现这种死锁预防策略。

1 哲学家进餐问题

5 位哲学家围坐在一张圆桌旁边,圆桌中央放置一碗米饭,每两人之间放置一支筷子。每位哲学家思考、饥饿,然后吃饭。为了吃饭,他必须拿起与他相邻的左、右两支筷子。他不能从别的哲学家手里抢夺筷子。吃完饭后,他会放下筷子,并又开始思考。

如果对每个哲学家的吃饭过程不加限制,很快就进入这样一个状态,每人抢得一支筷子,结果谁也吃不了饭,也就是进入了死锁的状态。产生死锁有 4 个必要条件,互斥、占有并等待、非抢占和循环等待[1]。如果能够使一组进程(线程)在推进的整个过程中,这 4 个条件之一或更多保持不成立,那么这组进程(线程)就不会陷入死锁状态。哲学家问题的死锁预防有多种方法。其中一种是:每位哲学家要能取到手边的两支筷子才开始吃饭,否则他一支筷子也不取。这种解法的实质是预先分配需要的全部资源,从而破坏产生死锁的占有并等待这个必要条件。本文就是采用这种解法。

2 管程

为了解决同步问题程序中使用信号量容易出错的问题,70 年代初,P.B.Hansen 和 C. A. R. Hoare 等人提出了管程的概念。其基本思想是:把分散于各进程(线程)中的临界区集中起来统一管理,并把共享资源用数据结构来抽象表示,建立一个管程结构来管理相应的访问[2]。管程结构确保一次只能有一个进程(线程)在管程内活动。

管程结构通过防止对一个资源的并发访问来达到了实现临界区的目的,从而提供了实现互斥的手段,但是管程并没有提供进程(线程)和其他进程(线程)之间同步的途径。当一个进程(线程)进入了管程并调用了管程的一个过程。如果该过程在执行时发现资源不能得到满足,当然应该让此进程(线程)阻塞,同时需要开放管程,让之前被阻挡在管程外边的进程(线程)之一进入。为此需要定义一个另外的同步机制,这可由条件(Condition)结构来提供。条件变量只有操作 wait( ) 和 signal( )。前者用于阻塞调用的进程(线程)。后者用于启动一个被阻塞的进程(线程)。

3 Java 高级别并发对象

在 Java SE 5.0 之前,用 Java 实现管程有些不精确。因为线程之间的同步只能使用 Object 类的 wait( ),和 notify( )或 notifyAll( )来实现。只能向任何一个被阻塞线程或者全部被阻塞线程发送启动消息,不能准确

收稿日期:2015-08-22
基金项目:福建省教育厅 A 类项目(JA14341)
作者简介:詹劲松(1967-),男,福建尤溪人,副教授,硕士,主要从事大数据、操作系统方面的研究。

定位向某一个被阻塞线程发送消息。Java SE 5.0 引入了 ReentrantLock 类和 Condition 接口[3],改变了这个状况。通过调用 Condtiton 对象的 signal 方法,某个哲学家吃完饭,放下筷子就可以准确地向其相邻的两位哲学家线程发出启动信号。

4　算法描述

借助信号量,算法描述如下:

```
Semaphore mutex=1;
Philosopher i:
    while(true){
       p(mutex);
测试该线程状态;
        while(该哲学家吃饭条件不满足){
阻塞该线程;}
V(mutex);
吃饭;
P(mutex)
放下筷子;
测试左右两位哲学家,如果条件满足,启动相关线程;
V(mutex);
思考;
}
```

其 Java 实现代码:

以 Monitor1 类实现管程概念,内含 pickup、putdown 两个方法,供哲学家线程对象调用。这两个方法是互斥的,确保一次只能有一个哲学家线程在管程内活动。运用管程概念,哲学家线程运行过程编程十分简单。只要调用管程相应的方法,就能够保证线程之间的互斥和同步。

```
package monitor;
import java.util.concurrent.locks.*;

enum State {THINKING,HUNGRY,EATING}

public class Monitor1 {//管程类
static State[] state=new State[5];
static Lock lock=new ReentrantLock();
static Condition[] self1=new Condition[5];
{
for(int i=0;i<5;i++){
state[i]=State.THINKING;
      self1[i]=lock.newCondition();
}
}

public void pickup(int i){
lock.lock();
state[i]=State.HUNGRY;
test(i);
```

```
while(state[i]! =State.EATING)
{
try{
self1[i].await();
}catch(InterruptedException e){}
}
lock.unlock();
}

public void putdown(int i){
lock.lock();
state[i]=State.THINKING;
test((i+4)%5);//测试右侧哲学家
test((i+1)%5);//测试左侧哲学家
lock.unlock();
}

public void test(int i){
if((state[(i+4)%5]! =State.EATING)&&
(state[i]==State.HUNGRY)&&
(state[(i+1)%5]! =State.EATING)){
state[i]=State.EATING;
lock.lock();
self1[i].signal();//启动对应的哲学家
lock.unlock();
}
}

}

class Philosopher1  extends Thread{
private static Monitor1 dp1=new Monitor1();
private int i;

public Philosopher1(int i){
this.i=i;
}

public void run(){
while(true){
dp1.pickup(i);
System.out.println("Philosopher "+i+" is eating");
dp1.putdown(i);
System.out.println("Philosopher "+i+" is thinking");
```

```
}
}
}

class Main1{
public static void main(String[] args){
Philosopher[] p=new Philosopher[5];
for(int i=0;i<5;i++){
p[i]=new Philosopher(i);
p[i].start();
}
}
}
```

我们在四核 i5 CPU,4G 内存的计算机,Windows7 平台上运行该程序 4 个小时没有发生死锁和也没有发生饿死。5 个哲学家线程进入吃饭状态的几率差不多。

5　结语

本文利用 Java 高级别对象和管程概念给出了哲学家进餐问题死锁预防的一种解法。与文献 4 中的方法相比[4],借助管程概念求解哲学家进餐问题,把所有的互斥、同步相关代码集中在于管程类内,在更高的层次上解决问题,使代码不容易出错,可读性更好,并且程序模块化的程度更高。

参考文献:

[1] Abraham Silbertschatz, Peter Baer Galvin, Geg Gagne.操作系统概念[M].郑扣根,译.7 版.北京:高等教育出版社,2010.

[2] 费翔林,骆斌.操作系统教程[M].5 版。北京:高等教育出版社,2014.

[3] CayS. Horstmann, Gary Cornell .Java 核心技术卷 1[M].周立新,陈波,叶乃文,等译.北京:机械工业出版社,2013.

[4] 詹劲松.利用 Java 高级别并发对象求解哲学家进餐问题[J].佳木斯大学学报(自然科学版),2013,31(6):905-906.

责任编辑:吴旭云

A Solution to Dining Problem of Philosophers by Utilizing Monitor Concept

ZHAN Jinsong

(School of Electronics and Information Engineering, Fuqing Branch of Fujian Normal University, Fuqing 350300, China)

Abstract: A solution to the dining problem of philosophers by utilizing monitor concept is given, which is compared with our former solution. The result shows that the introduction of monitor concept greatly improves modularization and readability of the program.

Keywords: monitor; dining problem of philosophers; high-grade concurrent object