Lab6实验报告

1. 类概述

共包含8个类，除主类外可以按类的功能分成3个部分

1. Writefile类

用于向txt文件中写入数据

1. CD类和CDstore类

构成需要进行处理的数据的基础结构

1. RentCDthread类、SaleCDthread类、PurchaseCDthread类和Controlthread类

对数据进行多线程处理

1. 类代码功能分析（以实验结果一的源码为例）
2. writefile类

用于将得到的数据输入到txt文件中，在每个线程中都有该类的实例化对象用于输出处理完毕的数据。包含一个用于输出字符串的函数，在线程中通过调用该函数输出需要的信息。通过将FileWriter(file,**true**);构造函数的第二个值置为true使得之前输入过的数据不被覆盖

**class** Writefile

{

**public** **void** writeFile(String str)

{

File file=**new** File("D:/output.txt");

**if**(file.exists()){

**try**{

FileWriter fileWriter=**new** FileWriter(file,**true**);

BufferedWriter out=**new** BufferedWriter(fileWriter);

out.write(str);

out.flush();

out.close();

}

**catch**(FileNotFoundException e)

{

System.***out***.println("找不到文件");

e.printStackTrace();

}

**catch**(IOException e)

{

System.***out***.println("输出流出错");

e.printStackTrace();

}

}

**else**

{

System.***out***.println("file dosen't exist");

}

}

}

1. CD类和CDstore类

CD类中含有num表示该光盘的数量，以及用于增加和减少光盘数量的两个函数。在CDstore中ArrayList的序号即可看作光盘的id，CDstore中含有两个ArrayList<CD>，分别存储用于出租的CD和用于售卖的CD

**class** CD{

**int** num;

**public** CD(**int** thenum)

{

**this**.num=thenum;

}

**public** **void** lessen()

{

num--;

}

**public** **void** add()

{

num++;

}

}

**class** CDstore

{

ArrayList<CD> rentCD;

ArrayList<CD> saleCD;

CDstore(){

rentCD=**new** ArrayList<CD>();

saleCD=**new** ArrayList<CD>();

**for**(**int** i=0;i<10;i++)

{

**this**.rentCD.add(**new** CD(1));

}

**for**(**int** j=0;j<10;j++)

**this**.saleCD.add(**new** CD(10));

}

}

1. RentCDthread类、SaleCDthread类、PurchaseCDthread类和Controlthread类

RentCDthread类、SaleCDthread类、purchaseCDthread类分别用于出租CD、售出CD、和购买CD，Controlthread用于控制这些线程

（1）、RentCDthread

用于借出CD，借出后将对应的Arraylist减1并输出借出信息，然后睡眠200~300ms再归还CD，如果对应的ArrayList为0，则输出CD已被借出的信息，然后睡眠，归还时将对应的ArrayList加1，该类含有存储数据的CDstore theCDstore、生成随机数的i以及用于输出的Writefile output

**class** RentCDthread **extends** Thread

{

CDstore theCDstore;

Random i=**new** Random();

Writefile output=**new** Writefile();

RentCDthread(CDstore aCDstore)

{

**this**.theCDstore=aCDstore;

}

**public** **void** run()

{

**while**(**true**)

{

**synchronized**(**this**.theCDstore.rentCD)

{**int** id=i.nextInt(10);

**if**(**this**.theCDstore.rentCD.get(id).num>0)

{

**this**.theCDstore.rentCD.get(id).lessen();

String s=id+": borrowed "+(**new** Date())+"\r\n";

output.writeFile(s);

**try** {

**this**.*sleep*(i.nextInt(100)+200);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**this**.theCDstore.rentCD.get(id).add();

s=id+": returned "+(**new** Date())+"\r\n";

output.writeFile(s);

**try** {

**this**.*sleep*(i.nextInt(200));

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**else**

{

String str=id+": the CD has been rented "+(**new** Date())+"\r\n";

output.writeFile(str);

**try** {

**this**.*sleep*(i.nextInt(200));

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

//this.theCDstore.rentCD.notifyAll();

}

}

}

}

（2）SaleCDthread类

用于出售CD，含有的属性与RentCDthread类相似，当购买CD的数量大于CD内存时输出库存不足信息，并唤醒所有线程，在库存补足之前一直等待

**class** SaleCDthread **extends** Thread

{

CDstore theCDstore;

Writefile wf=**new** Writefile();

SaleCDthread(CDstore thestore)

{

**this**.theCDstore=thestore;

}

Random i=**new** Random();

**public** **void** run()

{

Random i=**new** Random();

**while**(**true**)

{

**synchronized**(**this**.theCDstore.saleCD)

{

**int** id=i.nextInt(10);

**int** thenum=i.nextInt(5);

**if**(theCDstore.saleCD.get(id).num<thenum)

{

String s=id+":lack of inventory "+**new** Date()+"\r\n";

wf.writeFile(s);

**this**.theCDstore.saleCD.notifyAll();

**while**(theCDstore.saleCD.get(id).num<thenum)

{

**try** {

**this**.theCDstore.saleCD.wait();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

theCDstore.saleCD.get(id).num=theCDstore.saleCD.get(id).num-thenum;

String str=id+": sale "+thenum+" "+**new** Date()+"\r\n";

wf.writeFile(str);

**try** {

**this**.*sleep*(i.nextInt(200));

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

}

（3）PurchaseCDthread类

被SaleCD唤醒或者在1s后自动执行，将ArryList <CD>saleCD中所有存货全部变为10，输出进货信息，然后唤醒其它所有线程

**class** PurchaseCDthread **extends** Thread

{

CDstore theCDstore;

Random i=**new** Random();

Writefile wf=**new** Writefile();

PurchaseCDthread(CDstore aCDstore)

{

**this**.theCDstore=aCDstore;

}

**public** **void** run()

{

**while**(**true**)

{

**synchronized**(**this**.theCDstore.saleCD)

{

**for**(**int** j=0;j<10;j++)

{

**int** thenum=**this**.theCDstore.saleCD.get(j).num;

**if**(thenum<10)

{

String s="purchaseCD "+j+":"+(10-thenum)+(**new** Date())+"\r\n";

wf.writeFile(s);

theCDstore.saleCD.get(j).num=10;

}

}

**this**.theCDstore.saleCD.notifyAll();

**try** {

**this**.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

}

（4）Controlthread

生成需要的出租、出售、购买线程，保证这些线程对同一个CDstore对象进行操作，并且将这些线程设置为守护线程，这样这些线程在主线程停止后会自动停止，由于购买线程执行的比较少，因此设立了3个购买线程

**class** Controlthread **extends** Thread

{

CDstore theCDstore;

Controlthread(CDstore aCDstore)

{

**this**.theCDstore=aCDstore;

**this**.setDaemon(**true**);

}

**public** **void** run()

{

**for**(**int** i=0;i<2;i++)

{

Thread t=**new** RentCDthread(theCDstore);

t.setDaemon(**true**);

t.start();

}

**for**(**int** i=0;i<3;i++)

{

Thread t=**new** SaleCDthread(theCDstore);

t.setDaemon(**true**);

t.start();

}

Thread t=**new** PurchaseCDthread(theCDstore);

t.setDaemon(**true**);

t.start();

}

}

1. 实验结果分析

实验结果见附件1、2、3

1. 在实验结果1中，经常出现一长段的借出和归还；如：

0: returned Wed May 16 20:42:40 CST 2018

8: borrowed Wed May 16 20:42:40 CST 2018

8: returned Wed May 16 20:42:41 CST 2018

1: borrowed Wed May 16 20:42:41 CST 2018

1: returned Wed May 16 20:42:41 CST 2018

1: borrowed Wed May 16 20:42:41 CST 2018

1: returned Wed May 16 20:42:41 CST 2018

9: borrowed Wed May 16 20:42:42 CST 2018

9: returned Wed May 16 20:42:42 CST 2018

2: borrowed Wed May 16 20:42:42 CST 2018

2: returned Wed May 16 20:42:42 CST 2018

6: borrowed Wed May 16 20:42:42 CST 2018

6: returned Wed May 16 20:42:43 CST 2018

8: borrowed Wed May 16 20:42:43 CST 2018

8: returned Wed May 16 20:42:43 CST 2018

这是因为在这篇代码中上锁的对象是一整个数组，原本saleCD线程应该可以对没有被其它saleCD线程占用的ArrayList中的其它数进行处理，但是由于一整个ArrayList都被锁住了，所以这些本可以售出的线程被阻塞，然后等待，这个时候操作系统可能调用其它线程，导致sale线程一直往后推，所以输出了很多租借的结果，可以将上锁的对象改为具体更改的ArrayList中的某个成员来对这样的情况进行处理

1. 在实验结果1中，依然是以上片段，出现了许多比如先借CD2，然后归还CD2中间没有其它结果的情况，这是因为在RentCDthread中，依然是ArrayList<CD> rentCD上锁，导致只有一个租借线程在执行，这时候的程序相当于不是并行而是串行，在该线程睡眠期间，整个ArrayList依旧被锁住，其它租借线程只能等到该线程执行完毕，释放锁，更改后得到实验结果2
2. 实验结果2中则会出现在后面一段中，大多数都是租借记录而没有出售记录的情况，例如：

2: returned Thu May 17 00:00:16 CST 2018

4: borrowed Thu May 17 00:00:16 CST 2018

5: borrowed Thu May 17 00:00:16 CST 2018

4: returned Thu May 17 00:00:16 CST 2018

5: returned Thu May 17 00:00:16 CST 2018

4: borrowed Thu May 17 00:00:16 CST 2018

8: borrowed Thu May 17 00:00:17 CST 2018

4: returned Thu May 17 00:00:17 CST 2018

8: returned Thu May 17 00:00:17 CST 2018

1: borrowed Thu May 17 00:00:17 CST 2018

0: borrowed Thu May 17 00:00:17 CST 2018

这是因为在PurchaseCDthread中，由于要将所有进货补齐，将ArrayList<CD> saleCD上锁，这个时候可能有saleCD线程正在占用ArrayList的成员，这个时候就出现了死锁，导致购买和出售线程都不再执行，只执行出租线程，将分别锁加在成员上，得到实验结果3。由于并行的线程增多，可以看出3的记录远远长于实验结果1和实验结果2。

1. 源码（实验结果三）

package lab6;

import java.util.\*;

import java.util.Date;

import java.util.ArrayList;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.io.File;

import java.io.BufferedWriter;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

public class Main{

public static void main(String[] arg){

Scanner out=new Scanner(System.in);

CDstore theCDstore=new CDstore();

Writefile wf=new Writefile();

Controlthread ct=new Controlthread(theCDstore);

ct.start();

try {

Thread.sleep(120000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

class Writefile

{

public void writeFile(String str)

{

File file=new File("D:/output.txt");

if(file.exists()){

try{

FileWriter fileWriter=new FileWriter(file,true);

BufferedWriter out=new BufferedWriter(fileWriter);

out.write(str);

out.flush();

out.close();

}

catch(FileNotFoundException e)

{

System.out.println("找不到文件");

e.printStackTrace();

}

catch(IOException e)

{

System.out.println("输出流出错");

e.printStackTrace();

}

}

else

{

System.out.println("file dosen't exist");

}

}

}

class CD{

int num;

public CD(int thenum)

{

this.num=thenum;

}

public void lessen()

{

num--;

}

public void add()

{

num++;

}

}

class CDstore

{

ArrayList<CD> rentCD;

ArrayList<CD> saleCD;

CDstore(){

rentCD=new ArrayList<CD>();

saleCD=new ArrayList<CD>();

for(int i=0;i<10;i++)

{

this.rentCD.add(new CD(1));

}

for(int j=0;j<10;j++)

this.saleCD.add(new CD(10));

}

}

class RentCDthread extends Thread

{

CDstore theCDstore;

Random i=new Random();

Writefile output=new Writefile();

RentCDthread(CDstore aCDstore)

{

this.theCDstore=aCDstore;

}

public void run()

{

while(true)

{

int id=i.nextInt(10);

if(this.theCDstore.rentCD.get(id).num>0)

{

synchronized(this.theCDstore.rentCD.get(id))

{

this.theCDstore.rentCD.get(id).lessen();

String s=id+": borrowed "+(new Date())+"\r\n";

output.writeFile(s);

try {

this.sleep(i.nextInt(100)+200);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

this.theCDstore.rentCD.get(id).add();

s=id+": returned "+(new Date())+"\r\n";

output.writeFile(s);

}

try {

this.sleep(i.nextInt(200));

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

else

{

String str=id+": the CD has been rented "+(new Date())+"\r\n";

output.writeFile(str);

try {

this.sleep(i.nextInt(200));

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

}

}

class SaleCDthread extends Thread

{

CDstore theCDstore;

Writefile wf=new Writefile();

SaleCDthread(CDstore thestore)

{

this.theCDstore=thestore;

}

Random i=new Random();

public void run()

{

Random i=new Random();

while(true)

{

int id=i.nextInt(10);

int thenum=i.nextInt(5);

synchronized(this.theCDstore.saleCD.get(id))

{

if(theCDstore.saleCD.get(id).num<thenum)

{

String s=id+":lack of inventory "+new Date()+"\r\n";

wf.writeFile(s);

this.theCDstore.saleCD.get(id).notifyAll();

while(theCDstore.saleCD.get(id).num<thenum)

{

try {

this.theCDstore.saleCD.get(id).wait();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

theCDstore.saleCD.get(id).num=theCDstore.saleCD.get(id).num-thenum;

String str=id+": sale "+thenum+" "+new Date()+"\r\n";

wf.writeFile(str);

}

try {

this.sleep(i.nextInt(200));

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

}

class PurchaseCDthread extends Thread

{

CDstore theCDstore;

Random i=new Random();

Writefile wf=new Writefile();

PurchaseCDthread(CDstore aCDstore)

{

this.theCDstore=aCDstore;

}

public void run()

{

while(true)

{

for(int j=0;j<10;j++)

{

int thenum=this.theCDstore.saleCD.get(j).num;

if(thenum<10)

{

synchronized(this.theCDstore.saleCD.get(j))

{

String s="purchaseCD "+j+":"+(10-thenum)+(new Date())+"\r\n";

wf.writeFile(s);

theCDstore.saleCD.get(j).num=10;

this.theCDstore.saleCD.get(j).notifyAll();

}

}

}

try {

this.sleep(1000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

}

class Controlthread extends Thread

{

CDstore theCDstore;

Controlthread(CDstore aCDstore)

{

this.theCDstore=aCDstore;

this.setDaemon(true);

}

public void run()

{

for(int i=0;i<2;i++)

{

Thread t=new RentCDthread(theCDstore);

t.setDaemon(true);

t.start();

}

for(int i=0;i<2;i++)

{

Thread t=new SaleCDthread(theCDstore);

t.setDaemon(true);

t.start();

}

Thread t=new PurchaseCDthread(theCDstore);

t.setDaemon(true);

t.start();

}

}