

**面向对象程序设计上机实验报告**

实验题目5： 多线程程序开发

**学院名称 智能与计算学部**

**专 业 软件工程**

**学生姓名 王杨森**

**学 号 3020244116**

**年 级 2020级**

**班 级 软件工程3班**

**时 间 2022年 04月28日**

1. **实验目的**
2. 熟悉并理解Java多线程编程；
3. 理解并熟悉线程之间控制的方法；
4. 用多线程解决实际问题；
5. **实验内容**
6. 使用多线程方式，计算某个区间的质数。
7. **程序实现**

**public** **class** ThreadUtil **extends** Thread {

**private** StringBuffer buffer;

**public** ThreadUtil(StringBuffer buf) {

**this**.buffer = buf;

}

@Override

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

**try** {

Thread.*sleep*(100);//设置在0-1000ms内

} **catch**(InterruptedException e) {

e.printStackTrace();

} **finally** {

**this**.buffer.append("ok");

}

}

}

**public** **class** PrimeUtil {

**public** PrimeUtil() {

}

**public** List<Long> getPrimeList(**long** start, **long** end, **int** threadCount) {

List<Long> res = **new** ArrayList<>();

PrimeThread[] threads = **new** PrimeThread[threadCount]; //分成多个线程

**for**(**int** i = 0; i < threadCount; i++) {

**long** start1 = (end - start) / threadCount \* i + start;

**long** end1 = (end - start) / threadCount \* (i + 1) + start;

threads[i] = **new** PrimeThread(start1, end1); //设置每个线程的区间，使每个线程在一个区间范围内计算

}

**for**(**int** i = 0; i < threadCount; i++) {

threads[i].start();

}

**for**(**int** i = 0; i < threadCount; i++) {

**try** {

threads[i].join();//需要依次等待所有线程结束

res.addAll(threads[i].getPrime());//加入每个线程的结果

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**return** res;

}

}

**public** **class** PrimeThread **extends** Thread{

**private** **final** Long start;

**private** **final** Long end;

**private** List<Long> res = **new** ArrayList<>();

**public** PrimeThread(Long start, Long end) {

**this**.start = start;

**this**.end = end;

}

//判断是否为素数

**public** **boolean** isPrime(Long num) {

**for**(**long** i = 2; i <= Math.*sqrt*(num); i++) {

**if**(num % i == 0) {

**return** **false**;

}

}

**return** **true**;

}

**public** List<Long> getPrime() {

**return** **this**.res;

}

@Override

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

//线程运行，遍历区间，如果是素数则加入list中

**for**(**long** i = **this**.start; i < **this**.end; i++) {

**if**(isPrime(i)) {

res.add(i);

}

}

}

}

1. **实验结果**

测试代码：

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**try** {

StringBuffer buf = **new** StringBuffer();

/\*\*

\* 作业一 测试程序输入一个StringBuffer，刚输入的时候StringBuffer值为空

\* 等待1000ms后，这个StringBuffer的值变为"ok"

\*/

ThreadUtil threadUtil = **new** ThreadUtil(buf);

threadUtil.start();

**if** (buf.length() > 0) {

System.***out***.println("第一小题 错误,现在不能给buf值必须为空");

}

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

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

e.printStackTrace();

}

**if** (buf.toString().equalsIgnoreCase("ok")) {

System.***out***.println("第一小题 正确，测试程序在休眠1000毫秒以后，需要从buf中得到\"ok\"值");

}

/\*\*

\* 作业二，多线程方式，统计[start,end)区间所有的质数，并且小到大排序返回

\*/

PrimeUtil util = **new** PrimeUtil();

//测试的时候 start、end、threadCount都有可能变化

**long** start = 1000000L;//测试的时候可能为大于2的任何一个整数

**long** end = start\*2;//测试的时候 可能为大于start的任何一个整数

**int** threadCount = 4;//测试的时候取值可能为[1,20]之间

**long** l=System.*currentTimeMillis*();

//多线程计算

List<Long> list = util.getPrimeList(start, end, threadCount);

// for(int i = 0; i <list.size();i = i+10000){

// System.out.println(list.get(i));

// }

//要求算法具有一定的效率速度

**long** cost=(System.*currentTimeMillis*()-l);

System.***out***.println(cost+"耗时 "+cost +" ms! 参考：Intel I5-7200U 耗时700ms 左右!" );

**if**(cost>1500) {

System.***out***.println("计算速度太慢或者使用了电池供电， 请设法优化你的程序或者请在使用外接电源情况下跑你的程序！");

}

//测试的时候，会随机的从list中检查一些结果，本例子检查的是0,10001,20001...位置的质数

**long**[] result= {1000003,1138589,1278481,1420121,1561801,1705141,1849319,1993921};

**int** j=0;

**boolean** flag=**true**;

**for**(**int** i=0;i<list.size();i=i+10000) {

**if**(result[j]!=list.get(i)) {

System.***out***.print ("第"+i+"个质数错误!");

flag=**false**;

}

j++;

}

**if**(flag) {

System.***out***.println("第二小题测试正确");

}

} **catch** (Exception e) {

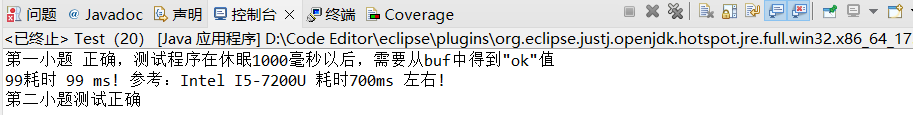
// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

测试结果：



1. **实验中遇到的问题及解决方法**

最开始在做第一个任务时，将线程等待时间设为了1000ms，但其实这样是不对的，因为后面的程序也等待了1000ms，很有可能线程还没运算完就进行了结果判断，所以将等待时间修改为100ms。

在第二个任务和利用多线程进行排序很相似，都是将区间进行拆分，划分区域利用线程计算。一定要记得线程启动运行后要join，并且要安装顺序依次join，否则不能实现按照顺序进行存储。

其余的编程过程都还算很顺利。