

华东师范大学数据科学与工程学院实验报告

课程名称：计算机网络与编程

年级：21 级

上机实践成绩：

指导教师：张召

姓名：杨茜雅

学号：10215501435

上机实践名称： Lab05

上机实践日期：2023.3.30

上机实践编号：5

组号：

上机实践时间：3.30-4.7

一、实验目的

熟悉Java多线程编程

熟悉并掌握线程同步和线程交互

二、实验任务

学习使用synchronized 关键字

学习使用wait() 、notify() 、notifyAll() 方法进行线程交互

三、使用环境

IntelliJ IDEA

JDK 版本: Java 19

四、实验过程

Task 1: 对Lab4的3.2中给出的PlusMinus 、TestPlus 、Plus 代码，使用synchronized 关键字进行修改，使用两种不同的修改方式，使得num值不出现线程处理不同步的问题，将实现代码段及运行结果附在实验报告中。

方法一：

```
package Task_1;

public class PlusMinus {
    public volatile int num;
    public void plusOne(){
        num = num + 1;
    } public void minusOne(){
        num = num - 1;
    } public int printNum(){
        return num;
    }
}

class TestPlusMinus {
    public static void main(String[] args){
        PlusMinus plusMinus = new PlusMinus();
        plusMinus.num = 1000;
        int threadNum = 1000;
        Thread[] plusThreads = new Thread[threadNum];
        Thread[] minusThreads = new Thread[threadNum];
        for(int i = 0; i < threadNum; i++){
            Thread thread1 = new Thread(){
                @Override
                public void run(){
                    try {
                        sleep( millis: 1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    } synchronized (
                        PlusMinus.class){
                            plusMinus.plusOne();}
                }
            };
            Thread thread2 = new Thread(){
```

```

    });
    Thread thread2 = new Thread(){
        @Override
        public void run(){
            try {
                sleep( millis: 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            } synchronized (
                PlusMinus.class){
                plusMinus.minusOne();}
        }
    };
    thread1.start();
    thread2.start();
    plusThreads[i] = thread1;
    minusThreads[i] = thread2;
} for(Thread thread:plusThreads){
    try {
        thread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
for(Thread thread:minusThreads){try {
    thread.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}
}
System.out.println("所有线程结束后的num 值为:  " +
    plusMinus.printNum());
}
}

```

运行结果:

```

C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files
所有线程结束后的num 值为:  1000

```

方法二:

```

package Task_12;
public class PlusMinus {
    public volatile int num;
    public synchronized void plusOne(){
        num = num + 1;
    } public synchronized void minusOne(){
        num = num - 1;
    } public synchronized int printNum(){
        return num;}
}
class TestPlusMinus {
    public static void main(String[] args){
        PlusMinus plusMinus = new PlusMinus();
        plusMinus.num = 1000;
        int threadNum = 1000;
        Thread[] plusThreads = new Thread[threadNum];
        Thread[] minusThreads = new Thread[threadNum];
        for(int i = 0; i < threadNum; i++){
            Thread thread1 = new Thread(){
                @Override
                public void run(){
                    try {
                        sleep( millis: 1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    } plusMinus.plusOne();}
            };
            Thread thread2 = new Thread(){
                @Override
                public void run(){

```

```
@Override
public void run(){
    try {
        sleep( millis: 1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    } plusMinus.plusOne();}
};

Thread thread2 = new Thread(){
    @Override
    public void run(){
        try {
            sleep( millis: 1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } plusMinus.minusOne();}
};

thread1.start();
thread2.start();
plusThreads[i] = thread1;
minusThreads[i] = thread2;
} for(Thread thread:plusThreads){
    try {
        thread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
for(Thread thread:minusThreads){try {
    thread.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}
}
System.out.println("所有线程结束后的num 值为:  " +
    plusMinus.printNum());
```

运行结果:

```
C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:
所有线程结束后的num 值为:  1000
```

Task 2: 给出以下TestMax 、MyThread 、Res 代码，使用synchronized 关键字在TODO 处进行修改，实现最后打印出的res.max_idx 的值是所有MyThread对象的list 中保存的数的最大值，将实现代码段及运行结果附在实验报告中。

```
package Task_2;

import java.util.ArrayList;
import java.util.Random;

public class TestMax {
    public static void main(String[] args) throws InterruptedException {
        Res res = new Res();
        int threadNum = 3;
        Thread[] threads = new Thread[threadNum];
        for (int i = 0; i < threadNum; i++) {
            threads[i] = new Thread(new MyThread(i, res));
        }
        for (int i = 0; i < threadNum; i++) {
            threads[i].start();
        }
        for (int i = 0; i < threadNum; i++) {
            threads[i].join();
        }
        System.out.println(res.max_idx);
    }
}

class MyThread implements Runnable {
    static int[] seeds = {1234567, 2345671, 3456712};
    MyThread(int id, Res _res) {
        Random r = new Random(seeds[id]);
        list = new ArrayList<>();
        for (int i = 0; i < 100; i++) {
            list.add(r.nextInt( bound: 10000));
        }
        res = _res;
    }
    @Override
    public void run() {
        Byte Interger = null;
        int max_val = Integer.MIN_VALUE;
        for(int i = 0; i< list.size();i++){
            if(list.get(i) > max_val){
                max_val = list.get(i);
            }
        }
        synchronized(res){
            if(max_val > res.max_idx){
                res.max_idx = max_val;
            }
        }
    }
    ArrayList<Integer> list;
    Res res;
}

class Res {
    int max_idx;
}
```

运行截图：

```
C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:E:\P
9951

Process finished with exit code 0
```

Task 3: 设计3个线程彼此死锁的场景并编写代码(可基于上述代码或自己编写)，将实现代码段及运行结果附在实验报告中。

```
package Task_3;
public class DeadLock {
    public static void main(String[] args){
        PlusMinus plusMinus1 = new PlusMinus();
        plusMinus1.num = 1000;
        PlusMinus plusMinus2 = new PlusMinus();
        plusMinus2.num = 1000;
        PlusMinus plusMinus3 = new PlusMinus();
        plusMinus3.num = 1000;
        Thread thread1 = new Thread(){
            @Override
            public void run(){
                synchronized (plusMinus1){
                    System.out.println("thread1 正在占用plusMinus1");
                    try {
                        sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    System.out.println("thread1 试图继续占用 plusMinus2");
                    synchronized (plusMinus2){
                        System.out.println("thread1 成功占用plusMinus2 ");
                    }
                    System.out.println("thread1 试图继续占用 plusMinus3");
                    synchronized (plusMinus3){
                        System.out.println("thread1 成功占用plusMinus3 ");
                    }
                }
            }
        };
        thread1.start();
        Thread thread2 = new Thread(){@Override
```

```

Thread thread2 = new Thread(){@Override
public void run(){
    synchronized (plusMinus2){
        System.out.println("thread2 正在占用plusMinus2");
        try {
            sleep( millis: 1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } System.out.println("thread2 试图继续占用 plusMinus1");
        synchronized (plusMinus1){
            System.out.println("thread2 成功占用plusMinus1 ");
        } System.out.println("thread2 plusMinus3");
        synchronized (plusMinus3){
            System.out.println("thread2 成功占用plusMinus3 ");
        }
    }
}};
thread2.start();
Thread thread3 = new Thread(){@Override
public void run(){
    synchronized (plusMinus3){
        System.out.println("thread3 正在占用plusMinus3");
        try {
            sleep( millis: 1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } System.out.println("thread3 试图继续占用 plusMinus1");
        synchronized (plusMinus1){
            System.out.println("thread3 成功占用plusMinus1 ");
        } System.out.println("thread3 试图继续占用 plusMinus2");
        synchronized (plusMinus2){
            System.out.println("thread3 成功占用plusMinus2 ");
        }
    }
}};
thread3.start();
}
}

```

运行结果:

```

C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files\JetBr
thread1 正在占用plusMinus1
thread2 正在占用plusMinus2
thread3 正在占用plusMinus3
thread1 试图继续占用 plusMinus2
thread2 试图继续占用 plusMinus1
thread3 试图继续占用 plusMinus1

```

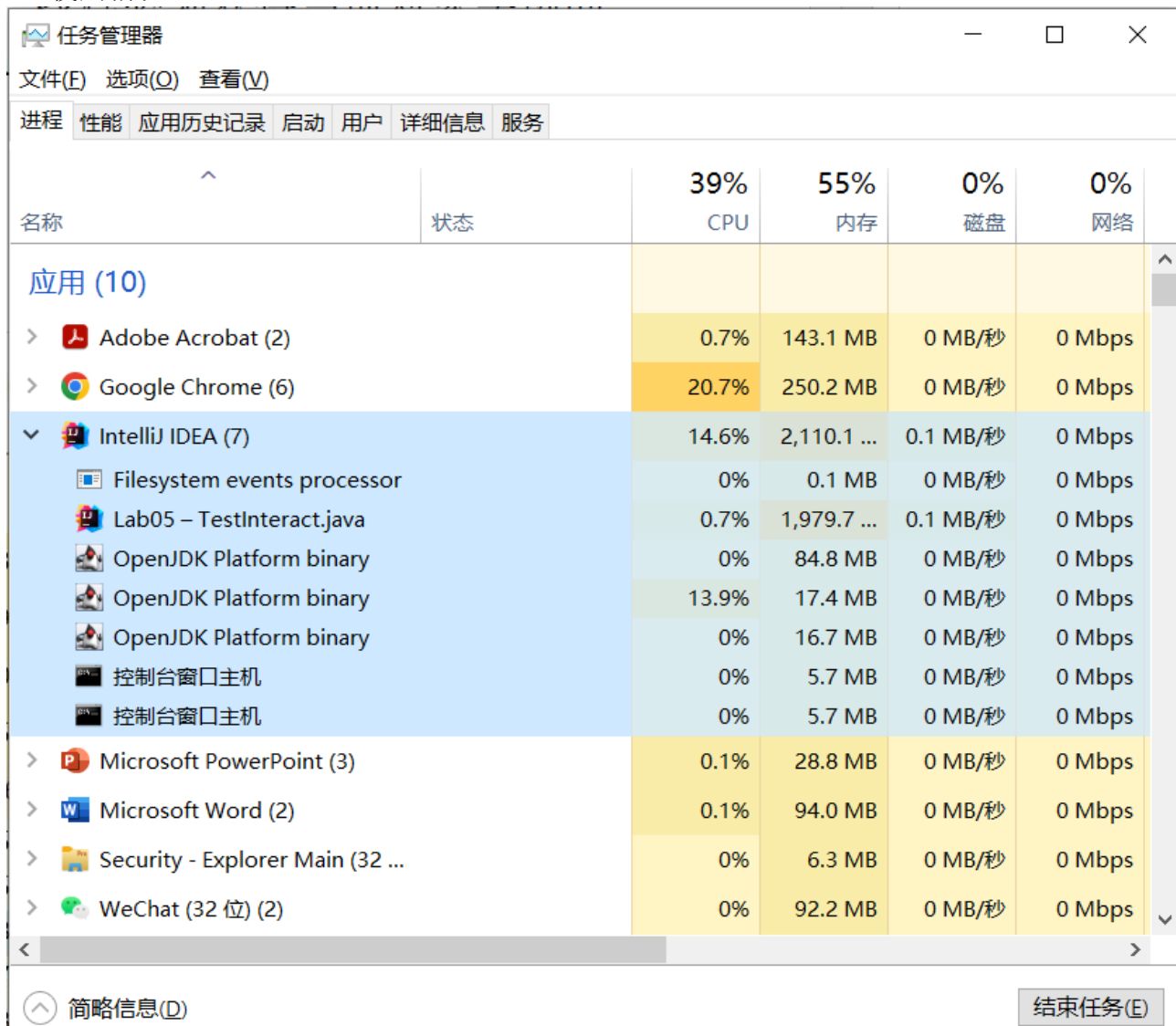
Task 4:

首先阐述**synchronized** 在实例方法上的作用，然后运行本代码段，同时打开检测cpu的工具，观察cpu的使用情况，将实验结果和cpu使用情况截图附在实验报告中。

作用：锁定了整个方法时的内容。在进入实例方法前，线程必须获得当前对象实例的锁。当两个并发线程(thread1 和thread2)访问同一个对象(plusMinus)中的实例方法时，在

同一时刻只能有一个线程得到执行，另一个线程受阻塞。

CPU使用情况：



名称	状态	39% CPU	55% 内存	0% 磁盘	0% 网络
应用 (10)					
> Adobe Acrobat (2)		0.7%	143.1 MB	0 MB/秒	0 Mbps
> Google Chrome (6)		20.7%	250.2 MB	0 MB/秒	0 Mbps
▼ IntelliJ IDEA (7)		14.6%	2,110.1 ...	0.1 MB/秒	0 Mbps
Filesystem events processor		0%	0.1 MB	0 MB/秒	0 Mbps
Lab05 - TestInteract.java		0.7%	1,979.7 ...	0.1 MB/秒	0 Mbps
OpenJDK Platform binary		0%	84.8 MB	0 MB/秒	0 Mbps
OpenJDK Platform binary		13.9%	17.4 MB	0 MB/秒	0 Mbps
OpenJDK Platform binary		0%	16.7 MB	0 MB/秒	0 Mbps
控制台窗口主机		0%	5.7 MB	0 MB/秒	0 Mbps
控制台窗口主机		0%	5.7 MB	0 MB/秒	0 Mbps
> Microsoft PowerPoint (3)		0.1%	28.8 MB	0 MB/秒	0 Mbps
> Microsoft Word (2)		0.1%	94.0 MB	0 MB/秒	0 Mbps
> Security - Explorer Main (32 ...		0%	6.3 MB	0 MB/秒	0 Mbps
> WeChat (32 位) (2)		0%	92.2 MB	0 MB/秒	0 Mbps

Task 5: 在**Task4**基础上增加若干减一操作线程，运行久一点，观察有没有发生错误。若有，请分析错误原因，给出解决代码。

有错误。

错误：**num** 会出现负数。因为存在多个减一线程，假设**n** 个减一线程在同一时刻读取**num** 值，而此时**num** 值小于**n**，则进程结束后**num** 会为负数。

解决方法：

对于**n** 个减法线程，其中**n-1** 个线程在**num==1** 成立后直接**break**；另一个减法线程在**num==1** 后将**num** 重新赋值**n**，**continue** 循环。

```

package Task_4;

public class TestInteract {
    public static void main(String[] args) {
        PlusMinusOne pmo = new PlusMinusOne();
        PlusMinus plusMinus = new PlusMinus();
        pmo.num = 50;
        plusMinus.num = 0;
        Thread thread1 = new Thread(){
            @Override
            public void run(){
                while (true){
                    if(plusMinus.num <= 1){
                        plusMinus.num = 2;
                        continue;
                    } plusMinus.minusOne();
                    try {
                        sleep(10); //减法线程10ms
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
        thread1.start();
        Thread thread3 = new Thread(){
            @Override
            public void run(){
                while (true){
                    if(plusMinus.num <= 1){
                        break;
                    } plusMinus.minusOne();
                    try {
                        sleep(10); //减法线程10ms
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
        thread3.start();
    }
}

```

Task 6: 在以下代码中加入若干获取product的线程，并运行截图；之后将while (productQueue.isEmpty()) 修改为if (productQueue.isEmpty())，并观察运行结果，如发生错误，试分析原因。

运行截图：


```
t1 get product: product
t2 add product: product
t1 get product: product
t2 add product: product
t1 get product: product
t2 add product: product
t1 get product: product
t2 add product: product
t2 add product: product
t1 get product: product
t2 add product: product
t1 get product: product
t1 get product: product
t2 add product: product
```

修改后运行截图：

```
t1 get product: product
t2 add product: product
t2 add product: product
t1 get product: product
t2 add product: product
t1 get product: product
t2 add product: product
t1 get product: product
t1 get product: product
t2 add product: product
t1 get product: product
t2 add product: product
t1 get product: product
t2 add product: product
```

原因分析：读写线程分别为get和add方法的实现，写锁被写线程获取以后，读写线程都会被阻塞，读线程不能获取到最新的数据，所以不能满足数据的最终一致性。

Task 7: 在**Task5**的基础上，使用**wait**和**notify**修改代码，达到一致的代码逻辑，同时打开检测**cpu**的工具，观察**cpu**的使用情况，将实验结果和**cpu**使用情况截图附在实验报告中。

```
package Task_4;

class PlusMinus02 {
    public volatile int num;
    public void plusOne(){
        synchronized (this){
            num = num + 1;
            printNum();
        }
    }
    public void minusOne() {
        synchronized (this) {
            num = num - 1;
            printNum();
        }
    }
    public void printNum(){
        System.out.println("num = " + num);
    }
}

public class InteractTest {
    public static void main(String[] args){
        PlusMinus02 plusMinus = new PlusMinus02();
        plusMinus.num = 1000;
        Object obj= new Object();
        Thread thread1 = new Thread(){
            @Override
            public void run(){
                while (true){
                    synchronized(obj){
                        plusMinus.minusOne();
                        if(plusMinus.num==1){
```

```
        plusMinus.minusOne();
        if(plusMinus.num==1){
            try {
                obj.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        } try {
            sleep( millis: 10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}










thread1.start();
Thread thread2 = new Thread(){
    @Override
    public void run(){
        while (true){
            synchronized(obj){
                plusMinus.plusOne();
                obj.notifyAll();
            } try {
                sleep( millis: 100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

thread2.start();
}
```

运行结果:

```
C:\Users\86138\.jdk\openjdk-19.0.2\bin\java.exe "-ja
num = 999
num = 1000
num = 999
num = 998
num = 997
num = 996
num = 995
num = 994
num = 993
num = 994
num = 993
num = 992
num = 991
num = 990
num = 989
num = 988
num = 987
num = 988
num = 987
num = 986
num = 985
```

Cpu使用情况:

名称	状态	27% CPU	59% 内存	0% 磁盘	0% 网络
应用 (10)					
>  Adobe Acrobat (2)		0%	183.0 MB	0 MB/秒	0 Mbps
>  Google Chrome (13)		23.5%	227.5 MB	0.1 MB/秒	0 Mbps
▼  IntelliJ IDEA (6)		0.1%	2,020.9 ...	0 MB/秒	0 Mbps
 Lab05 – InteractTest.java		0.1%	1,962.8 ...	0 MB/秒	0 Mbps
 OpenJDK Platform binary		0%	1.2 MB	0 MB/秒	0 Mbps
 OpenJDK Platform binary		0%	16.4 MB	0 MB/秒	0 Mbps
 OpenJDK Platform binary		0%	16.4 MB	0 MB/秒	0 Mbps
 OpenJDK Platform binary		0%	18.4 MB	0 MB/秒	0 Mbps
 控制台窗口主机		0%	5.7 MB	0 MB/秒	0 Mbps

五、总结

通过本次实验，我学习了使用`synchronized` 以及使用`wait()` 、`notify()` 、`notifyAll()` 方法进行线程交互，为以后的java学习打下基础。