死锁和死循环会导致cpu升高吗

典型回答

死循环会导致CPU使用率升高，死锁不会导致升高，甚至可能降低。
原因：死循环的代码依旧会在CPU上执行，而死锁的线程会被挂起，不会占用CPU。

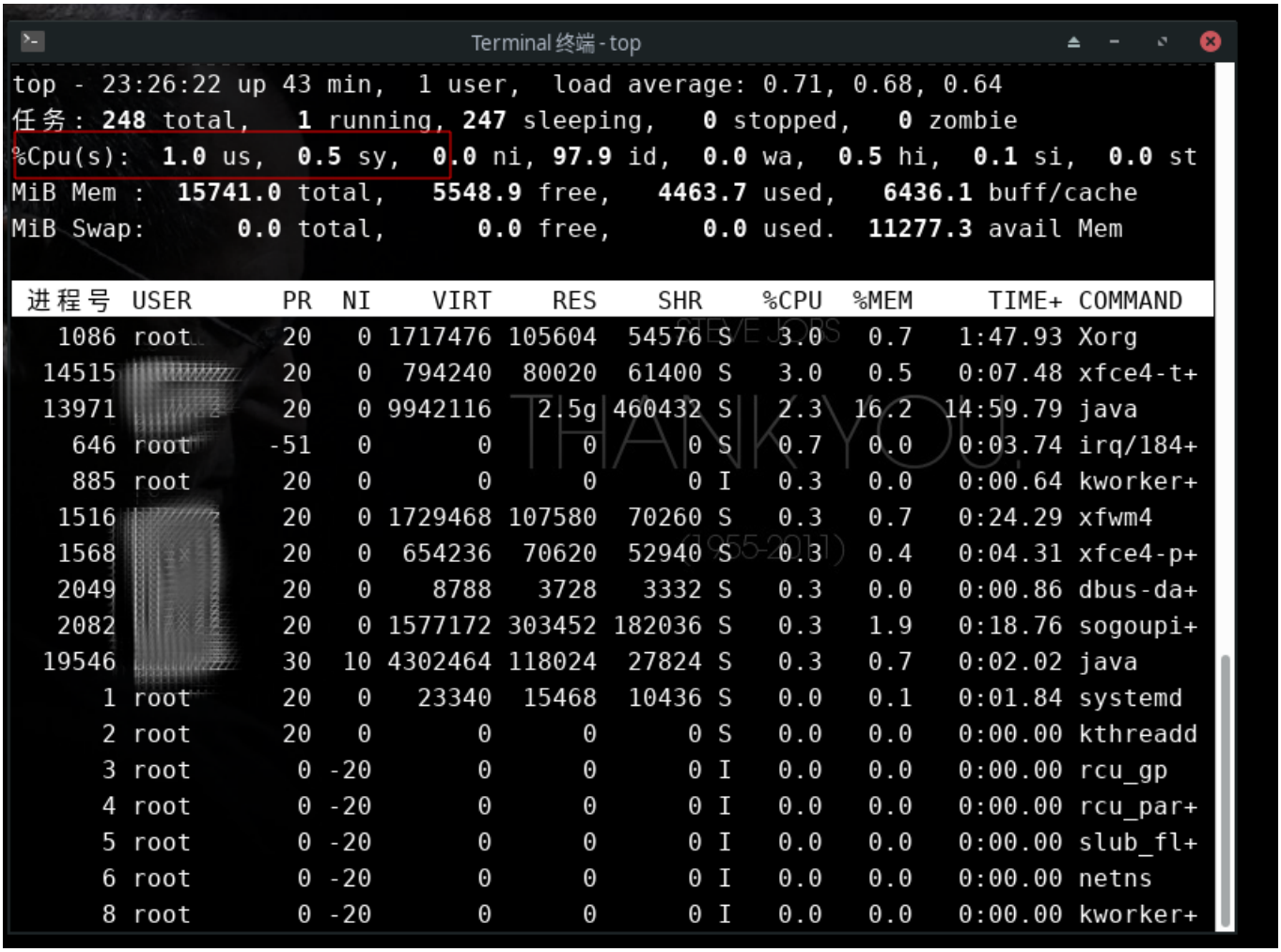
扩展知识

我们可以通过实验验证上面的结论，以下实验在8C16G的物理机上实现。图中total=15g，少了1g是因为Linux内核分配给SLAB了。

编写死循环代码：

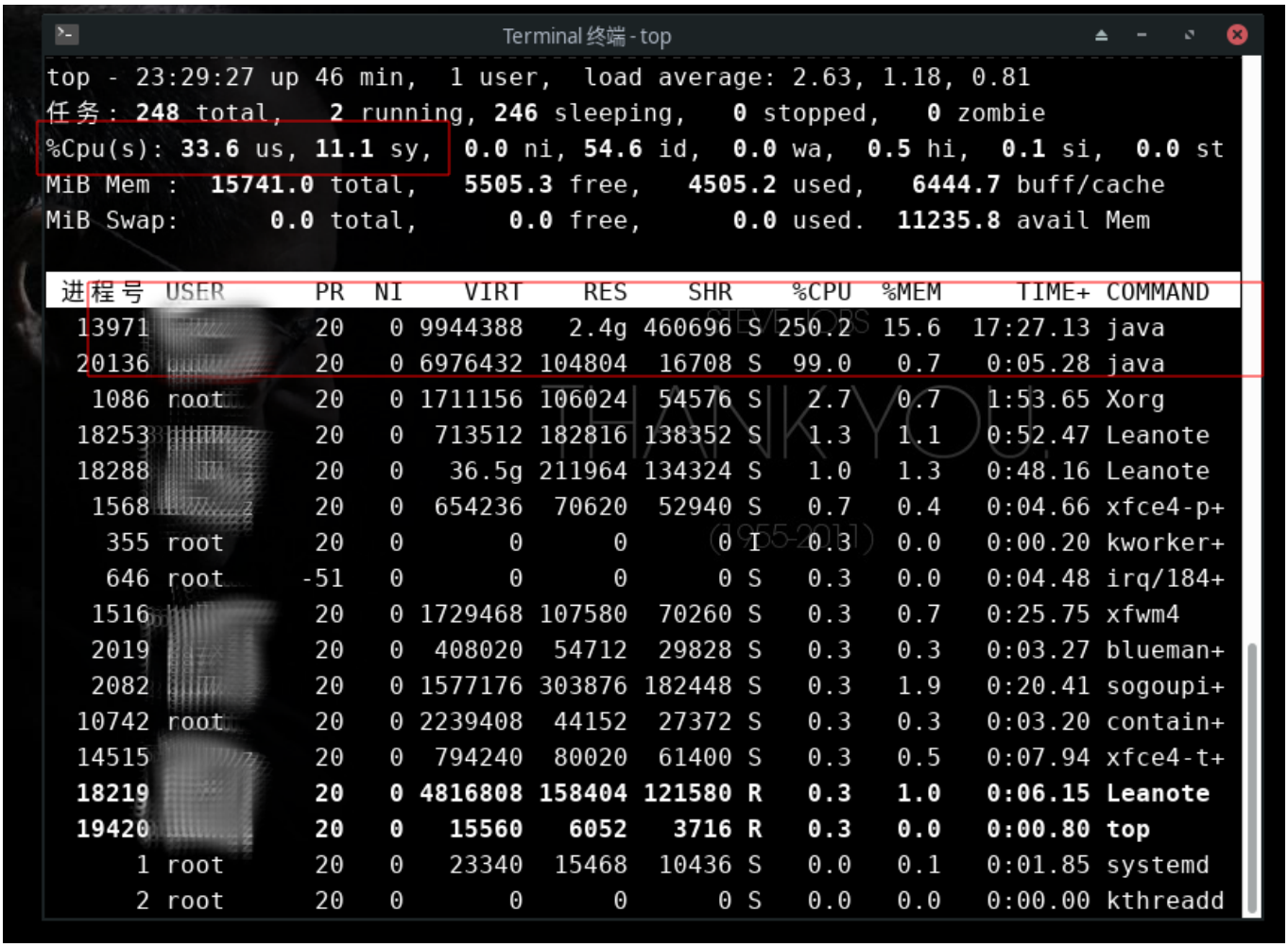
```
public class Main {
    public static void main(String[] args) {
        while (true){
            System.out.println(System.currentTimeMillis());//死循环输出系统时间戳，强制产生2个系统调用，可以看到CPU us 和sy 占比都会升高
        }
    }
}
```

没有跑以上代码之前的CPU使用率下图所示，`us`代表用户态1%，`sy`代表内核态0.5%。

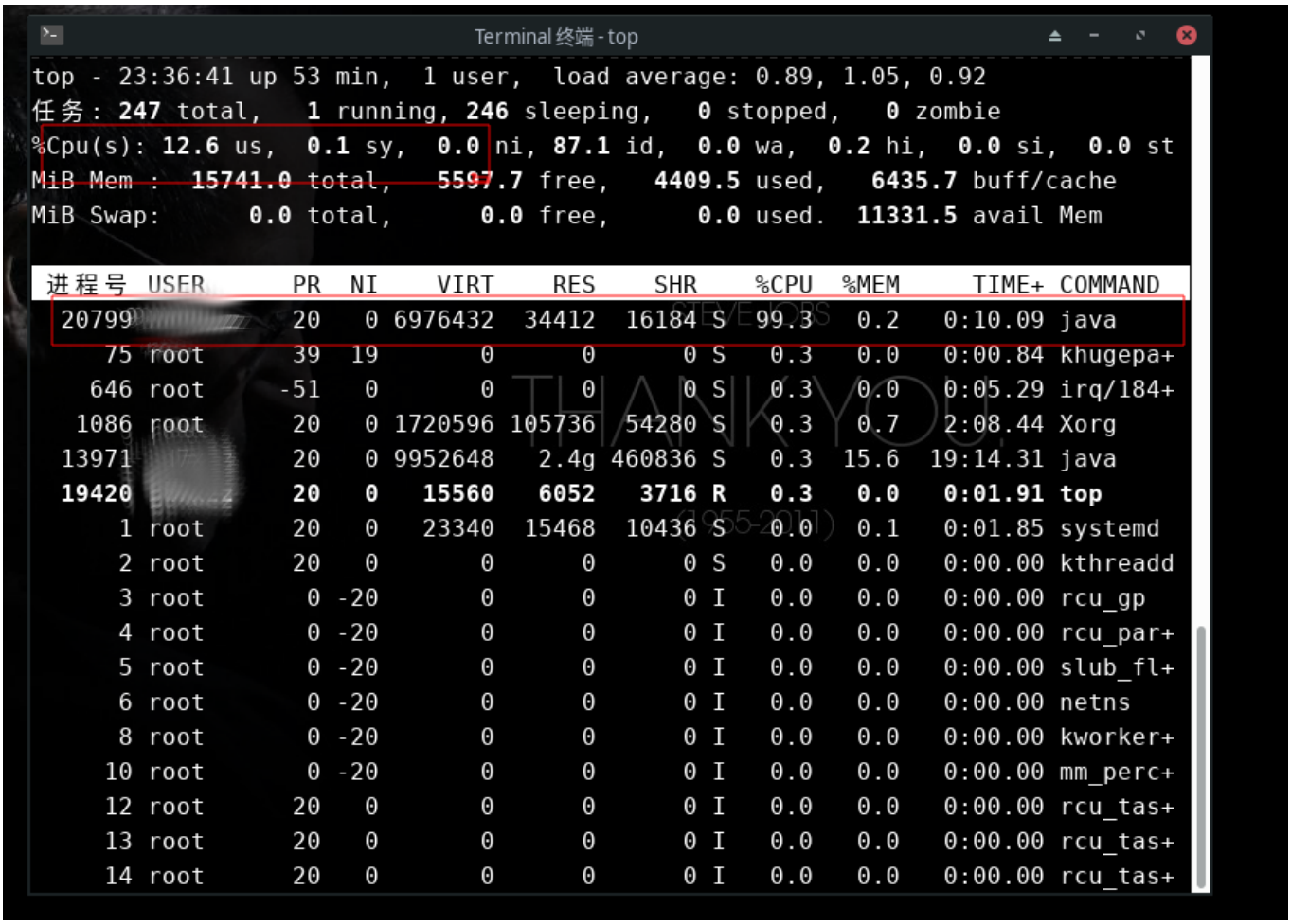


运行代码后，CPU使用率 `us` 和 `sy` 都有升高，同时进程列表中的java进程CPU使用一列也大幅度升高，

注意：俩个java进程，一个是IDEA编译器的，一个是通过IDEA启动的测试代码，由于死循环疯狂在IDEA控制台输出系统时间戳，导致IDEA的CPU使用率也升高了。



如果把死循环输出的那一行注释掉，可以看到只有 `us` 用户态的使用率升高了，同时IDEA的进程使用率不会升高。原因是 `System.out.println()` 和 `System.currentTimeMillis()` 都会产生系统调用进入内核态，注释掉死循环代码后，while死循环只是用户态的程序代码，不需要进入内核态了。

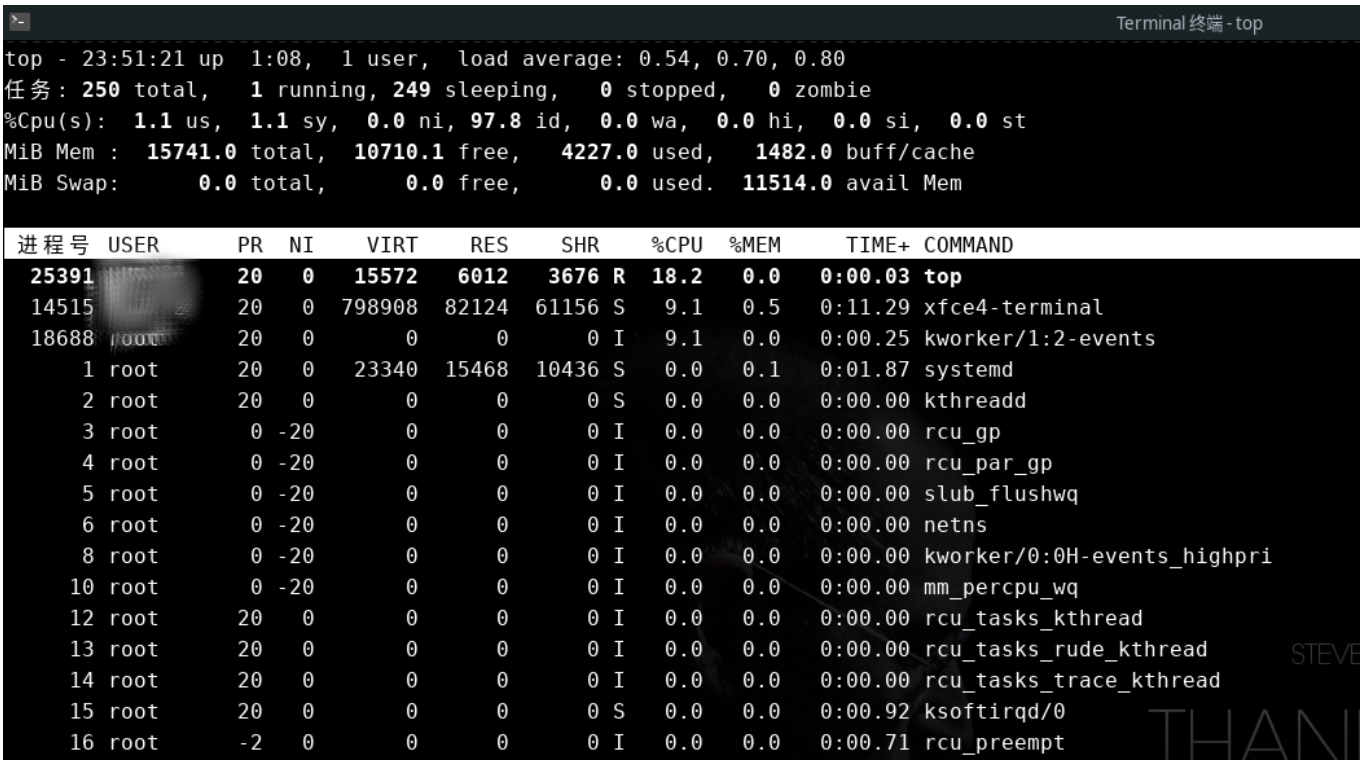


编写死锁代码：

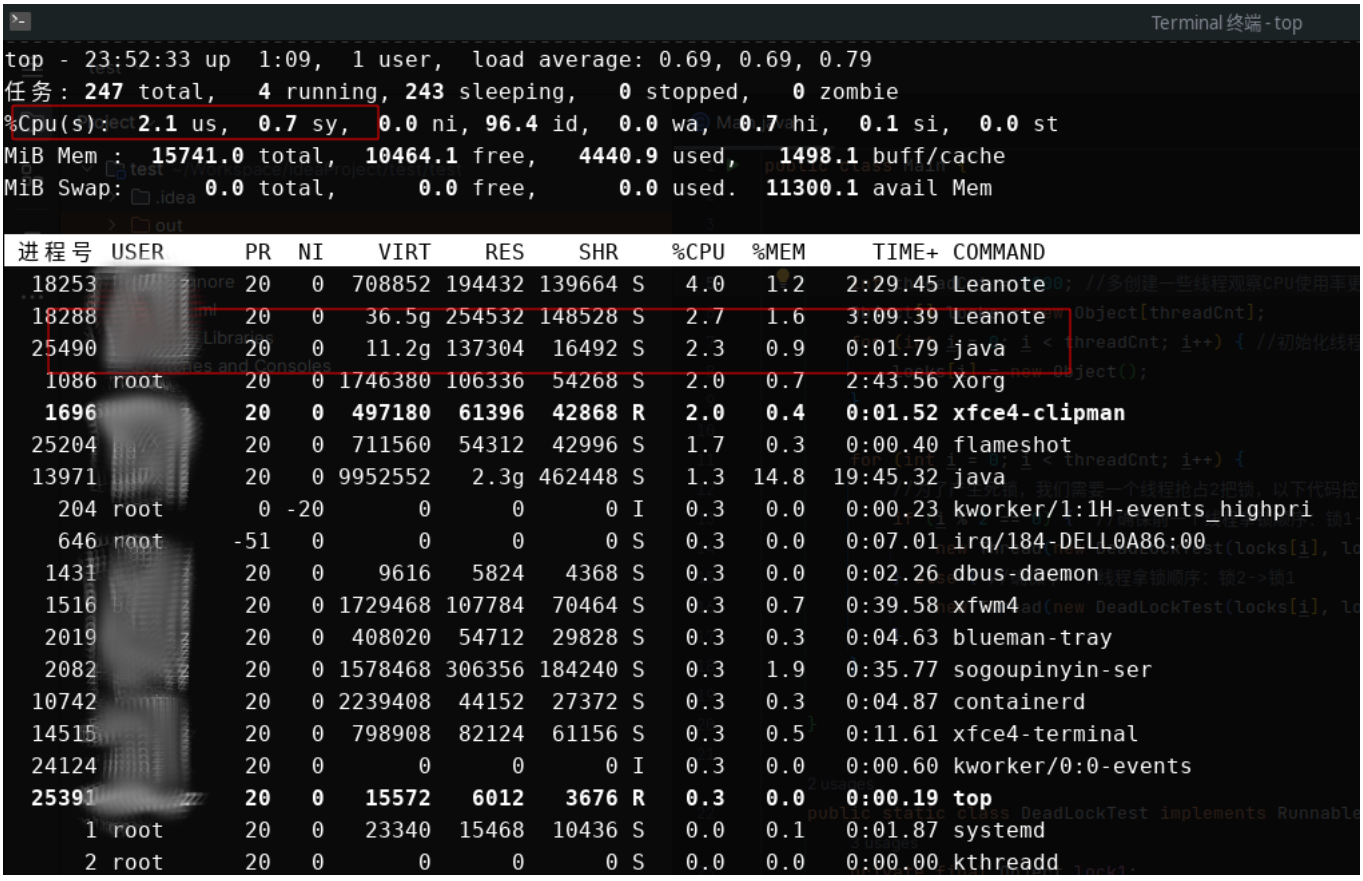
```
public class Main {
    public static void main(String[] args) {
        int threadCnt = 2000; //多创建一些线程观察CPU使用率更合理
        Object[] locks = new Object[threadCnt];
        for (int i = 0; i < threadCnt; i++) { //初始化线程锁对象
            locks[i] = new Object();
        }

        for (int i = 0; i < threadCnt; i++) {
            //为了产生死锁，我们需要一个线程抢占2把锁，以下代码控制创建时相邻2个线程分别拿到相同的锁
            if (i % 2 == 0) {  //确保前一个线程拿锁顺序：锁1->锁2
                new Thread(new DeadLockTest(locks[i], locks[i + 1])).start();
            } else { //确保下一个线程拿锁顺序：锁2->锁1
                new Thread(new DeadLockTest(locks[i], locks[i - 1])).start();
```

```
            }
        }

    }

    public static class DeadLockTest implements Runnable {
        private final Object lock1;
        private final Object lock2;

        public DeadLockTest(Object lock1, Object lock2) {
            this.lock1 = lock1;
            this.lock2 = lock2;
        }

        @Override
        public void run() {
            synchronized (lock1) {
                System.out.println(Thread.currentThread().getName() + " get
Lock:" + lock1); //打印线程名抢到的锁
                try {
                    Thread.sleep(1);//休眠1毫秒保证别的线程有机会拿到下面依赖的锁
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
                System.out.println(Thread.currentThread().getName() + " wait
Lock:" + lock2);//打印线程名要拿的锁
                synchronized (lock2) {//以下会由于拿锁顺序不正确，产生死锁
                    System.out.println(Thread.currentThread().getName() + "
get Lock:" + lock2);//这里由于死锁不会打印出来
                }
            }
        }
    }
}
```

为了验证死锁线程会不会导致CPU使用率升高，我们需要弄许多线程死锁，这样才能模拟出线上环境因为业务代码错误导致的死锁，来观察对CPU的影响。
运行前的CPU使用率



运行后的CPU使用率



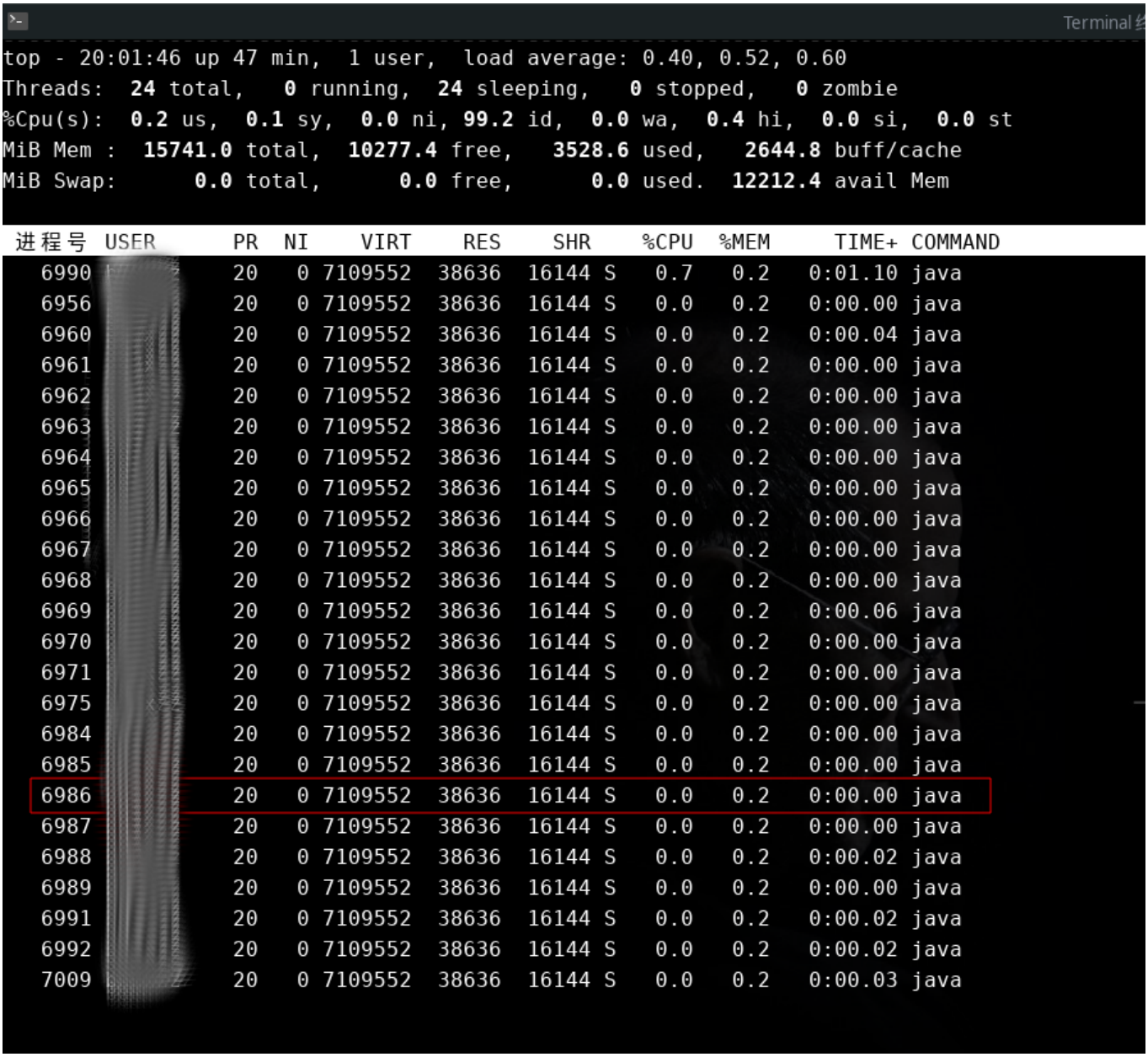
从图中可以看到CPU使用率并没有明显升高，java进程占用的CPU非常低。

`jstack -l` 也打印出了大量的死锁。

```
Found one Java-level deadlock:
=============================
"Thread-3":
  waiting to lock monitor 0x00007ff828003b98 (object 0x00000007766a3aa8, a java.lang.Object),
  which is held by "Thread-2"
"Thread-2":
  waiting to lock monitor 0x00007ff828005198 (object 0x000000077669f140, a java.lang.Object),
  which is held by "Thread-3"

Java stack information for the threads listed above:
===================================================
"Thread-3":
        at Main$DeadLockTest.run(Main.java:42)
        - waiting to lock <0x00000007766a3aa8> (a java.lang.Object)
        - locked <0x000000077669f140> (a java.lang.Object)
        at java.lang.Thread.run(Thread.java:748)
"Thread-2":
        at Main$DeadLockTest.run(Main.java:42)
        - waiting to lock <0x000000077669f140> (a java.lang.Object)
        - locked <0x00000007766a3aa8> (a java.lang.Object)
        at java.lang.Thread.run(Thread.java:748)

Found one Java-level deadlock:
=============================
"Thread-1":
  waiting to lock monitor 0x00007ff85c0066e8 (object 0x00000007766a87f8, a java.lang.Object),
  which is held by "Thread-0"
"Thread-0":
  waiting to lock monitor 0x00007ff828002598 (object 0x00000007766a8808, a java.lang.Object),
  which is held by "Thread-1"

Java stack information for the threads listed above:
===================================================
"Thread-1":
        at Main$DeadLockTest.run(Main.java:42)
        - waiting to lock <0x00000007766a87f8> (a java.lang.Object)
        - locked <0x00000007766a8808> (a java.lang.Object)
        at java.lang.Thread.run(Thread.java:748)
"Thread-0":
        at Main$DeadLockTest.run(Main.java:42)
        - waiting to lock <0x00000007766a8808> (a java.lang.Object)
        - locked <0x00000007766a87f8> (a java.lang.Object)
        at java.lang.Thread.run(Thread.java:748)

Found 999 deadlocks.
```

选一个Thread1查看一下线程状态：

```
jps -l |grep -E '[0-9].Main' |awk '{print $1}' #拿到测试代码Java进程pid

jstack -l <java进程pid>|grep Thread1| awk -v FS='nid=| ' '{print $9}' |xargs printf '%d\n' #提取死锁线程Thread1的线程pid

top -Hp <java进程pid> #查看线程状态
```



第一列为线程的pid，可以看到状态列为`S`，代表线程处在`sleeping`状态，因为拿不到锁让出CPU的使用权。

CPU使率用是统计online-cpu的任务，即任务状态为`running`的任务。所以印证了死锁不会导致CPU使用率升高的结论。之所以线上出现死锁使用率降低，是因为死锁后业务代码无法继续运行，导致使用率会降低。