# 深入理解Thread.sleep()的意义

2017-09-19 16:26:28 [椰子是只猫](https://me.csdn.net/agony_sun" \t "https://blog.csdn.net/agony_sun/article/details/_blank) 阅读数 3889更多

分类专栏： [转载技术贴](https://blog.csdn.net/agony_sun/article/category/6517917" \t "https://blog.csdn.net/agony_sun/article/details/_blank)

我们可能经常会用到 Thread.Sleep 函数来使线程挂起一段时间。那么你有没有正确的理解这个函数的用法呢？思考下面这两个问题：

假设现在是 2008-4-7 12:00:00.000，如果我调用一下 Thread.Sleep(1000) ，在 2008-4-7 12:00:01.000 的时候，这个线程会 不会被唤醒？   
某人的代码中用了一句看似莫明其妙的话：Thread.Sleep(0) 。既然是 Sleep 0 毫秒，那么他跟去掉这句代码相比，有啥区别么？   
我们先回顾一下操作系统原理。

操作系统中，CPU竞争有很多种策略。Unix系统使用的是时间片算法，而Windows则属于抢占式的。

在时间片算法中，所有的进程排成一个队列。操作系统按照他们的顺序，给每个进程分配一段时间，即该进程允许运行的时间。如果在 时间片结束时进程还在运行，则CPU将被剥夺并分配给另一个进程。如果进程在时间片结束前阻塞或结束，则CPU当即进行切换。调度程 序所要做的就是维护一张就绪进程列表，，当进程用完它的时间片后，它被移到队列的末尾。

所谓抢占式操作系统，就是说如果一个进程得到了 CPU 时间，除非它自己放弃使用 CPU ，否则将完全霸占 CPU 。因此可以看出，在抢 占式操作系统中，操作系统假设所有的进程都是“人品很好”的，会主动退出 CPU 。

在抢占式操作系统中，假设有若干进程，操作系统会根据他们的优先级、饥饿时间（已经多长时间没有使用过 CPU 了），给他们算出一 个总的优先级来。操作系统就会把 CPU 交给总优先级最高的这个进程。当进程执行完毕或者自己主动挂起后，操作系统就会重新计算一 次所有进程的总优先级，然后再挑一个优先级最高的把 CPU 控制权交给他。

我们用分蛋糕的场景来描述这两种算法。假设有源源不断的蛋糕（源源不断的时间），一副刀叉（一个CPU），10个等待吃蛋糕的人（10 个进程）。

如果是 Unix操作系统来负责分蛋糕，那么他会这样定规矩：每个人上来吃 1 分钟，时间到了换下一个。最后一个人吃完了就再从头开始。于是，不管这10个人是不是优先级不同、饥饿程度不同、饭量不同，每个人上来的时候都可以吃 1 分钟。当然，如果有人本来不太饿，或者饭量小，吃了30秒钟之后就吃饱了，那么他可以跟操作系统说：我已经吃饱了（挂起）。于是操作系统就会让下一个人接着来。

如果是 Windows 操作系统来负责分蛋糕的，那么场面就很有意思了。他会这样定规矩：我会根据你们的优先级、饥饿程度去给你们每个人计算一个优先级。优先级最高的那个人，可以上来吃蛋糕——吃到你不想吃为止。等这个人吃完了，我再重新根据优先级、饥饿程度来计算每个人的优先级，然后再分给优先级最高的那个人。

这样看来，这个场面就有意思了——可能有些人是PPMM，因此具有高优先级，于是她就可以经常来吃蛋糕。可能另外一个人是个丑男，而去很ws，所以优先级特别低，于是好半天了才轮到他一次（因为随着时间的推移，他会越来越饥饿，因此算出来的总优先级就会越来越高，因此总有一天会轮到他的）。而且，如果一不小心让一个大胖子得到了刀叉，因为他饭量大，可能他会霸占着蛋糕连续吃很久很久，导致旁边的人在那里咽口水。。。   
而且，还可能会有这种情况出现：操作系统现在计算出来的结果，5号PPMM总优先级最高，而且高出别人一大截。因此就叫5号来吃蛋糕。5号吃了一小会儿，觉得没那么饿了，于是说“我不吃了”（挂起）。因此操作系统就会重新计算所有人的优先级。因为5号刚刚吃过，因此她的饥饿程度变小了，于是总优先级变小了；而其他人因为多等了一会儿，饥饿程度都变大了，所以总优先级也变大了。不过这时候仍然有可能5号的优先级比别的都高，只不过现在只比其他的高一点点——但她仍然是总优先级最高的啊。因此操作系统就会说：5号mm上来吃蛋糕……（5号mm心里郁闷，这不刚吃过嘛……人家要减肥……谁叫你长那么漂亮，获得了那么高的优先级）。

那么，Thread.Sleep 函数是干吗的呢？还用刚才的分蛋糕的场景来描述。上面的场景里面，5号MM在吃了一次蛋糕之后，觉得已经有8分饱了，她觉得在未来的半个小时之内都不想再来吃蛋糕了，那么她就会跟操作系统说：在未来的半个小时之内不要再叫我上来吃蛋糕了。这样，操作系统在随后的半个小时里面重新计算所有人总优先级的时候，就会忽略5号mm。Sleep函数就是干这事的，他告诉操作系统“在未来的多少毫秒内我不参与CPU竞争”。

看完了 Thread.Sleep 的作用，我们再来想想文章开头的两个问题。

对于第一个问题，答案是：不一定。因为你只是告诉操作系统：在未来的1000毫秒内我不想再参与到CPU竞争。那么1000毫秒过去之后，这时候也许另外一个线程正在使用CPU，那么这时候操作系统是不会重新分配CPU的，直到那个线程挂起或结束；况且，即使这个时候恰巧轮到操作系统进行CPU 分配，那么当前线程也不一定就是总优先级最高的那个，CPU还是可能被其他线程抢占去。

与此相似的，Thread有个Resume函数，是用来唤醒挂起的线程的。好像上面所说的一样，这个函数只是“告诉操作系统我从现在起开始参与CPU竞争了”，这个函数的调用并不能马上使得这个线程获得CPU控制权。

对于第二个问题，答案是：有，而且区别很明显。假设我们刚才的分蛋糕场景里面，有另外一个PPMM 7号，她的优先级也非常非常高（因为非常非常漂亮），所以操作系统总是会叫道她来吃蛋糕。而且，7号也非常喜欢吃蛋糕，而且饭量也很大。不过，7号人品很好，她很善良，她没吃几口就会想：如果现在有别人比我更需要吃蛋糕，那么我就让给他。因此，她可以每吃几口就跟操作系统说：我们来重新计算一下所有人的总优先级吧。不过，操作系统不接受这个建议——因为操作系统不提供这个接口。于是7号mm就换了个说法：“在未来的0毫秒之内不要再叫我上来吃蛋糕了”。这个指令操作系统是接受的，于是此时操作系统就会重新计算大家的总优先级——注意这个时候是连7号一起计算的，因为“0毫秒已经过去了”嘛。因此如果没有比7号更需要吃蛋糕的人出现，那么下一次7号还是会被叫上来吃蛋糕。

因此，Thread.Sleep(0)的作用，就是“触发操作系统立刻重新进行一次CPU竞争”。竞争的结果也许是当前线程仍然获得CPU控制权，也许会换成别的线程获得CPU控制权。这也是我们在大循环里面经常会写一句Thread.Sleep(0) ，因为这样就给了其他线程比如Paint线程获得CPU控制权的权力，这样界面就不会假死在那里。

末了说明一下，虽然上面提到说“除非它自己放弃使用 CPU ，否则将完全霸占 CPU”，但这个行为仍然是受到制约的——操作系统会监控你霸占CPU的情况，如果发现某个线程长时间霸占CPU，会强制使这个线程挂起，因此在实际上不会出现“一个线程一直霸占着 CPU 不放”的情况。至于我们的大循环造成程序假死，并不是因为这个线程一直在霸占着CPU。实际上在这段时间操作系统已经进行过多次CPU竞争了，只不过其他线程在获得CPU控制权之后很短时间内马上就退出了，于是就又轮到了这个线程继续执行循环，于是就又用了很久才被操作系统强制挂起。。。因此反应到界面上，看起来就好像这个线程一直在霸占着CPU一样。

末了再说明一下，文中线程、进程有点混乱，其实在Windows原理层面，CPU竞争都是线程级的，本文中把这里的进程、线程看成同一个东西就好了。

原文链接：<http://www.cnblogs.com/javaee6/p/4754523.html>