

死锁和死循环会导致cpu升高吗

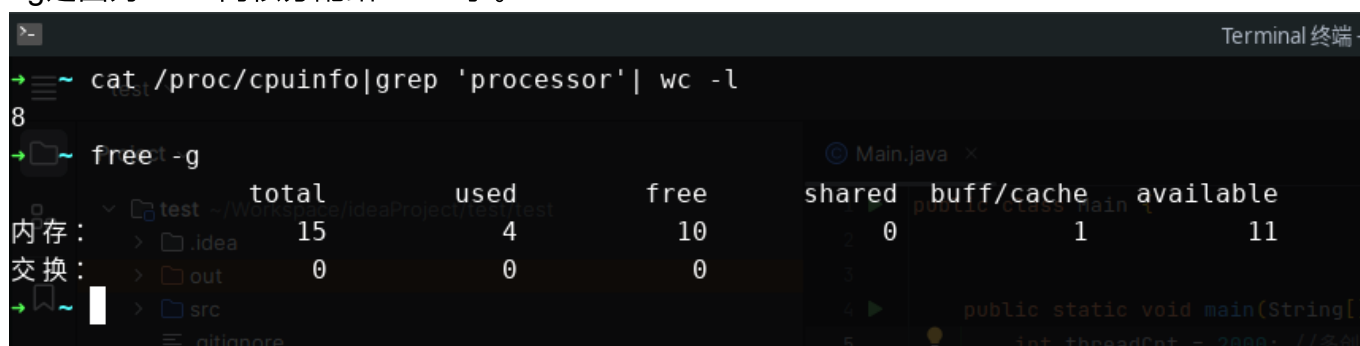
典型回答

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。



The terminal window shows the following commands and output:

```
cat /proc/cpuinfo | grep 'processor' | wc -l
8

free -g
total      used      free      shared  buff/cache   available
内存:    15g        4g        10g         0g         1g         11g
交换:     0g         0g         0g         0g         0g         0g
```

Below the terminal output, a snippet of Java code is shown:

```
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
        }
    }
}
```

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%。

```
Terminal 终端 - top
top - 23:26:22 up 43 min, 1 user, load average: 0.71, 0.68, 0.64
任务: 248 total, 1 running, 247 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.0 us, 0.5 sy, 0.0 ni, 97.9 id, 0.0 wa, 0.5 hi, 0.1 si, 0.0 st
MiB Mem : 15741.0 total, 5548.9 free, 4463.7 used, 6436.1 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 11277.3 avail Mem

 进程号  USER    PR   NI   VIRT   RES   SHR  %CPU  %MEM   TIME+  COMMAND
    1086  root      20    0 1717476 105604 54576 S    3.0   0.7   1:47.93 Xorg
    14515  root      20    0  794240  80020  61400 S    3.0   0.5   0:07.48 xfce4-t+
    13971  root      20    0 9942116 2.5g 460432 S    2.3  16.2  14:59.79 java
     646  root     -51    0      0      0      0 S    0.7   0.0   0:03.74 irq/184+
     885  root      20    0      0      0      0 I    0.3   0.0   0:00.64 kworker+
    1516  root      20    0 1729468 107580 70260 S    0.3   0.7   0:24.29 xfwm4
    1568  root      20    0  654236  70620  52940 S    0.3   0.4   0:04.31 xfce4-p+
    2049  root      20    0   8788   3728   3332 S    0.3   0.0   0:00.86 dbus-da+
    2082  root      20    0 1577172 303452 182036 S    0.3   1.9   0:18.76 sogoupi+
    19546  root      30   10 4302464 118024 27824 S    0.3   0.7   0:02.02 java
      1  root      20    0   23340  15468  10436 S    0.0   0.1   0:01.84 systemd
      2  root      20    0      0      0      0 S    0.0   0.0   0:00.00 kthreadd
      3  root       0  -20      0      0      0 I    0.0   0.0   0:00.00 rcu_gp
      4  root       0  -20      0      0      0 I    0.0   0.0   0:00.00 rcu_par+
      5  root       0  -20      0      0      0 I    0.0   0.0   0:00.00 slub_fl+
      6  root       0  -20      0      0      0 I    0.0   0.0   0:00.00 netns
      8  root       0  -20      0      0      0 I    0.0   0.0   0:00.00 kworker+
```

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。

```
Terminal 终端 - top
top - 23:29:27 up 46 min, 1 user, load average: 2.63, 1.18, 0.81
任务: 248 total, 2 running, 246 sleeping, 0 stopped, 0 zombie
%Cpu(s): 33.6 us, 11.1 sy, 0.0 ni, 54.6 id, 0.0 wa, 0.5 hi, 0.1 si, 0.0 st
MiB Mem : 15741.0 total, 5505.3 free, 4505.2 used, 6444.7 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 11235.8 avail Mem

 进程号  USER      PR    NI   VIRT    RES    SHR    %CPU  %MEM    TIME+  COMMAND
13971   java      20     0 9944388    2.4g  460696 S 250.2  15.6  17:27.13 java
20136   java      20     0 6976432 104804   16708 S  99.0   0.7   0:05.28 java
 1086   root      20     0 1711156 106024   54576 S   2.7   0.7   1:53.65 Xorg
18253   Leanote   20     0  713512 182816  138352 S   1.3   1.1   0:52.47 Leanote
18288   Leanote   20     0   36.5g 211964 134324 S   1.0   1.3   0:48.16 Leanote
 1568   xfce4-p+  20     0  654236  70620  52940 S   0.7   0.4   0:04.66 xfce4-p+
  355   root      20     0         0         0   0 I   0.3   0.0   0:00.20 kworker+
  646   root     -51     0         0         0   0 S   0.3   0.0   0:04.48 irq/184+
 1516   xfwm4     20     0 1729468 107580  70260 S   0.3   0.7   0:25.75 xfwm4
 2019   blueman+  20     0  408020  54712  29828 S   0.3   0.3   0:03.27 blueman+
 2082   sogoupi+  20     0 1577176 303876 182448 S   0.3   1.9   0:20.41 sogoupi+
10742   contain+  20     0 2239408  44152  27372 S   0.3   0.3   0:03.20 contain+
14515   xfce4-t+  20     0  794240  80020  61400 S   0.3   0.5   0:07.94 xfce4-t+
18219   Leanote   20     0 4816808 158404 121580 R   0.3   1.0   0:06.15 Leanote
19420   top       20     0   15560   6052   3716 R   0.3   0.0   0:00.80 top
    1   root      20     0   23340  15468  10436 S   0.0   0.1   0:01.85 systemd
    2   root      20     0         0         0   0 S   0.0   0.0   0:00.00 kthreadd
```

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。

```
Terminal 终端 - top
top - 23:36:41 up 53 min, 1 user, load average: 0.89, 1.05, 0.92
任务: 247 total, 1 running, 246 sleeping, 0 stopped, 0 zombie
%Cpu(s): 12.6 us, 0.1 sy, 0.0 ni, 87.1 id, 0.0 wa, 0.2 hi, 0.0 si, 0.0 st
MiB Mem : 15741.0 total, 5597.7 free, 4409.5 used, 6435.7 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 11331.5 avail Mem

 进 程 号  USER  PR  NI  VIRT  RES  SHR  %CPU  %MEM  TIME+  COMMAND
20799  java    20   0 6976432 34412 16184 S  99.3  0.2  0:10.09 java
75  root    39  19   0      0   0 S  0.3  0.0  0:00.84 khugepa+
646  root   -51   0   0      0   0 S  0.3  0.0  0:05.29 irq/184+
1086  root    20   0 1720596 105736 54280 S  0.3  0.7  2:08.44 Xorg
13971  java    20   0 9952648 2.4g 460836 S  0.3 15.6 19:14.31 java
19420  top     20   0  15560  6052  3716 R  0.3  0.0  0:01.91 top
1  root    20   0  23340 15468 10436 S  0.0  0.1  0:01.85 systemd
2  root    20   0   0      0   0 S  0.0  0.0  0:00.00 kthreadd
3  root     0 -20   0      0   0 I  0.0  0.0  0:00.00 rcu_gp
4  root     0 -20   0      0   0 I  0.0  0.0  0:00.00 rcu_par+
5  root     0 -20   0      0   0 I  0.0  0.0  0:00.00 slub_fl+
6  root     0 -20   0      0   0 I  0.0  0.0  0:00.00 netns
8  root     0 -20   0      0   0 I  0.0  0.0  0:00.00 kworker+
10  root     0 -20   0      0   0 I  0.0  0.0  0:00.00 mm_perc+
12  root    20   0   0      0   0 I  0.0  0.0  0:00.00 rcu_tas+
13  root    20   0   0      0   0 I  0.0  0.0  0:00.00 rcu_tas+
14  root    20   0   0      0   0 I  0.0  0.0  0:00.00 rcu_tas+
```

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

```
Terminal 终端 - top
top - 23:51:21 up 1:08, 1 user, load average: 0.54, 0.70, 0.80
任务: 250 total, 1 running, 249 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.1 us, 1.1 sy, 0.0 ni, 97.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 15741.0 total, 10710.1 free, 4227.0 used, 1482.0 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 11514.0 avail Mem

 进程号  USER    PR    NI   VIRT   RES   SHR  %CPU  %MEM   TIME+  COMMAND
25391      root      20     0   15572   6012   3676 R   18.2   0.0   0:00.03 top
14515      root      20     0  798908  82124  61156 S    9.1   0.5   0:11.29 xfce4-terminal
18688      root      20     0      0      0      0 I    9.1   0.0   0:00.25 kworker/1:2-events
 1      root      20     0   23340  15468  10436 S    0.0   0.1   0:01.87 systemd
 2      root      20     0      0      0      0 S    0.0   0.0   0:00.00 kthreadd
 3      root       0  -20     0      0      0 I    0.0   0.0   0:00.00 rcu_gp
 4      root       0  -20     0      0      0 I    0.0   0.0   0:00.00 rcu_par_gp
 5      root       0  -20     0      0      0 I    0.0   0.0   0:00.00 slub_flushwq
 6      root       0  -20     0      0      0 I    0.0   0.0   0:00.00 netns
 8      root       0  -20     0      0      0 I    0.0   0.0   0:00.00 kworker/0:0H-events_highpri
10      root       0  -20     0      0      0 I    0.0   0.0   0:00.00 mm_percpu_wq
12      root      20     0      0      0      0 I    0.0   0.0   0:00.00 rcu_tasks_kthread
13      root      20     0      0      0      0 I    0.0   0.0   0:00.00 rcu_tasks_rude_kthread
14      root      20     0      0      0      0 I    0.0   0.0   0:00.00 rcu_tasks_trace_kthread
15      root      20     0      0      0      0 S    0.0   0.0   0:00.92 ksoftirqd/0
16      root      -2     0      0      0      0 I    0.0   0.0   0:00.71 rcu_preempt
```

运行后的CPU使用率

```
Terminal 终端 - top
top - 23:52:33 up 1:09, 1 user, load average: 0.69, 0.69, 0.79
任务: 247 total, 4 running, 243 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.1 us, 0.7 sy, 0.0 ni, 96.4 id, 0.0 wa, 0.7 hi, 0.1 si, 0.0 st
MiB Mem : 15741.0 total, 10464.1 free, 4440.9 used, 1498.1 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 11300.1 avail Mem

 进程号  USER    PR    NI   VIRT   RES   SHR  %CPU  %MEM   TIME+  COMMAND
18253      root      20     0  708852  194432  139664 S    4.0   1.2   2:29.45 Leanote
18288      root      20     0   36.5g  254532  148528 S    2.7   1.6   3:09.39 Leanote
25490      root      20     0   11.2g  137304  16492 S    2.3   0.9   0:01.79 java
1086      root      20     0  1746380  106336  54268 S    2.0   0.7   2:43.56 Xorg
1696      root      20     0  497180  61396  42868 R    2.0   0.4   0:01.52 xfce4-clipman
25204      root      20     0   711560  54312  42996 S    1.7   0.3   0:00.40 flameshot
13971      root      20     0  9952552  2.3g  462448 S    1.3  14.8  19:45.32 java
 204      root       0  -20      0      0      0 I    0.3   0.0   0:00.23 kworker/1:1H-events_highpri
 646      root     -51     0      0      0      0 S    0.3   0.0   0:07.01 irq/184-DELL0A86:00
1431      root      20     0   9616   5824   4368 S    0.3   0.0   0:02.26 dbus-daemon
1516      root      20     0  1729468  107784  70464 S    0.3   0.7   0:39.58 xfwm4
2019      root      20     0   408020  54712  29828 S    0.3   0.3   0:04.63 blueman-tray
2082      root      20     0  1578468  306356  184240 S    0.3   1.9   0:35.77 sogoupinyin-ser
10742      root      20     0  2239408  44152  27372 S    0.3   0.3   0:04.87 containerd
14515      root      20     0  798908  82124  61156 S    0.3   0.5   0:11.61 xfce4-terminal
24124      root      20     0      0      0      0 I    0.3   0.0   0:00.60 kworker/0:0-events
25391      root      20     0   15572   6012   3676 R    0.3   0.0   0:00.19 top
 1      root      20     0   23340  15468  10436 S    0.0   0.1   0:01.87 systemd
 2      root      20     0      0      0      0 S    0.0   0.0   0:00.00 kthreadd
```

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
```

Terminal

top - 20:01:46 up 47 min, 1 user, load average: 0.40, 0.52, 0.60
Threads: 24 total, 0 running, 24 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.2 us, 0.1 sy, 0.0 ni, 99.2 id, 0.0 wa, 0.4 hi, 0.0 si, 0.0 st
MiB Mem : 15741.0 total, 10277.4 free, 3528.6 used, 2644.8 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 12212.4 avail Mem

进程号	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+	COMMAND
6990		20	0	7109552	38636	16144 S	0.7	0.2	0:01.10	java
6956		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6960		20	0	7109552	38636	16144 S	0.0	0.2	0:00.04	java
6961		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6962		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6963		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6964		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6965		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6966		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6967		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6968		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6969		20	0	7109552	38636	16144 S	0.0	0.2	0:00.06	java
6970		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6971		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6975		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6984		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6985		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6986		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6987		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6988		20	0	7109552	38636	16144 S	0.0	0.2	0:00.02	java
6989		20	0	7109552	38636	16144 S	0.0	0.2	0:00.00	java
6991		20	0	7109552	38636	16144 S	0.0	0.2	0:00.02	java
6992		20	0	7109552	38636	16144 S	0.0	0.2	0:00.02	java
7009		20	0	7109552	38636	16144 S	0.0	0.2	0:00.03	java

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 使用率是统计 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。