### **http://www.361way.com/linux-context-switch/5131.html**

### **<https://www.linuxidc.com/Linux/2017-08/146284.htm>**

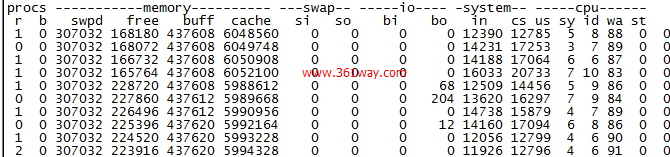
https://www.google.fr/search?q=linux%E5%A6%82%E4%BD%95%E6%9F%A5%E7%9C%8B%E5%93%AA%E4%B8%AA%E8%BF%9B%E7%A8%8B%E4%B8%8A%E4%B8%8B%E6%96%87%E5%88%87%E6%8D%A2%E8%BE%83%E9%A2%91%E7%B9%81&oq=linux%E5%A6%82%E4%BD%95%E6%9F%A5%E7%9C%8B%E5%93%AA%E4%B8%AA%E8%BF%9B%E7%A8%8B%E4%B8%8A%E4%B8%8B%E6%96%87%E5%88%87%E6%8D%A2%E8%BE%83%E9%A2%91%E7%B9%81&aqs=chrome..69i57.32635j0j7&sourceid=chrome&ie=UTF-8

### **一、问题现象**

现网有两台虚拟机主机95%的cpu处于idle状态，内存使用率也不是特别高，而主机的load average达到了40多。

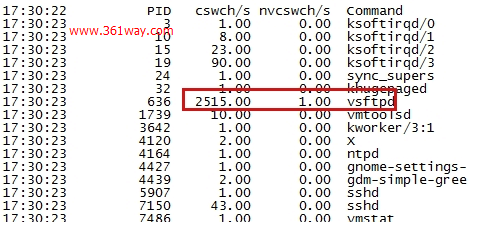
### **二、问题分析**

先在主机上通过top、free、ps、iostat 等常用工具分析了下主机的CPU、内存、IO使用情况，发现三者都不高。通过vmstat 1 查看的结果如下：

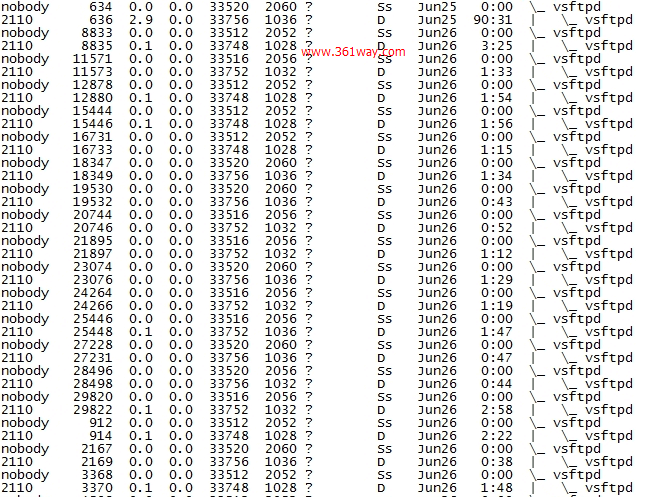


从vmstat的输出结果来看，io项的block in 和block out 并不频繁。而system项的每称的中断数（in)、每秒的上下文切换（cs）特别频繁。这就造成load avaerage会特别高。大方向上的根因找到了，具体是哪个进程如何频繁的进行中断和上下文件的切换呢？

这里使用pidstat -w 1 （每秒刷新输出上下文切换情况），输出见下图：



从上图中可以看到有cswch(自愿的上下文切换)和nvcswch(非自愿的上下文切换)及对应的命令， 出vsftpd占用的文件交换比较多。可以看到这里显示的cs 值和总值还是有比较大的差距，由于主机上启动了不止一个vsftpd进程，而且pidstat 通过1秒刷新的时候并不会显示所有，通过pidstat -w执行几次收集所有发现所有的vsftpd进程占用的cs值叠加和vmstat里的比较相近了。



将结果通知业务人员后，和业务人员的猜测也一致，由于ftp使用的目录结构层次较深、文件数也比较多，业务在备份老的使用目录并重新创建单层目录后，观察一段后，发现load average降下来了，稳定在1以下。

当然这里只是处理方法的一种，现网中有些程序不好进行这样的修改的，又不让让进程在cpu之间频繁切换的，也有通过设置固定运行的CPU上进行调优的方法，如下两个进程运行在0－7号cpu上：

1. [root@www ~]# taskset -c -p 6389
2. pid 6389's current affinity list: 0-7
3. [root@www ~]# taskset -c -p 6580
4. pid 6580's current affinity list: 0-7

可以通过taskset让其固定在0－1号cpu上运行：

1. [root@www ~]# taskset -c 0,1 -p 6389

这样做的原理是每当进程在切换到下一个cpu core上进会flush当前的cache数据，指定CPU时会减少这样的操作，增加进程的处理速度。这个对老的程序调优时比较有效。

### **三、有关上下文切换**

#### **1、上下文切换的理解**

什么是上下文件切换呢？引用老外的一句话：A context switch (also sometimes referred to as a process switch or a task switch) is the switching of the CPU (central processing unit) from one process or thread to another.更详细的说明可以参看[linfo站点](http://www.linfo.org/context_switch.html" \t "http://www.361way.com/linux-context-switch/_blank) 或 [维基百科](https://en.wikipedia.org/wiki/Context_switch" \t "http://www.361way.com/linux-context-switch/_blank) 。

context switch过高会导致CPU像个搬运工，频繁在寄存器和运行队列之间奔波 ，更多的时间花在了线程切换，而不是真正工作的线程上。直接的消耗包括CPU寄存器需要保存和加载，系统调度器的代码需要执行。间接消耗在于多核cache之间的共享数据。

#### **2、引起上下文切换的原因**

对于抢占式操作系统而言， 大体有几种：

* 当前任务的时间片用完之后，系统CPU正常调度下一个任务；
* 当前任务碰到IO阻塞，调度线程将挂起此任务，继续下一个任务；
* 多个任务抢占锁资源，当前任务没有抢到，被调度器挂起，继续下一个任务；
* 用户代码挂起当前任务，让出CPU时间；
* 硬件中断；

什么样的操作会引起CS，[这里有一篇博文](http://www.cnblogs.com/zhiranok/archive/2012/08/13/context_switch_1.html" \t "http://www.361way.com/linux-context-switch/_blank)感觉写的很不错，虽然其中的代码部分并不是理解 。其中有如下几句话：

linux中一个进程的时间片到期，或是有更高优先级的进程抢占时，是会发生CS的，但这些都是我们应用开发者不可控的 －－－前面一部分描述的很到位，后面一部分在系统层面和kernel 开发层面可以调用nice 或 renice进行设置优先级以保证某些程序优先在CPU中的占用时间，但也不能细化到CS层面。

站在开发者的角度，我们的进程可以主动地向内核申请进行CS 。操作方法为：休眠当前进程/线程；唤醒其他进程/线程 。

#### **3、上下文切换测试工具**

1、[LMbench](http://www.bitmover.com/lmbench/" \t "http://www.361way.com/linux-context-switch/_blank) 是带宽（读取缓存文件、内存拷贝、读写内存、管道等）和反应时间（上下文切换、网路、进程创建等）的评测工具；

2、[micro-benchmark contextswitch](https://github.com/361way/contextswitch" \t "http://www.361way.com/linux-context-switch/_blank) 可以测试不同的CPU在最少多少ns可以进行一次上下文件切换，再转化为秒，我们可以确认该处理器每可以进行的上下文件切换数 ，该工具的使用可以参看[tsuna的blog](http://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html" \t "http://www.361way.com/linux-context-switch/_blank)。

### **4、上下文切换的查看方法**

**sar -w** ，这个只是能看出主机上总的上下文件切换的情况

1. # sar -w 1
2. proc/s
3. **Total** number of tasks created per second.
4. cswch/s
5. **Total** number of context switches per second.

同样，**vmstat**也可以查看总的上下文切换情况，不过vmstart输出的结果更多，便比通过对比发现问题：

1. # vmstat 3
2. procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
3. r b swpd free buff cache si so bi bo **in** cs us sy id wa
4. 2 0 7292 249472 82340 2291972 0 0 0 0 0 0 7 13 79 0
5. 0 0 7292 251808 82344 2291968 0 0 0 184 24 20090 1 1 99 0
6. 0 0 7292 251876 82344 2291968 0 0 0 83 17 20157 1 0 99 0
7. 0 0 7292 251876 82344 2291968 0 0 0 73 12 20116 1 0 99 0

查看每个进程或线程的上下文件使用情况，可以使用pidstat命令或者通过查看proc 。

1. # pidstat -w 每个进程的context switching情况
2. # pidstat -wt 细分到每个threads
3. 查看proc下的文件方法如下：
4. # pid=307
5. # grep ctxt /proc/$pid/status
6. voluntary\_ctxt\_switches: 41 #自愿的上下文切换
7. nonvoluntary\_ctxt\_switches: 16 #非自愿的上下文切换

cswch/s: 每秒任务主动(自愿的)切换上下文的次数，当某一任务处于阻塞等待时，将主动让出自己的CPU资源。

nvcswch/s: 每秒任务被动(不自愿的)切换上下文的次数，CPU分配给某一任务的时间片已经用完，因此将强迫该进程让出CPU的执行权。

上下文切换部分零零碎碎先到这里吧，只是想说明上下文切换还是比较重要的一个指标的。nagios check\_mk默认有对上下文的监控，其使用的方法是通过两/proc/stat文件里取到ctxt行，并取两个时间段之间的差值来确认。

1. # cat /proc/stat|grep ctxt
2. ctxt 111751207