**1.  概念介绍**

**1.1  Linux系统进程状态**

在linux中,process有以下状态: runnable （就绪状态）；blocked waiting for an event to complete（阻塞等待一个时间完成，此状态的process可能在等待一个I/O操作获取的数据，或者是一个系统调用的结果等）；running（正在执行）。

如果一个process在runnable状态，即它和其他同样处在runnable状态的process在等待CPU时间，而不是立即获得CPU时间，即“就绪”状态，表示随时可以执行但是没有在执行，这种状态是不消耗CPU时间的。他们形成的等待队列称作Run Queue，Run Queue越大，表示等待的队列越长。Linux调度进程，从runnable队列中（Run Queue），选择一个process下次执行，那么这个process就会获得CPU时间，变成running状态。

process的runnable和running状态，在linux系统中用TASK\_RUNNING这个全局变量来表示。TASK\_RUNNING表示process正在被CPU执行，或者已经准备就绪随时可由调度程序调度执行；若此时进程没有被CPU执行，则称其处于就绪状态（runnable），若此时进程正在被CPU执行，则称其处于执行状态（running）。当一个进程在内核代码中运行时，我们称其处于内核态；当一个进程正在执行用户自己的代码时，我们称其处于用户态。当系统资源已经可用时，进程就被唤醒而进入准备运行状态，也就是就绪状态。这些状态在内核中表示方法相同，都被称为TASK\_RUNNING状态。当一个进程刚被创建后就处于TASK\_RUNNING状态。监控的load average，就是这个TASK\_RUNNING值，它指处在running和runnable的process的总和。例如：如果有2个processes在running和有3个runnable，那么系统的load为5。

**1.2  Load Average概述**

Load Average是CPU的Load，它所包含的信息不是CPU的使用率状况，而是在一段时间内CPU正在处理以及等待CPU处理的进程数之和的统计信息，也就是CPU使用队列的长度的统计信息。也有简单说是进程队列的长度，load average数据是每隔5秒钟检查一次活跃的进程数，然后按特定算法计算出的数值。

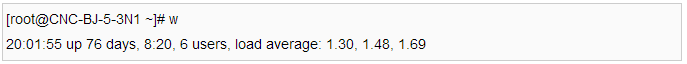
需要区别CPU负载和CPU利用率，两个不同的概念。CPU利用率显示的是程序在运行期间实时占用的CPU百分比，而CPU负载显示的是一段时间内正在使用和等待使用CPU的平均任务数。CPU利用率高，并不意味着负载就一定大。PS：某公用电话亭，有一个人在打电话，四个人在等待，每人限定使用电话一分钟，若有人一分钟之内没有打完电话，只能挂掉电话去排队，等待下一轮。电话在这里就相当于CPU，而正在或等待打电话的人就相当于任务数。在电话亭使用过程中，肯定会有人打完电话走掉，有人没有打完电话而选择重新排队，更会有新增的人在这儿排队，这个人数的变化就相当于任务数的增减。

为了统计平均负载情况，我们5秒钟统计一次人数，并在第1、5、15分钟的时候对统计情况取平均值，从而形成第1、5、15分钟的平均负载。有的人拿起电话就打，一直打完1分钟，而有的人可能前三十秒在找电话号码，或者在犹豫要不要打，后三十秒才真正在打电话。如果把电话看作CPU，人数看作任务，我们就说前一个人（任务）的CPU利用率高，后一个人（任务）的CPU利用率低。

当然， CPU并不会在前三十秒工作，后三十秒歇着，只是说，有的程序涉及到大量的计算，所以CPU利用率就高，而有的程序牵涉到计算的部分很少，CPU利用率自然就低。但无论CPU的利用率是高是低，跟后面有多少任务在排队没有必然关系。

**2.  如何查看当前load average情况**

**可以通过系统命令"w"查看**



**使用uptime 命令输出**

http://images.cnitblog.com/blog/348208/201411/261617504492272.png

**数值说明**

第一位1.30：表示最近1分钟平均负载

第二位1.48：表示最近5分钟平均负载

第三位1.69：表示最近15分钟平均负载

**3.  如何判断是否Over Load了**

理论上前15分钟的平均CPU负载越小越好

指标：< CPU个数 \* 核心数 \* 0.7

网上也有指标：< CPU个数 \* 核心数 \* 0.5

备注：Linux里有一个/proc目录，存放的是当前运行系统的虚拟映射，文件cpuinfo存放着CPU的信息。processor：逻辑CPU的标识，model name：真实CPU的型号信息，physical id：真实CPU和标识，cpu cores：真实CPU的内核数

使用命令grep 'model name' /proc/cpuinfo | wc –l查看CPU核数

**3.1  单核处理器**

假设我们的系统是单CPU单内核的，把它比喻成是一条单向桥，把CPU任务比作汽车。当车不多的时候，load <1；当车占满整个马路的时候 load=1；当马路都站满了，而且马路外还堆满了汽车的时候，load>1

http://images.cnitblog.com/blog/348208/201411/261618516378011.pngLoad < 1

http://images.cnitblog.com/blog/348208/201411/261619023245147.pngLoad = 1

http://images.cnitblog.com/blog/348208/201411/261619132623038.png Load >1

**3.2  多核处理器**

我们经常会发现服务器Load > 1但是运行仍然不错，那是因为服务器是多核处理器（Multi-core）。假设我们服务器CPU是2核，那么将意味我们拥有2条马路，我们的Load = 2时，所有马路都跑满车辆。

http://images.cnitblog.com/blog/348208/201411/261619319815553.pngLoad = 2时马路都跑满了

#查看CPU core

grep 'model name' /proc/cpuinfo | wc -l

**3.3  什么样的Load average值要提高警惕**

ü  0.7 < load < 1: 此时是不错的状态，如果进来更多的汽车，你的桥仍然可以应付。

ü  load = 1: 你的桥即将拥堵，而且没有更多的资源额外的任务，赶紧看看发生了什么吧。

ü  load > 5: 非常严重拥堵，我们的桥非常繁忙，每辆车都无法很快的运行

**3.4  三种Load值，应该看哪个？**

通常我们先看15分钟load，如果load很高，再看1分钟和5分钟负载，查看是否有下降趋势。1分钟负载值 > 1，那么我们不用担心，但是如果15分钟负载都超过1，我们要赶紧看看发生了什么事情。所以我们要根据实际情况查看这三个值。

**4.  Load Average在压力测试中的作用**

**4.1  CPU时间片**

为了提高程序执行效率，大家在很多应用中都采用了多线程模式，这样可以将原来的序列化执行变为并行执行，任务的分解以及并行执行能够极大地提高程序的运行效率。但这都是代码级别的表现，而硬件是如何支持的呢？那就要靠CPU的时间片模式来说明这一切。程序的任何指令的执行往往都会要竞争CPU这个最宝贵的资源，不论你的程序分成了多少个线程去执行不同的任务，他们都必须排队等待获取这个资源来计算和处理命令。先看看单CPU的情况。下面两图描述了非时间片模式和时间片模式下的线程执行的情况：

在图一中可以看到，任何线程如果都排队等待正在使用CPU的线程结束后，来获取CPU资源，那么所谓的多线程就没有任何实际意义。图二中的CPU Manager只是我虚拟的一个角色，由它来分配和管理CPU的使用状况，此时多线程将会在运行过程中都有机会得到CPU资源，也真正实现了在单CPU的情况下实现多线程并行处理。

多CPU的情况只是单CPU的扩展，当所有的CPU都满负荷运作的时候，就会对每一个CPU采用时间片的方式来提高效率。

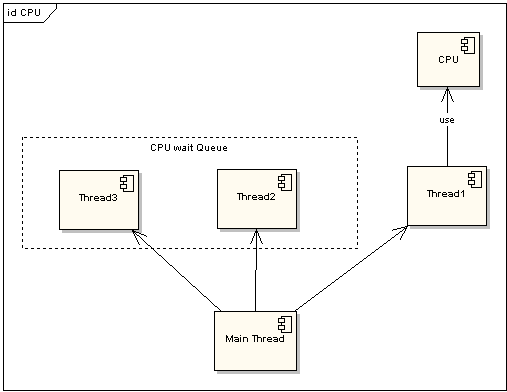


图1：非时间片模式线程执行情况

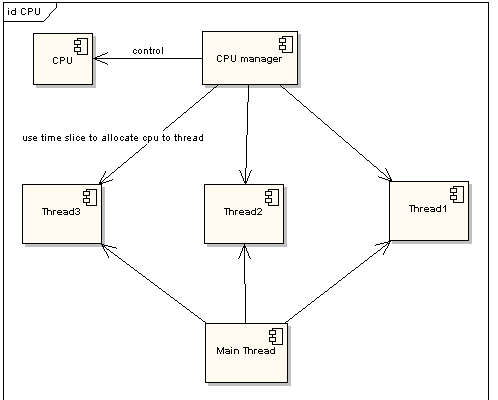


图2：时间片模式线程执行情况

 在Linux的内核处理过程中，每一个进程默认会有一个固定的时间片来执行命令（默认为1/100秒），这段时间内进程被分配到CPU，然后独占使用。如果使用完，同时未到时间片的规定时间，那么就主动放弃CPU的占用，如果到时间片尚未完成工作，那么CPU的使用权也会被收回，进程将会被中断挂起等待下一个时间片。

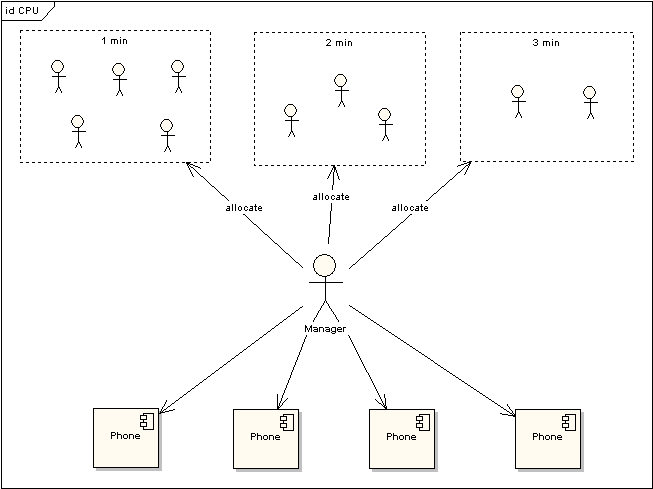
说明：时间片设定。从一个进程切换到另一个进程是需要一定时间的（保存和装入寄存器值及内存映像，更新各种表格和队列等），假如进程切换(process switch) - 有时称为上下文切换(context switch)，需要5毫秒，再假设时间片设为20毫秒，则在做完20毫秒有用的工作之后，CPU将花费5毫秒来进行进程切换。CPU时间的20%被浪费在了管理开销上。为了提高CPU效率，可以将时间片设为500毫秒。这时浪费的时间只有1%。但考虑在一个分时系统中，如果有十个交互用户几乎同时按下回车键，将发生什么情况？假设所有其他进程都用足它们的时间片的话，最后一个不幸的进程不得不等待5秒钟才获得运行机会。多数用户无法忍受一条简短命令要5秒钟才能做出响应。同样的问题在一台支持多道程序的个人计算机上也会发生。即时间片设得太短会导致过多的进程切换，降低了CPU效率；而设得太长又可能引起对短的交互请求的响应变差。通常设为100毫秒。

**4.2  CPU利用率和Load Average的区别**

CPU利用率，顾名思义就是对于CPU的使用状况，这是对一个时间段内CPU使用状况的统计，通过这个指标可以看出在某一个时间段内CPU被占用的情况，如果被占用时间很高，那么就需要考虑CPU是否已经处于超负荷运作，长期超负荷运作对于机器本身来说是一种损害，因此必须将CPU的利用率控制在一定的比例下，以保证机器的正常运作。

Load Average是CPU的Load，它所包含的信息不是CPU的使用率状况，而是在一段时间内CPU正在处理以及等待CPU处理的进程数之和的统计信息，也就是CPU使用队列的长度的统计信息。

我们将CPU类比为电话亭，每一个进程都是一个需要打电话的人。现在一共有4个电话亭（就好比我们的机器有4核），有10个人需要打电话。现在使用电话的规则是管理员会按照顺序给每一个人轮流分配1分钟的使用电话时间，如果使用者在1分钟内使用完毕，那么可以立刻将电话使用权返还给管理员，如果到了1分钟电话使用者还没有使用完毕，那么需要重新排队，等待再次分配使用。

  
图 3 电话使用场景

上图中对于使用电话的用户又作了一次分类，1min的代表这些使用者占用电话时间小于等于1min，2min表示使用者占用电话时间小于等于2min，以此类推。根据电话使用规则，1min的用户只需要得到一次分配即可完成通话，而其他两类用户需要排队两次到三次。

电话的利用率 = sum (active use cpu time)/period

每一个分配到电话的使用者使用电话时间的总和去除以统计的时间段。这里需要注意的是是使用电话的时间总和(sum(active use cpu time))，这与占用时间的总和(sum(occupy cpu time))是有区别的。（例如一个用户得到了一分钟的使用权，在10秒钟内打了电话，然后去查询号码本花了20秒钟，再用剩下的30秒打了另一个电话，那么占用了电话1分钟，实际只是使用了40秒）

电话的Average Load体现的是在某一统计时间段内，所有使用电话的人加上等待电话分配的人一个平均统计。

电话利用率的统计能够反映的是电话被使用的情况，当电话长期处于被使用而没有得到足够的时间休息，那么对于电话硬件来说是一种超负荷的运作，需要调整使用频度。而电话Average Load却从另一个角度来展现对于电话使用状态的描述，Average Load越高说明对于电话资源的竞争越激烈，电话资源比较短缺。对于资源的申请和维护其实也是需要很大的成本，所以在这种高Average Load的情况下电话资源的长期“热竞争”也是对于硬件的一种损害。

低利用率的情况下是否会有高Load Average的情况产生呢？理解占有时间和使用时间就可以知道，当分配时间片以后，是否使用完全取决于使用者，因此完全可能出现低利用率高Load Average的情况。由此来看，仅仅从CPU的使用率来判断CPU是否处于一种超负荷的工作状态还是不够的，必须结合Load Average来全局的看CPU的使用情况和申请情况。

**5.  常见误区**

**5.1  系统 Load 高一定是性能有问题**

真相：系统 Load 高也或许是因为在进行 CPU 密集型的计算(比如编译)

**5.2  系统 Load 高一定是 CPU 能力问题或数量不够**

真相：Load 高只是代表需要运行的队列累积过多了。但队列中的任务实际可能是耗 CPU的，也可能是耗 I/O 乃至其它因素的。

**5.3  系统长期 Load 高，首选增加 CPU**

真相：Load 只是表象，不是实质。增加 CPU 个别时候会临时看到系统 Load 下降，但治标不治本。