Linux中CPU亲和性(affinity)

0、准备知识

超线程技术(Hyper-Threading)：就是利用特殊的硬件指令，把两个逻辑内核(CPU core)模拟成两个物理芯片，

让单个处理器都能使用线程级并行计算，进而兼容多线程操作系统和软件，减少了CPU的闲置时间，提高的CPU的运行效率。

我们常听到的双核四线程/四核八线程指的就是支持超线程技术的CPU.

物理CPU:机器上安装的实际CPU, 比如说你的主板上安装了一个8核CPU,那么物理CPU个数就是1个,所以物理CPU个数就是主板上安装的CPU个数。

逻辑CPU:一般情况，我们认为一颗CPU可以有多核，加上intel的超线程技术(HT)，可以在逻辑上再分一倍数量的CPU core出来；

```
逻辑CPU数量 = 物理CPU数量 x CPU cores x 2(如果支持并开启HT) //前提是CPU的型号一致，如果不一致只能一个一个的加起来，不用直接乘以物理CPU数量
//比如你的电脑安装了一块4核CPU，并且支持且开启了超线程（HT）技术，那么逻辑CPU数量 = 1 x 4 x 2 = 8
```

Linux下查看CPU相关信息, CPU的信息主要都在/proc/cupinfo中，

```
# 查看物理CPU个数
cat /proc/cpuinfo|grep "physical id"|sort -u|wc -l

# 查看每个物理CPU中core的个数(即核数)
cat /proc/cpuinfo|grep "cpu cores"|uniq

# 查看逻辑CPU的个数
cat /proc/cpuinfo|grep "processor"|wc -l

# 查看CPU的名称型号
cat /proc/cpuinfo|grep "name"|cut -f2 -d:|uniq
```

Linux查看某个进程运行在哪个逻辑CPU上

```
ps -eo pid,args,psr
#参数的含义:
pid  - 进程ID
args - 该进程执行时传入的命令行参数
psr  - 分配给进程的逻辑CPU

例子:
[~]# ps -eo pid,args,psr | grep nginx
9073 nginx: master process /usr/   1
9074 nginx: worker process         0
9075 nginx: worker process         1
9076 nginx: worker process         2
9077 nginx: worker process         3
13857 grep nginx                   3
```

Linux查看线程的TID

TID就是Thread ID,他和POSIX中pthread_t表示的线程ID完全不是同一个东西.

Linux中的POSIX线程库实现的线程其实也是一个轻量级进程(LWP),这个TID就是这个线程的真实PID.

但是又不能通过getpid()函数获取，Linux中定义了gettid()这个接口，但是通常都是未实现的，所以需要使用下面的方式获取TID。

```
//program
#include <sys/syscall.h>
pid_t tid;
tid = syscall(__NR_gettid);// or syscall(SYS_gettid)

//command-line 3种方法(推荐第三种方法)
(1) ps -efL | grep prog_name
(2) ls /proc/pid/task              //文件夹名即TID
(3) ps -To 'pid,lwp,psr,cmd' -p PID
```

1、CPU亲和性(亲和力)

1.1 基本概念

CPU affinity 是一种调度属性(scheduler property), 它可以将一个进程"绑定" 到一个或一组CPU上.

在SMP(Symmetric Multi-Processing对称多处理)架构下，Linux调度器(scheduler)会根据CPU affinity的设置让指定的进程运行在"绑定"的CPU上,而不会在别的CPU上运行.

Linux调度器同样支持自然CPU亲和性(natural CPU affinity)：调度器会试图保持进程在相同的CPU上运行，这意味着进程通常不会在处理器之间频繁迁移,进程迁移的频率小就意味着产生的负载小。

因为程序的作者比调度器更了解程序,所以我们可以手动地为其分配CPU核，而不会过多地占用CPU0，或是让我们关键进程和一堆别的进程挤在一起,所有设置CPU亲和性可以使某些程序提高性能。

1.2 表示方法

CPU affinity 使用位掩码(bitmask)表示, 每一位都表示一个CPU, 置1表示"绑定".

最低位表示第一个逻辑CPU, 最高位表示最后一个逻辑CPU.

CPU affinity典型的表示方法是使用16进制,具体如下.

```
0x00000001
    is processor #0

0x00000003
    is processors #0 and #1

0xFFFFFFFF
    is all processors (#0 through #31)
```

2、taskset命令

taskset命名用于获取或者设定CPU亲和性.

```
# 命令行形式
taskset [options] mask command [arg]...
taskset [options] -p [mask] pid

PARAMETER
       mask : cpu亲和性,当没有-c选项时, 其值前无论有没有0x标记都是16进制的,
              当有-c选项时,其值是十进制的.
       command : 命令或者可执行程序
       arg : command的参数
       pid : 进程ID,可以通过ps/top/pidof等命令获取


OPTIONS
       -a, --all-tasks （旧版本中没有这个选项）
              这个选项涉及到了linux中TID的概念,他会将一个进程中所有的TID都执行一次CPU亲和性设置.
              TID就是Thread ID,他和POSIX中pthread_t表示的线程ID完全不是同一个东西.
              Linux中的POSIX线程库实现的线程其实也是一个进程(LWP),这个TID就是这个线程的真实PID.
       -p, --pid
              操作已存在的PID,而不是加载一个新的程序
       -c, --cpu-list
              声明CPU的亲和力使用数字表示而不是用位掩码表示. 例如 0,5,7,9-11.
       -h, --help
              display usage information and exit
       -V, --version
              output version information and exit

  USAGE
       1) 使用指定的CPU亲和性运行一个新程序
          taskset [-c] mask command [arg]...
              举例:使用CPU0运行ls命令显示/etc/init.d下的所有内容
                  taskset -c 0 ls -al /etc/init.d/
       2) 显示已经运行的进程的CPU亲和性
          taskset -p pid
              举例:查看init进程(PID=1)的CPU亲和性
                  taskset -p 1
       3) 改变已经运行进程的CPU亲和力
          taskset -p[c] mask pid
              举例:打开2个终端,在第一个终端运行top命令,第二个终端中
                  首先运行:[~]# ps -eo pid,args,psr | grep top #获取top命令的pid和其所运行的CPU号
                  其次运行:[~]# taskset -cp 新的CPU号 pid      #更改top命令运行的CPU号
                  最后运行:[~]# ps -eo pid,args,psr | grep top #查看是否更改成功

  PERMISSIONS
       一个用户要设定一个进程的CPU亲和性,如果目标进程是该用户的,则可以设置,如果是其他用户的,则会设置失败,提示 Operation not permitted.当然root用户没有任何限制.

       任何用户都可以获取任意一个进程的CPU亲和性.
```

taskset命令其实就是使用sched_getaffinity()和sched_setaffinity()接口实现的,相信看完了第3节你也能自己实现一个taskset命令.

有兴趣的可以看一下其源代码:ftp://ftp.kernel.org/pub/linux/utils/util-linux/vX.YZ/util-linux-X.YZ-xxx.tar.gz /schedutils/taskset.c