

linux进程、线程与cpu的亲 and 性（ affinity ）

最近的工作中对性能的要求比较高，下面简单做一下总结：

一、什么是cpu亲和性（ affinity ）

CPU的亲和性， 就是进程要在指定的 CPU 上尽量长时间地运行而不被迁移到其他处理器，也称为CPU关联性；再简单的点的描述就将制定的进程或线程绑定到相应的cpu上；在多核运行的机器上，每个CPU本身自己会有缓存，缓存着进程使用的信息，而进程可能会被OS调度到其他CPU上，如此，CPU cache命中率就低了，当绑定CPU后，程序就会一直在指定的cpu跑，不会由操作系统调度到其他CPU上，性能有一定的提高。

软亲和性（ affinity ）：就是进程要在指定的 CPU 上尽量长时间地运行而不被迁移到其他处理器，Linux 内核进程调度器天生就具有被称为 *软CPU 亲和性（ affinity ）* 的特性，这意味着进程通常不会在处理器之间频繁迁移。这种状态正是我们希望的，因为进程迁移的频率小就意味着产生的负载小。

硬亲和性（ affinity ）：简单来说就是利用linux内核提供给用户的API，强行将进程或者线程绑定到某一个指定的cpu核运行。

解释：在linux内核中，所有的进程都有一个相关的数据结构，称为 task_struct。这个结构非常重要，原因有很多；其中与 亲和性（ affinity ）相关度最高的是 cpus_allowed 位掩码。这个位掩码由 n 位组成，与系统中的 n 个逻辑处理器——对应。 具有 4 个物理 CPU 的系统可以有 4 位。如果这些 CPU 都启用了超线程，那么这个系统就有一个 8 位的位掩码。 如果为给定的进程设置了给定的位，那么这个进程就可以在相关的 CPU 上运行。因此，如果一个进程可以在任何 CPU 上运行，并且能够根据需要在处理器之间进行迁移，那么位掩码就全是 1。实际上，这就是 Linux 中进程的缺省状态；（ 这部分内容在这个博客中有提到一点：<http://www.cnblogs.com/wenqiang/p/4802619.html>）

cpus_allowed用于控制进程可以在哪里处理器上运行

- sched_set_affinity() （ 用来修改位掩码 ）

- sched_get_affinity() （ 用来查看当前的位掩码 ）

二、进程与cpu的绑定

sched_setaffinity可以将某个进程绑定到一个特定的CPU。你比操作系统更了解自己的程序，为了避免调度器愚蠢的调度你的程序，或是为了在多线程程序中避免缓存失效造成的开销，你可能会希望这样做


在进行进程与cpu的绑定前，我们先了解编写程序需要准备的知识

Linux Programmer's Manual


```
1 SCHED_SETAFFINITY(2)
SCHED_SETAFFINITY(2)
2
3 NAME
4     sched_setaffinity, sched_getaffinity - set and get a process's CPU affinity mask
5
6 SYNOPSIS
7     #define _GNU_SOURCE          /* See feature_test_macros(7) */
8     #include <sched.h>
9
10    int sched_setaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);
11    /*该函数设置进程为pid的这个进程,让它运行在mask所设定的CPU上.如果pid的值为0,
12     *则表示指定的是当前进程,使当前进程运行在mask所设定的那些CPU上.
13     *第二个参数cpusetsize是mask所指定的数的长度,通常设定为sizeof(cpu_set_t).
14     *如果当前pid所指定的进程此时没有运行在mask所指定的任意一个CPU上,
15     *则该指定的进程会从其它CPU上迁移到mask的指定的一个CPU上运行.*/
16
17    int sched_getaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);
18    /*该函数获得pid所指示的进程的CPU位掩码,并将该掩码返回到mask所指向的结构中.
19     *即获得指定pid当前可以运行在哪些CPU上.
20     *同样,如果pid的值为0,也表示的是当前进程*/
21
22 RETURN VALUE
23     On success, sched_setaffinity() and sched_getaffinity() return 0. On error, -1 is returned, and errno is set appropriately.
```




设置cpu affinity还需要用到一下宏函数




```
1 void CPU_ZERO (cpu_set_t *set)
2 /*这个宏对 CPU 集 set 进行初始化,将其设置为空集.*/
3 void CPU_SET (int cpu, cpu_set_t *set)
4 /*这个宏将 指定的 cpu 加入 CPU 集 set 中*/
5 void CPU_CLR (int cpu, cpu_set_t *set)
6 /*这个宏将 指定的 cpu 从 CPU 集 set 中删除.*/
7 int CPU_ISSET (int cpu, const cpu_set_t *set)
8 /*如果 cpu 是 CPU 集 set 的一员,这个宏就返回一个非零值 (true),否则就返回零 (false).*/
9
```



下面下一个具体的例子：将当前进程绑定到0、1、2、3号cpu上



```
1 #define _GNU_SOURCE
2 #include <sched.h>
3 #include <stdio.h>
4 #include <string.h>
5 #include <stdlib.h>
6 #include <unistd.h>
7 #include <errno.h>
8
9 /* sysconf( _SC_NPROCESSORS_CONF ) 查看cpu的个数;打印用%ld长整.
10  * sysconf( _SC_NPROCESSORS_ONLN ) 查看在使用的cpu个数;打印用%ld长整 */
11 int main(int argc, char **argv)
12 {
13     int cpus = 0;
14     int i = 0;
15     cpu_set_t mask;
16     cpu_set_t get;
17
18     cpus = sysconf(_SC_NPROCESSORS_CONF);
19     printf("cpus: %d\n", cpus);
20
21     CPU_ZERO(&mask); /* 初始化set集,将set置为空*/
22     CPU_SET(0, &mask); /* 依次将0、1、2、3号cpu加入到集合,前提是你的机器是多核处理器*/
23     CPU_SET(1, &mask);
24     CPU_SET(2, &mask);
25     CPU_SET(3, &mask);
26
27     /*设置cpu 亲和性 (affinity)*/
28     if (sched_setaffinity(0, sizeof(mask), &mask) == -1) {
29         printf("Set CPU affinity failure, ERROR:%s\n", strerror(errno));
30         return -1;
31     }
32     usleep(1000); /* 让当前的设置有足够时间生效*/
33
34     /*查看当前进程的cpu 亲和性*/
35     CPU_ZERO(&get);
36     if (sched_getaffinity(0, sizeof(get), &get) == -1) {
37         printf("get CPU affinity failure, ERROR:%s\n", strerror(errno));
38         return -1;
39     }
40
41     /*查看运行在当前进程的cpu*/
42     for(i = 0; i < cpus; i++) {
43
44         if (CPU_ISSET(i, &get)) { /*查看cpu i 是否在get 集合当中*/
45             printf("this process %d of running processor: %d\n", getpid(), i);
46         }
47     }
48     sleep(3); //让程序停在这儿,方便top命令查看
49
50     return 0;
51 }
```



运行结果如下：

```
1 [root@localhost test]# ./test
2 cpus: 24
3 this process 2848 of running processor: 0
4 this process 2848 of running processor: 1
5 this process 2848 of running processor: 2
6 this process 2848 of running processor: 3
```

上面代码当中用到了syscall这个函数，顺便也在这里做一下说明

syscall是执行一个系统调用，根据指定的参数number和所有系统调用的接口来确定调用哪个系统调用，用于用户空间跟内核之间的数据交换

公告

昵称：z572089387
园龄：2年
粉丝：7
关注：16
+加关注

< 2017年8月 >						
日	一	二	三	四	五	六
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

最新随笔

1. 记录几个经典的字符串hash算法
2. linux timerfd系列函数总结
3. linux 获取网络状态信息（ Rtnetlink ）
4. linux netlink通信机制
5. linux进程、线程与cpu的亲 and 性（ affinity ）
6. C语言检查ip是否合法
7. 使用libpcap获取http报文
8. 使用libpcap过滤arp
9. 初识函数库libpcap
10. HTTP协议GET HEAD简单介绍

随笔分类

C/C++(7)
CAPWAP学习笔记(7)
libpcap学习笔记(3)
linux内核(10)
Shell学习(2)
Tiny4412裸板开发(10)
常用小程序(1)
工作笔记
其他(1)
网络编程(6)
系统编程(6)

随笔档案

2017年4月 (2)
2017年3月 (1)
2017年1月 (1)
2016年11月 (1)
2016年10月 (1)
2016年8月 (1)
2016年7月 (4)
2016年5月 (5)
2016年4月 (7)
2016年1月 (7)
2015年12月 (1)
2015年11月 (2)
2015年10月 (2)
2015年9月 (12)
2015年8月 (14)

积分与排名

积分 - 22108
排名 - 14162

最新评论

1. Re:使用libpcap获取http报文
@z572089387懂了懂了，b是一行的起始位置，a是一行的结束位置。谢谢楼主分享...
--青儿哥哥
2. Re:使用libpcap获取http报文
@青儿哥哥该函数主要功能是判断是不是HTTP GET 报文，如果是就进行解析，通过字符串"vvn"即（ 0x0doa ）进行字符串分割，分辨获取HTTP GET报文信息中的Host 及文件路径，最后组装Ur.....
--z572089387
3. Re:使用libpcap获取http报文
楼主：parse_http_head 函数看不太明白，能大概说一下思路吗？？万分感谢
--青儿哥哥
4. Re:使用libpcap获取http报文
@青儿哥哥在工作，刚刚结束学生时代！！...
--z572089387
5. Re:使用libpcap获取http报文
@z572089387嗯嗯，谢谢。楼主是学生还是工作了？...
--青儿哥哥

阅读排行榜

1. linux下redis数据库的简单使用(11689)
2. linux几种时间函数总结(10634)
3. linux几种定时函数的使用(4772)
4. linux进程、线程与cpu的亲 and 性（ affinity ）(1417)
5. Tiny4412 烧写uboot到jemmc步骤(1042)

评论排行榜

1. 使用libpcap获取http报文(6)

推荐排行榜

1. linux timerfd系列函数总结(1)
2. 使用libpcap获取http报文(1)
3. linux下redis数据库的简单使用(1)

下面是syscal函数原型及一些常用的number

```
1 //syscall - indirect system call
2 SYNOPSIS
3     #define _GNU_SOURCE          /* See feature_test_macros(7) */
4     #include <unistd.h>
5     #include <sys/syscall.h>    /* For SYS_xxx definitions */
6
7     int syscall(int number, ...);
8
9 /* sysconf( _SC_PAGESIZE );  此宏查看缓存内存页面的大小；打印用%d长整型。
10 sysconf( _SC_PHYS_PAGES )  此宏查看内存的总页数；打印用%d长整型。
11 sysconf( _SC_AVPHYS_PAGES ) 此宏查看可以利用的总页数；打印用%d长整型。
12 sysconf( _SC_NPROCESSORS_CONF ) 查看cpu的个数；打印用%d长整。
13 sysconf( _SC_NPROCESSORS_ONLN ) 查看在使用的cpu个数；打印用%d长整。
14 (long long)sysconf(_SC_PAGESIZE) * (long long)sysconf(_SC_PHYS_PAGES)  计算内存大小。
15 sysconf( _SC_LOGIN_NAME_MAX ) 查看最大登录名长度；打印用%d长整。
16 sysconf( _SC_HOST_NAME_MAX ) 查看最大主机长度；打印用%d长整。
17 sysconf( _SC_OPEN_MAX ) 每个进程运行时打开的文件数目；打印用%d长整。
18 sysconf( _SC_CLK_TCK) 查看每秒中跑过的运算速率；打印用%d长整。*/
```

三、线程与cpu的绑定

线程于进程的绑定方法大体一致，需要注意的是线程绑定于进程的分别是所用函数不一样

线程绑定用到下面两个函数，跟进程类似就不做详细说明，下面直接贴出函数原型：

```
1 NAME
2     pthread_setaffinity_np, pthread_getaffinity_np - set/get CPU affinity of a thread
3
4 SYNOPSIS
5     #define _GNU_SOURCE          /* See feature_test_macros(7) */
6     #include <pthread.h>
7
8     int pthread_setaffinity_np(pthread_t thread, size_t cpusetsize,
9                               const cpu_set_t *cpuset);
10    int pthread_getaffinity_np(pthread_t thread, size_t cpusetsize,
11                              cpu_set_t *cpuset);
12
13    Compile and link with -pthread.
14
15 DESCRIPTION
16    The pthread_setaffinity_np() function sets the CPU affinity mask of the thread thread to the CPU set pointed to by cpuset. If the call is
17    successful, and the thread is not
18    currently running on one of the CPUs in cpuset, then it is migrated to one of those CPUs.
19
20    The pthread_getaffinity_np() function returns the CPU affinity mask of the thread thread in the buffer pointed to by cpuset.
21
22    For more details on CPU affinity masks, see sched_setaffinity(2). For a description of a set of macros that can be used to manipulate and inspect CPU
23    sets, see CPU_SET(3).
24
25    The argument cpusetsize is the length (in bytes) of the buffer pointed to by cpuset. Typically, this argument would be specified as sizeof(cpu_set_t).
26    (It may be some other
27    value, if using the macros described in CPU_SET(3) for dynamically allocating a CPU set.)
28
29 RETURN VALUE
30    On success, these functions return 0; on error, they return a nonzero error number
```

下面同样是个具体的例子：将当前线程绑定到0、1、2、3号cpu上

```
1 #define _GNU_SOURCE
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <unistd.h>
6 #include <pthread.h>
7 #include <sched.h>
8
9 void *testfunc(void *arg)
10 {
11     int i, cpus = 0;
12     cpu_set_t mask;
13     cpu_set_t get;
14
15     cpus = sysconf(_SC_NPROCESSORS_CONF);
16     printf("this system has %d processor(s)\n", cpus);
17
18     CPU_ZERO(&mask);
19     for (i = 0; i < 4; i++) { /*将0、1、2、3添加到集合中*/
20         CPU_SET(i, &mask);
21     }
22
23     /* 设置cpu 亲和性(affinity)*/
24     if (pthread_setaffinity_np(pthread_self(), sizeof(mask), &mask) < 0) {
25         fprintf(stderr, "set thread affinity failed\n");
26     }
27
28     /* 查看cpu 亲和性(affinity)*/
29     CPU_ZERO(&get);
30     if (pthread_getaffinity_np(pthread_self(), sizeof(get), &get) < 0) {
31         fprintf(stderr, "get thread affinity failed\n");
32     }
33
34     /* 查看当前线程所运行的所有cpu*/
35     for (i = 0; i < cpus; i++) {
36         if (CPU_ISSET(i, &get)) {
37             printf("this thread %d is running in processor %d\n", (int)pthread_self(), i);
38         }
39     }
40     sleep(3); //查看
41
42     pthread_exit(NULL);
43 }
44
45 int main(int argc, char *argv[])
46 {
47     pthread_t tid;
48     if (pthread_create(&tid, NULL, (void *)testfunc, NULL) != 0) {
49         fprintf(stderr, "thread create failed\n");
50         return -1;
51     }
52
53     pthread_join(tid, NULL);
54     return 0;
55 }
```


运行结果如下：

```
1 [root@localhost thread]# ./test
2 this system has 24 processor(s)
3 this thread 2812323584 is running in processor 0
4 this thread 2812323584 is running in processor 1
5 this thread 2812323584 is running in processor 2
6 this thread 2812323584 is running in processor 3
```

分类: 系统编程 , linux内核

好文置顶 关注我 收藏该文



z572089387

关注 - 16

粉丝 - 7

+加关注

« 上一篇：C语言检查ip是否合法
» 下一篇：linux netlink通信机制

posted @ 2016-11-17 17:39 z572089387 阅读(1417) 评论(0) 编辑 收藏

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。
【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库
【推荐】极光开发者服务平台，五大功能一站集齐

刷新评论 刷新页面 返回顶部