设置、查看 进程、线程、中断 绑核、调度策略、优先级 参数 的 程序内API、命令行命令

绑核

中断

中断请求（IRQ）有一个亲和度属性smp_affinity，smp_affinity决定允许哪些CPU核心处理IRQ。

某一特定IRQ的亲和度值储存在/proc/irq/IRP_NUMBER/smp_affinity文件中，储存的值是一个十六进制位掩码(hexadecimal bit-mask)，代表着系统的所有CPU核心。从低位到高位的每个bit代表从编号从0开始的CPU能否接受此IRQ，1表示接受，0表示拒绝。

命令行设置

cat /proc/interrupts 查看所有设备的中断请求信息

orin中如图，第一列为IRQ的编号

```
lotus@tegra-ubuntu:~$ cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7       CPU8       CPU9       CPU10
 11:    1907293    2155009     510297    1677919     282231     268483     235942     294196     693740     693298     666733     GICv3  27 Level     arch_timer
 16:        100          0          0          0          0          0          0          0          0          0          0     GICv3 682 Edge      ivc407
 18:          9          0          0          0          0          0          0          0          0          0          0     GICv3 684 Edge      gr-virt
 20:     159919          0          0          0          0          0          0          0          0          0          0     GICv3 686 Edge      vblk
 25:         53          0          0          0          0          0          0          0          0          0          0     GICv3 691 Edge      vblk
 26:         52          0          0          0          0          0          0          0          0          0          0     GICv3 692 Edge      vblk
 27:        187          0          0          0          0          0          0          0          0          0          0     GICv3 693 Edge      vblk
 29:         23          0          0          0          0          0          0          0          0          0          0     GICv3 695 Edge      vblk
 30:         61          0          0          0          0          0          0          0          0          0          0     GICv3 696 Edge      vblk
 31:          4          0          0          0          0          0          0          0          0          0          0     GICv3 697 Edge      tegra_hv_pm_ctl
 32:      13116          0          0          0          0          0          0          0          0          0          0     GICv3 698 Edge      bpmp_irq_handler
 33:         28          0          0          0          0          0          0          0          0          0          0     GICv3 699 Edge      bpmp_irq_handler
 34:          2          0          0          0          0          0          0          0          0          0          0     GICv3 700 Edge      bpmp_irq_handler
 36:          1          0          0          0          0          0          0          0          0          0          0     GICv3 702 Edge      vse
 37:          2          0          0          0          0          0          0          0          0          0          0     GICv3 703 Edge      ivc520
 38:          2          0          0          0          0          0          0          0          0          0          0     GICv3 704 Edge      ivc521
 39:          2          0          0          0          0          0          0          0          0          0          0     GICv3 705 Edge      ivc506
 40:          0          0          0          0          0          0          0          0          0          0          0     GICv3 255 Level     mc_status
 42:          0          0          0          0          0          0          0          0          0          0          0     GICv3 202 Level     arm-smmu global fault
```

cat /proc/irq/N/smp_affinity 查看IRQ号为N的绑核信息，返回值是一个十六进制位掩码

例如，想在orin中查看32号IRQ的绑核情况，输入cat /proc/irq/32/smp_affinity。如下图所示，orin的默认值为7ff，而orin本身有11个CPU 核，所以表示这个IRQ能被所有CPU接受处理。

```
lotus@tegra-ubuntu:~$ cat /proc/irq/32/smp_affinity
7ff
```

echo 1 >/proc/irq/N/smp_affinity 修改IRQ号为N的绑核信息

例如，想在orin中修改32号IRQ的绑核参数，让其绑定到CPU0上执行，输入echo 1 >/proc/irq/32/smp_affinity。（注：需要在root下执行）

```
root@tegra-ubuntu:~# echo 1 > /proc/irq/32/smp_affinity
root@tegra-ubuntu:~# cat /proc/irq/32/smp_affinity
001
root@tegra-ubuntu:~#
```

程序内设置（orin下用不了，没有cpumask定义）

```
void cpumask_set_cpu(unsigned int cpu, struct cpumask *cpu_mask);
```

Linux内核提供cpumask结构体来帮助设置中断的绑定。cpumask通常指向一个32位整数，整数的二进制下的每个位表示一个CPU，1表示irq使用该CPU，0相反。cpumask的bit的置位表示了一个CPU的子集。通过cpumask_set_cpu函数可以向cpumask结构体表示的CPU子集添加CPU，第一个参数表示需要添加的CPU编号，第二个参数表示cpumask结构体表示的CPU子集。

```
int irq_set_affinity(unsigned int irq, struct cpumask *cpu_mask);
```

通过irq_set_affinity函数可以设置指定中断的绑定CPU集合。第一个参数irq表示中断编号，第二个指针参数cpu_mask指向设置好的CPU集合。

例程如下：

```
#include <stdio.h>
#include <include/linux/cpumask.h>
#include <lib/cpumask.c>
#include <kernel/cpu.c>

int main()
{
    struct cpumask cpu_mask;
    cpumask_set_cpu(1, &cpu_mask);
    irq_set_affinity(32, &cpu_mask);
    return 0;
}
```

进程/线程

进程/线程同样具有亲和度属性affinity，affinity决定允许哪些CPU核心处理进程/线程。

affinity储存的值是一个十六进制位掩码(hexadecimal bit-mask)，代表着系统的所有CPU核心。从低位到高位的每个bit代表从编号从0开始的CPU能否执行此进程/线程，1表示接受，0表示拒绝。

CPU宏与CPU核心集表示

set是一个指向十六进制位的整数的指针，整数的从低位到高位的每个bit代表从编号从0开始的CPU，1表示指定CPU在集合中，0表示不在集合中。

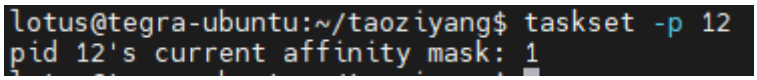
```
void CPU_ZERO(cpu_set_t *set)   //初始化CPU集
void CPU_SET(int cpu,cpu_set_t *set)  //向CPU集中添加某个CPU
void CPU_CLR(int cpu,cpu_set_t *set)  //从CPU集中移除某个CPU
int CPU_ISSET(int cpu,cpu_set_t *set)  //检查CPU集中是否存在某个CPU
int CPU_COUNT(cpu_set_t *set)          //返回CPU集中CPU的个数
```

进程

命令行设置

taskset -p pid 查看指定pid的进程的情况，返回值是进程的亲和度affinity，以十六进制位掩码表示

例如查看12号进程的绑核情况，输入taskset -p 12，如下图所示。可以看到亲和度值为1，表示此进程绑定到CPU0上。



taskset -p mask pid 以掩码形式修改指定pid的进程的绑核信息，其中mask是需要绑定的CPU集合的十六进制掩码

例如修改2219470号进程的绑核情况，输入taskset -p 0x001 2219470，如下图所示。可以看到原来亲和度值为7ff，后修改为1。这表示2219470号进程从可以在任意CPU上运行变为绑定到CPU0上。

```
lotus@tegra-ubuntu:~/taoziyang$ taskset -p 0x001 2219470
pid 2219470's current affinity mask: 7ff
pid 2219470's new affinity mask: 1
```

taskset -cp cpu-list pid 以列表形式修改指定pid的进程的绑核信息，其中cpu-list是是需要绑定的CPU核的列表，多个不连续的cpu可用逗号连接，连续的可用短线连接，比如0,2,5-11等。

例如修改2219470号进程的绑核情况，输入taskset -p 1-2 2219470，如下图所示。可以看到原来CPU绑定列表为0，后修改为1,2。这表示2219470号进程从绑定到CPU0上运行变为绑定到CPU1,2上。

```
lotus@tegra-ubuntu:~/taoziyang$ taskset -cp 1-2  2219470
pid 2219470's current affinity list: 0
pid 2219470's new affinity list: 1,2
```

taskset -c cpu-list ./executable_program& 进程启动时以列表形式绑核

例如启动.run可执行文件时绑定到CPU1,2上执行，输入taskset -c 1-2 ./run&，如下图所示。启动后查看对应京城的CPU绑定掩码为6，这表示可执行文件绑定到CPU1,2上。

```
lotus@tegra-ubuntu:~/taoziyang$ taskset -c 1-2 ./run&
[1] 2255420
lotus@tegra-ubuntu:~/taoziyang$ taskset -p 2255420
pid 2255420's current affinity mask: 6
```

程序内设置

```
int sched_setaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);
```

设定进程号为pid的进程运行在mask所设定的CPU上，第二个数cpusetsize是mask所指定的数的长度，通常为sizeof(cpu_set_t)。如果pid的值为0，则表示的是当前进程。*mask 通过CPU宏设置。

```
int sched_getaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);
```

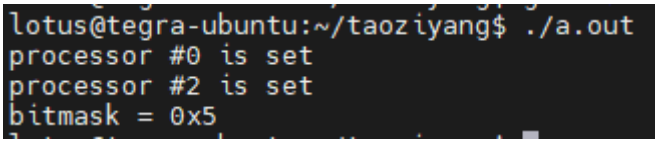
获取进程号为pid的进程的绑核CPU集合到mask中。

例程如下：

```
#define _GNU_SOURCE
#include <sched.h>
#include <unistd.h> /* sysconf */
#include <stdlib.h> /* exit */
#include <stdio.h>

int main(void)
{
    int i, nrcpus;
    cpu_set_t mask;
    unsigned long bitmask = 0;

    CPU_ZERO(&mask);

    CPU_SET(0, &mask); /* add CPU0 to cpu set */
    CPU_SET(2, &mask); /* add CPU2 to cpu set */

    /* Set the CPU affinity for a pid */
    if (sched_setaffinity(0, sizeof(cpu_set_t), &mask) == -1)
    {
        perror("sched_setaffinity");
        exit(EXIT_FAILURE);
```

```
    }

    CPU_ZERO(&mask);

     /* Get the CPU affinity for a pid */
    if (sched_getaffinity(0, sizeof(cpu_set_t), &mask) == -1)
    {
        perror("sched_getaffinity");
        exit(EXIT_FAILURE);
    }

    /* get logical cpu number */
    nrcpus = sysconf(_SC_NPROCESSORS_CONF);

    for (i = 0; i < nrcpus; i++)
    {
        if (CPU_ISSET(i, &mask))
        {
            bitmask |= (unsigned long)0x01 << i;
            printf("processor #%d is set\n", i);
        }
    }
    printf("bitmask = %#lx\n", bitmask);

    exit(EXIT_SUCCESS);
}
```

运行结果如下：



线程

程序内设置

```
int pthread_setaffinity_np(pthread_t thread, size_t cpusetsize, cpu_set_t
*cpuset);
```

设定线程号为thread的线程运行在mask所设定的CPU上，第二个数cpusetsize是mask所指定的数的长度，通常为sizeof(cpu_set_t)。如果thread的值为0，则表示的是当前线程。*mask 通过CPU宏设置。

```
int pthread_getaffinity_np(pthread_t thread, size_t cpusetsize, cpu_set_t
*cpuset);
```

获取进程号为pid的进程的绑核CPU集合到cpuset中。

例程如下：

```
#define _GNU_SOURCE
#include <pthread.h> //不用再包含<sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
```

```c
#define handle_error_en(en, msg) \
        do { errno = en; perror(msg); exit(EXIT_FAILURE); } while (0)

int main(int argc, char *argv[])
{
    int s, j;
    cpu_set_t cpuset;
    pthread_t thread;

    thread = pthread_self();

    /* Set affinity mask to include CPUs 0 to 7 */
    CPU_ZERO(&cpuset);
    for (j = 0; j < 8; j++)
        CPU_SET(j, &cpuset);

    s = pthread_setaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
    if (s != 0)
    {
        handle_error_en(s, "pthread_setaffinity_np");
    }

    /* Check the actual affinity mask assigned to the thread */
    s = pthread_getaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
    if (s != 0)
    {
        handle_error_en(s, "pthread_getaffinity_np");
    }

    printf("Set returned by pthread_getaffinity_np() contained:\n");
    for (j = 0; j < CPU_SETSIZE; j++) //CPU_SETSIZE 是定义在<sched.h>中的宏，通常是
1024
    {
        if (CPU_ISSET(j, &cpuset))
        {
            printf("CPU %d\n", j);
        }
    }
    exit(EXIT_SUCCESS);
}
```

运行结果如下：

```
lotus@tegra-ubuntu:~/taoziyang$ gcc a.c -lpthread
lotus@tegra-ubuntu:~/taoziyang$ ./a.out
Set returned by pthread_getaffinity_np() contained:
    CPU 0
    CPU 1
    CPU 2
    CPU 3
    CPU 4
    CPU 5
    CPU 6
    CPU 7
```

调度策略、优先级

调度策略种类

非实时进程调度策略

SCHED_OTHER：标准循环分时策略。

SCHED_BATCH：用于"批处理"样式的进程执行。

SCHED_IDLE：用于运行优先级较低的后台作业。

实时进程调度策略

SCHED_FIFO：先进先出策略。

SCHED_RR：循环策略。

SCHED_DEADLINE：运行此策略的线程将有最高的优先级（高于前面任意一种策略），当这个线程是就绪状态时，他会抢占任何其他策略的线程。该线程不能调用fork。运行此策略的进程需要给定三个进程参数：sched_runtime，sched_deadline，sched_period，分别表示进程运行时间，进程相对时限，进程周期。sched_runtime需要略大于该进程的平均执行时间，可以设置为WCET。kernel要求 sched_runtime <= sched_deadline <= sched_period，如果sched_period设置为0，它会被自动设置为与sched_deadline相同。并且所有这些值要求大于1024，单位为微秒。

linux API优先级设置参数与top下的优先级参数的映射关系

用户、top命令优先级表示法

- 用户优先级表示法
  - 设置实时优先级：设置RT参数，范围[0,99]，值越高优先级越高。
  - 设置静态优先级：设置NICE值，范围[-20,19]，值越低优先级越高。
- top优先级表示法

```
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
1289780 lotus     20   0 3323856  27236  25508 S   0.7   0.1  16:44.44 AA_Performance_
     22 root      20   0       0      0      0 S   0.3   0.0   2:52.90 ksoftirqd/1
    843 root     -51   0       0      0      0 S   0.3   0.0  10:44.87 irq/82-mgbe2_0.
   9310 root      20   0  342032  35304  34812 S   0.3   0.1   4:20.28 adg_dlt_systemd
1576297 lotus     20   0   14988   5380   4000 S   0.3   0.0   0:04.56 sshd
1638926 lotus     20   0    8140   3632   2608 R   0.3   0.0   0:01.30 top
```

上图中涉及优先级的有两个参数：PR、NI。PR代表进程优先级，NI代表进程的NICE值，由用户态设置，NI值是对PR值的修正。PR值的范围是[-100,39]

用户、top命令优先级的转换关系

拥有静态优先级的进程中，PR、NI的关系式为：

$$PR = 20 + NI$$

拥有实时优先级的进程中，NI恒等于0，PR的计算方式为：

$$PR = -1 - RT$$

特别地，RT=99，即为最高实时优先级时，top中的PR显示为rt。

观察到NICE值的范围[-20,19]，所以静态优先级进程的PR值的范围是[0,39]；RT值的范围[0,99]，所以实时优先级进程的PR值的范围是[-100,-1]。所以PR值大于0时，这是个静态优先级进程；PR值小于0时，这是个实时优先级进程。

以下是例子：

用户态：设置实时优先级RT为50，调度策略为SCHED_FIFO。top下显示：PR=-51，NI=0

```
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
1638718 root     -51   0    1792    456    392 R  95.3   0.0   2:30.96 a.out
1289780 lotus     20   0 3323856  27236  25508 S   1.3   0.1  16:43.66 AA_Performance_
     12 root      20   0       0      0      0 S   0.3   0.0   3:32.74 ksoftirqd/0
    922 root     -51   0       0      0      0 S   0.3   0.0   6:57.25 irq/90-mgbe3_0.
1289774 lotus     20   0  724600 721596   5848 S   0.3   2.4   0:47.39 gdb
1601780 lotus     20   0   14988   5384   3996 S   0.3   0.0   0:02.39 sshd
```

用户态：设置静态优先级NICE为5，调度策略为SCHED_OTHER。top下显示：PR=25，NI=5

```
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
1679966 root      25   5    1792    416    352 R  99.7   0.0   0:07.06 a.out
1289780 lotus     20   0 3323856  27236  25508 S   0.7   0.1  16:56.43 AA_Performance_
1638926 lotus     20   0    8140   3632   2608 R   0.7   0.0   0:12.85 top
     22 root      20   0       0      0      0 S   0.3   0.0   2:53.89 ksoftirqd/1
     46 root      20   0       0      0      0 S   0.3   0.0   4:00.96 ksoftirqd/5
```

进程调度策略、优先级设置

命令行设置

非实时进程

查看nice值可以使用**top**命令。如下图所示，NI列为nice值。

```
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
2303394 lotus     20   0    1792    416    352 R 100.0   0.0   0:05.24 run
```

nice -n priority ./executable_program & 启动程序时设置非实时进程的优先级

例如启动run程序时设置nice值为10，输入nice -n 10 ./run &，如下图所示。

```
lotus@tegra-ubuntu:~/taoziyang$ nice -n 10 ./run &
```

top查看

```
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
2335310 lotus     30  10    1792    424    360 R 100.0   0.0   1:31.67 run
```

renice -n priority pid 调整运行中进程pid的nice值

例如修改2335310号进程的nice值为5，输入renice -n 5 2335310，如下图所示。

```
lotus@tegra-ubuntu:~/taoziyang$ sudo renice -n 5 2335310
[sudo] password for lotus:
2335310 (process ID) old priority 10, new priority 5
```

实时进程

chrt -p pid 查看进程pid的调度策略、优先级

例如查看2303394号进程的调度策略、优先级，输入chrt -p 2303394，如下图所示。

```
lotus@tegra-ubuntu:~/taoziyang$ chrt -p 2303394
pid 2303394's current scheduling policy: SCHED_OTHER
pid 2303394's current scheduling priority: 0
```

chrt policy -p priority pid 设置实时进程pid的调度策略、优先级。policy可以输入的命令有：

-o for SCHED_OTHER,

-b for SCHED_BATCH,

-i for SCHED_IDLE,

-f for SCHED_FIFO,

-r for SCHED_RR,

-d for SCHED_DEADLINE

例如修改2303394号进程的调度策略为SCHED_FIFO、实时优先级为50，输入chrt -f -p 50 2303394，如下图所示。

```
lotus@tegra-ubuntu:~/taoziyang$ sudo chrt -f -p 50 2303394
[sudo] password for lotus:
lotus@tegra-ubuntu:~/taoziyang$ chrt -p 2303394
pid 2303394's current scheduling policy: SCHED_FIFO
pid 2303394's current scheduling priority: 50
```

特别地，执行SCHED_DEADLINE调度策略的进程需要给定参数sched_runtime，sched_deadline，sched_period。具体方式为：

-T nanoseconds for sched_runtime，

-P nanoseconds for sched_peroid，

-D nanoseconds for sched_deadline

程序内设置

非SCHED_DEADLINE调度策略的进程

设置、获取进程pid的调度策略

```
struct sched_param param;
param.sched_priority = NUM;
int sched_setscheduler(pid_t pid, int policy, const struct sched_param *param);
//设置优先级和调度策略
int sched_getscheduler(pid_t pid); //获取调度策略
```

若对当前进程操作，pid需要设为0。为policy表示调度策略，值域为：SCHED_OTHER，SCHED_BATCH，SCHED_IDLE，SCHED_FIFO，SCHED_RR。优先级通过param结构体设置。

非实时调度策略的优先级必须指定为0，实时策略sched_priority指定为1~99。

sched_getscheduler的调度策略返回值是一个非负整数。对应关系如下表

sched_getscheduler的调度策略返回值	对应调度策略
0	SCHED_OTHER
1	SCHED_FIFO
2	SCHED_RR
3	SCHED_BATCH
5	SCHED_IDLE

设置、获取进程pid的优先级

```
struct sched_param param;
param.sched_priority = NUM;
int sched_setparam(pid_t pid, const struct sched_param *param); //设置优先级
int sched_getparam(pid_t pid, struct sched_param *param); //获取优先级
```

若对当前进程操作，pid需要设为0。

SCHED_DEADLINE策调度略的进程

```
struct sched_attr attr;
attr.sched_runtime = pid_WCET;
attr.sched_deadline = pid_deadline;
attr.sched_period = pid_period;
int syscall(SYS_sched_setattr, pid_t pid, struct sched_attr *attr,
                    unsigned int flags);
```

进程优先级通过attr结构体设置。attr结构体需要设置的参数有：sched_runtime，sched_deadline，sched_period。flag位为保留位，目前必须设置为0。

线程调度策略、优先级设置

程序内设置

首先给线程创建属性并初始化

```
pthread_t thread_id;
pthread_attr_t thread_attr;
pthread_attr_init(&thread_attr);
```

再给线程设置优先级

```
struct sched_param thread_param;
param.sched_priority = 51;
int pthread_attr_setschedparam(pthread_attr_t *attr, const struct sched_param
*param); //设置优先级
int pthread_attr_getschedparam(pthread_attr_t *attr, struct sched_param *param);
//获取优先级
```

再给线程设置调度策略。这里policy的值域为：SCHED_OTHER，SCHED_BATCH，SCHED_IDLE，SCHED_FIFO，SCHED_RR

```
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy); //设置调度策略
int pthread_attr_getschedpolicy(pthread_attr_t *attr, int policy); //获取调度策略
```

最后调用pthread_create创建线程

```
pthread_create(&thread_id, &thread_attr, NULL, NULL);
```