



# wangfan 的 Linux 进程笔记-精简版

20180522 Edition

linux 进程管理 – 宋宝华

wangfanstar@163.com

2018 年 6 月 2 日

# 目录大纲

## PART I

前言 1

---

## PART II

进程课第 1 天 2

---

1 进程的代码结构 3

2 进程的状态特征 6

## PART III

进程课第 2 天 7

---

3 进程出生 8

4 进程运行 11

5 进程死亡 12

## PART IV

进程课第 3 天 13

---

6 进程分类 14

7 进程调度策略 15

8 调整优先级 16

## PART V

进程课第 4 天 17

---

9 负载均衡 18

10 实时系统 23

## PART VI

进程问题集锦 26

---

11 课后答疑 27

## PART VII

参考资料 28

---

12 参考文献 29

13 相关附件 30

# 目 录

## PART I

### 前言 1

---

## PART II

### 进程课第 1 天 2

---

#### 1 进程的代码结构 3

- 1.1 进程控制块 PCB 与 task\_struct 3
- 1.2 task\_struct 的属性特点 4
  - fork 炸弹让 linux 死机 4
  - pid 数量限制导致安卓的一键 root 4
  - linux 进程 task\_struct 的三种数据结构 5

#### 2 进程的状态特征 6

- 2.1 进程状态切换 6
- 2.2 进程的内存泄露 6

## PART III

### 进程课第 2 天 7

---

#### 3 进程出生 8

- 3.1 进程出生时资源处理 8
- 3.2 进程分裂时的资源变化 – COW 8
  - COW 现象代码 8
  - COW 实现技术原理 9
  - 无法用 COW 的情况: Vfork 10
- 3.3 第 1 个进程, 进程 0 与进程 1 10

#### 4 进程运行 11

- 4.1 进程睡眠 11
- 4.2 进程等待 11

#### 5 进程死亡 12

- 5.1 子死父收尸 12
- 5.2 父死子托孤 12

## PART IV

### 进程课第 3 天 13

---

#### 6 进程分类 14

- 6.1 CPU 消耗与 IO 消耗型 14

#### 7 进程调度策略 15

- 7.1 RT 进程调度 15
  - SCHED\_FIFO 15
  - SCHED\_RR 15
- 7.2 NORMAL 进程调度 15
  - CFS 调度 15

#### 8 调整优先级 16

- 8.1 用 renice 改变进程优先级 16
- 8.2 用 nice 改变进程优先级 16
- 8.3 用 chrt 改变进程优先级 16

## PART V

### 进程课第 4 天 17

---

#### 9 负载均衡 18

- 9.1 LINUX 下的负载均衡处理对象 18
- 9.2 中断负载均衡 18
- 9.3 软中断负载均衡-rps 19

9.4	进程间 (task_struct) 负载均衡	20
	linux 负载均衡算法原则	20
	设置进程在指定 CPU 上运行	20
	给进程指定比例的 CPU 负载-cgroup	22

## 10 实时系统 23

10.1	Real Time 实时系统的含义	23
10.2	抢占: Linux 无法硬实时的原因	23
10.3	Linux 实时补丁的用法	24

## PART VI

进程问题集锦	26
--------	----

---

11	课后答疑	27
----	------	----

## PART VII

参考资料	28
------	----

---

12	参考文献	29
----	------	----

12.1	宋宝华相关网站资源	29
12.2	代码网址	29

13	相关附件	30
----	------	----

13.1	pdf 课件	30
13.2	视频文件	30

## PART I

# 前言

起

因：文档是我在学习宋宝华老师 2018.05.22 开始的 4 天进程课程中做的笔记，内容大部分来自老师的课程，其中根据自己的理解调整了章节架构和顺序，可能有些内容和实际上课的有差异，另外上课的课件和几十个视频以 PDF 附件的形式在文档的最后参考资料中。用支持 PDF 附件的阅读器打开即可，测试用 ADOBE 的 PDF 阅读器打开没有问题。

文档相关：因视频文件占用空间太大，约 140Mb，本文档分有视频附件完整版和无视频附件的精简版，有视频附件的完整版直接双击 PDF 文档即可观看视频，无视频附件版提供链接下载。

### CHAPTERS IN THIS PART:

## PART II

# 进程课第 1 天

进程第 1 天课程摘要

### CHAPTERS IN THIS PART:

**1** 进程的代码结构 **3**    **2** 进程的状态特征 **6**

## 进程的代码结构

## 1.1 进程控制块 PCB 与 TASK\_STRUCT

进程是一个资源封装的单位，资源指占用的内存，文件系统，信号及处理方法。线程是调度执行的单元。一个进程区别与另一个进程的标记就是资源。linux 操作系统是可以做到进程与进程之间的资源隔离。进程的描述就是资源的描述。PCB（PROCESS CONTROL BLOCK）在不同操作系统中用于描述进程，在 Linux 的 PCB 就是用 `task_struct` 来描述。如图 1-1 所示，图中列出了主要对应包含的资源种类及作用。

进程控制块 PCB

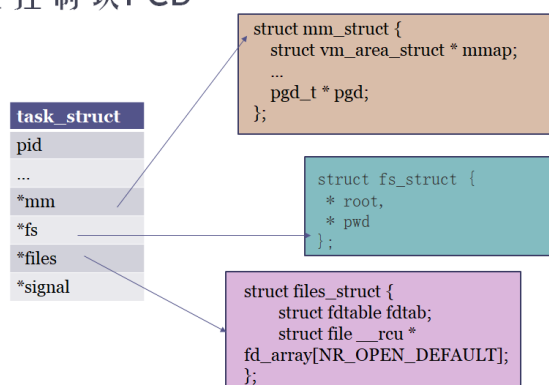


图 1-1

进程控制块 PCB

注：即 linux 中的  
`task_struct`

**mm 内存资源：**进程的内存

**fs 文件系统资源 1：**根路径和当前路径指针

**files 文件系统资源 2：**进程打开的文件，文件描述符数组

**signal 信号资源：**不同进程可以针对同一信号挂不同的处理方法

**pid 属性资源：** 描述进程的属性

## 1.2 TASK\_STRUCT 的属性特点

### fork 炸弹让 linux 死机

linux 下著名的 fork 炸弹，一敲就让 Linux 死机。是利用不断利用 fork 产生进程把 pid 耗尽，其命令如下：

应用

1-1

fork 炸弹

linux 下著名的 fork 炸弹，一敲就让 Linux 死机。

```
#linux fork 炸弹
:(){!:&};::
```

linux fork 炸弹解析

: 函数名为冒号

() 函数参数定义

{ } 函数定义

: 调用自己

|: 递归调用自己

& 后台执行

; 函数结束

: 调用函数:

### pid 数量限制导致安卓的一键 root

安卓的 2.2.1 之前的版本被发现一个漏洞，很容易就被一键 root，安卓的调试软件 adb 刚开始时有 root 权限，之后 adb 调用 `api setuid(shell)` 把自己从 root 用户降为 shell 用户。谷歌的工程师在调用时没有检查 `setuid` 的返回值，即默认 `setuid` 总是可以成功。黑客们利用 `uid` 数量有限制的属性，将 shell 用户内的 `pid` 进程全部用完，这样调用 `setuid` 时是无法成功的，但因为没有检查返回值，导致 adb 调用 `setuid(shell)` 后没有降权成功，还是有 root 权限。这就是 Android 著名的提权漏洞: `rageagainstthecage`。2.2 之后的安卓版本修复了此漏洞，方法是检查 `setuid` 的返回值。

查看 Linux 中最大 Pid 数量的命令如下：

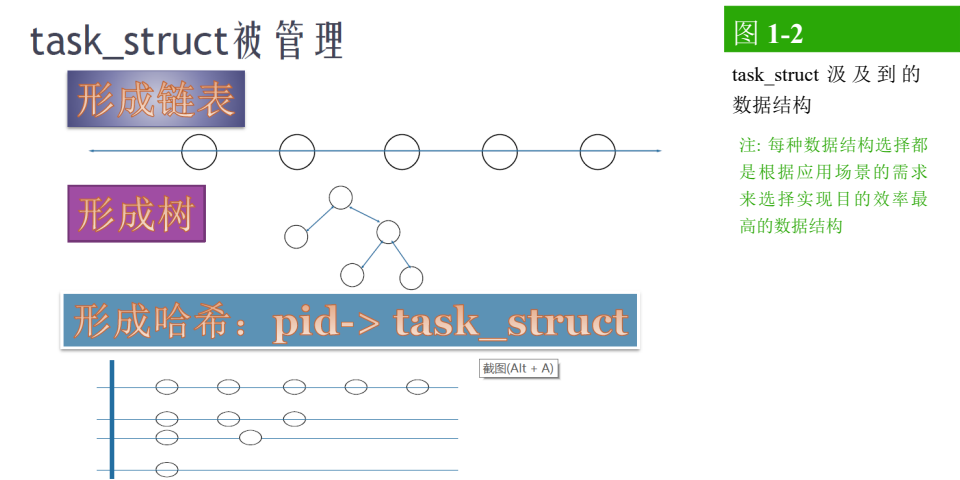
```
wangfan@wangfan-VirtualBox:~$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
```



scheduling priority	(-e) 0
file size	(blocks, -f) unlimited
pending signals	(-i) 15723
max locked memory	(kbytes, -l) 64
max memory size	(kbytes, -m) unlimited
open files	(-n) 1024
pipe size	(512 bytes, -p) 8
POSIX message queues	(bytes, -q) 819200
real-time priority	(-r) 0
stack size	(kbytes, -s) 8192
cpu time	(seconds, -t) unlimited
max user processes	(-u) 15723
virtual memory	(kbytes, -v) unlimited
file locks	(-x) unlimited

linux 进程 task\_struct 的三种数据结构

在 linux 代码中会涉及各种对 task\_struct 的引用关系，比如调度算法中会将 task\_struct 挂在链表上，父子进程的关系用树来描述，CFS 调度算法会用到红黑树，通过 pid 查找进程则是用 hash 表的结构。其对应的数据结构如图 1-2 所示



## CHAPTER

# 2

## 进程的状态特征

### 2.1 [进程状态切换](#)

### 2.2 [进程的内存泄露](#)

## PART III

# 进程课第 2 天

从

出生到死亡，进程资源是如何处理的？进程与进程间是怎样的关系？进程死亡后会不会内存泄露？不同生命周期下进程资源的处理方式有什么差异？

注：本章的架构是我根据讲课记录自己的理解划分的，可能与讲课不一致，如有错误欢迎指正。子死父收尸章节应该是第一天讲的，为了保持架构的一致性，把它挪到此处进行处理。

### CHAPTERS IN THIS PART:

- |   |      |    |   |      |    |
|---|------|----|---|------|----|
| 3 | 进程出生 | 8  | 5 | 进程死亡 | 12 |
| 4 | 进程运行 | 11 |   |      |    |

## CHAPTER

# 3

## 进程出生

### 3.1 进程出生时资源处理

### 3.2 进程分裂时的资源变化 – COW

父子进程刚诞生时所得到的资源是一样的，那这些资源在何时发生变化，以及变化后有什么影响，这就涉及到了 linux 的 copy-on-write 技术，下面通过具体的实例来说明父子进程的资源变化流程。

COW(copy-on-write) 技术是进程 fork 时采用的，涉及到虚拟内存和实际内存的映射关系。采用了 COW 技术后，进程处理会有一些现象需要重点注意。比如 fork 之后的父子进程读写同一个全局变量时，一个变量在不同的进程会显示出不同的值。

#### COW 现象代码

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int data = 10;
int child_process()
{
    printf("child process %d, data %d\n", getpid(), data);
    data = 20;
    printf("child process %d, data %d\n", getpid(), data);
    _exit(0);
}
int main(int argc, char * argv[])
{
    int pid;
    pid = fork();
```

```

if(pid == 0){
    child_process();
}
else{
    sleep(1);
    printf("parent process %d, data %d", getpid(),data);
    _exit(0);
}
return 0;
}

```

在正常情况下，程序修改全局变量 `data`，再打印 `data`，会是修改后的值 20，代码中子进程修改全局变量为 20 后，父进程等待 1s，确保子进程已修改完成，但父进程最终打印的结果还是 10。

COW 现象

```

# 运行程序的显示结果如下
child process 9491, data 10
child process 9491, data 20
parent process 9490, data 10

```

下面我们具体分析程序背后采用 COW 的原理和流程。

## COW 实现技术原理

fork 进程前后的内存关系如图 3-1 所示，

**fork 前第 1 阶段：** 全局变量 `data` 对应数据段内存 `vir` 和 `phy` 都在数据段，权限为可读可写。

**fork 后：** `vir` 和 `Phy` 的权限全部变成只读权限，读内存正常，写内存会进入 page fault 缺页中断。

**fork 后写内存：** 写内存后，发生缺页中，Linux 会重新申请一个 4k 内存，将新物理内存指向更改了内存地址的进程 `vir`。同时将老的 4k 内存拷贝给新的内存，同时将权限改为 R+W，这样父子进程的同一个 `vir` 虚拟地址就分别对应 2 个独立的、可读可写的物理地址。总之谁先写谁拿到新的物理内存，原内存留给剩下的进程。

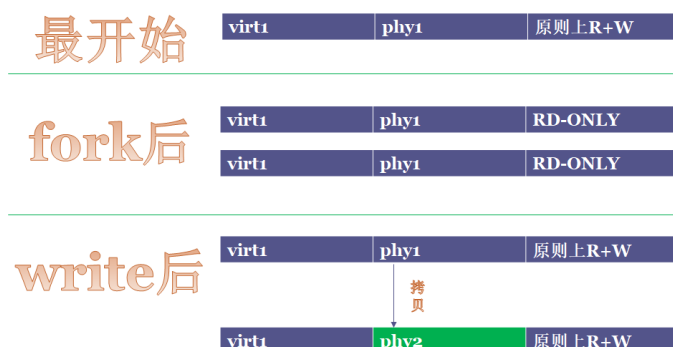


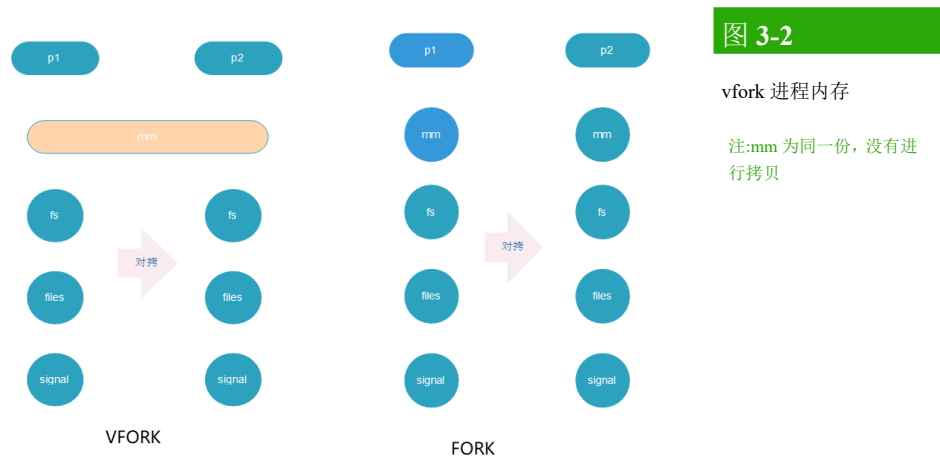
图 3-1

fork 进程前后内存映射关系

注：第一列为虚拟地址，第二列为物理地址，最后一列对应内存的读写权限

### 无法用 COW 的情况：VFORK

COW 技术必须借助 MMU（内存管理单元）来实现。COW 是通过改变虚拟内存和物理内存的映射关系来实现，没有 MMU 的系统，无法实现虚拟内存和物理内存的映射。也无法调用 fork 函数，无 MMU 系统对应调用的是 vfork 函数，其资源变化对比 fork 如图 3-2 所示：



vfork 的特点：vfork 会阻塞父进程，只有等子进程完全退出后才执行父进程。vfork 特性演示视频如下所示：

#### vfork 视频

此文档为不带视频附件的精简版本，请用带视频附件的 PDF，直接双击 PDF 中的视频链接观看视频文件

### 3.3 第 1 个进程，进程 0 与进程 1

## 进程运行

### 4.1 进程睡眠

### 4.2 进程等待

## CHAPTER

# 5

## 进程死亡

### 5.1 子死父收尸

### 5.2 父死子托孤



## PART IV

# 进程课第 3 天

类

型不同，策略不同。进程第 3 天课程摘要

本章讲述如何根据不同类型的进程分配不同策略的调度算法，调度算法的原理及应用对象。还有如何更改进程的调度策略

### CHAPTERS IN THIS PART:

- |   |        |    |   |       |    |
|---|--------|----|---|-------|----|
| 6 | 进程分类   | 14 | 8 | 调整优先级 | 16 |
| 7 | 进程调度策略 | 15 |   |       |    |

## CHAPTER

# 6

## 进程分类

### 6.1 CPU 消耗与 IO 消耗型

应用

6-1

采用大核 + 小核的设计，大核功耗高，运算力强，用于处理 CPU 消耗性任务，小核功耗低，功耗小，用于处理 I/O 消耗性任务。实现功耗降低，但处理效果与全是大核处理一致的效果

ARM 的 big.LITTLE  
设计

## 进程调度策略

### 7.1 RT 进程调度

**SCHED\_FIFO**

**SCHED\_RR**

### 7.2 NORMAL 进程调度

**CFS 调度**

## CHAPTER

# 8

## 调整优先级

8.1 用 **RENICE** 改变进程优先级

8.2 用 **NICE** 改变进程优先级

8.3 用 **CHRT** 改变进程优先级

# 进程课第 4 天

负

什么

负载均衡 这里讲的是 CPU 处理的负载，对应负载指 CPU 处理的任务

1. linux 的 4 种不同优先级任务是

1. 实时系统是什么？

2. linux 为什么不是一个硬实时系统？

3. linux RT 实时补丁的原理，使用方法及限制。

2. 不同优先级任务间的抢占原则是什么？

3. 负载均衡的不同类型及使用方法。

## CHAPTERS IN THIS PART:

9 负载均衡 18

10 实时系统 23

## CHAPTER

# 9

## 负载均衡

### 9.1 LINUX 下的负载均衡处理对象

负载均衡是最大化利用 CPU 资源的方法，要求在有任务 (task\_struct，中断，软中断) 执行时，所有的 CPU 都能利用上，不产生有任务处理却有 CPU 闲置的情况。首先从任务的优先级的角度来看，CPU 处理的任务只有下面 4 种优先级，按高到低依次是：

表 9-1	优先级	Linux 中 CPU 所处状态
Linux CPU 对应的 4 类不同优先级区间	1	中断
	2	软中断
	3	处于 spinlock 等关闭了调度区间的进程
	4	普通进程

附注：优先级数字越低优先级越高：中断 > 软中断 > spinlock > 普通进程

### 9.2 中断负载均衡

在 TOP 命令中，cpu 时间占用中有一列是 **hi** 和 **si**，分别对应中断和软中断。说明 cpu 时间除了在 task\_struct 上，还有可能花在中断和软中断，当网络流量比较大时，cpu 花在中断和软中断的时间比较大，可以考虑中断负载均衡。

分配 IRQ 到某个 CPU，掩码 01 代表 CPU0，02 代表 CPU1，04 代表 CPU2，08 代表 CPU3

以上优先级的任务在 linux 处理规则如下：

1. 中断不可以嵌套中断，在 2.6 版本后，处于中断区间再次发生中断时，会等到前一个中断执行结束后再进行处理下一个中断。

2. 中断可以唤起软中断

3. 软中断可以唤起中断

4. CFS 等调度算法只处理普通进程和普通进程之间的调度，不涉及中断，软中断，及关闭了调度的进程。具体表现如下：

- 如果 CPU 在处理 1, 2, 3 优先级的任务时，不受调度算法的调度，只有等处理完 1, 2, 3 优先级的任务后才会再由调度算法调度。

- 如果 CPU 在处理 4 普通进程的任务时，高优先级的中断和软中断可以直接抢占普通进程，不用等调度算法调度。

中断分配到 CPU 方法

```
# 此命令将中断 145 分配到 CPU0 上处理
[root@boss ~] # echo 01 > /proc/irq/145/smp_affinity
[root@boss ~] # cat /proc/irq/145/smp_affinity
00000001
```

### 9.3 软中断负载均衡-RPS

有时候有的网卡只有一个队列，一个队列的中断只能分配到一个核，Linux 设计是一个核上抛出的软中断只能在同一个核上执行，cpu 0 上的中断抛出一个软中断，tcp/ip 协议栈也只能在 cpu 0 的软中断上处理。google 在 linux 内核里面加入了 rps 补丁，其作用是尽管中断是在一个 cpu 核上，但 tcp/ip 协议处理的软中断可以在多个核上进行处理。rps 的原理是收到软中断后通过核间中断的方法给另外的核发中断，让其他核处理软中断，从而支持单队列情况下的网络负载均衡。

rps 使能方法

```
#rps 使能方法，除了 CPU 0 外都参与 TCP/IP 协议栈
echo ffe > /sys/class/net/eth1/queues/rx-0/rps_cpus
```

```
# 查看 softirqs
```

```
wangfan@wangfan-VirtualBox:~$ cat /proc/softirqs
```

	CPU0	CPU1
HI:	0	2
TIMER:	6841572	6725135
NET_TX:	1	17644
NET_RX:	679	224896
BLOCK:	61380	180153
IRQ_POLL:	0	0
TASKLET:	15	7834
SCHED:	3148547	3016778
HRTIMER:	0	0
RCU:	747890	885505

## 应用

## 9-1

利用 rps 解决 cpu 占用率高的问题

宋老师关于爱立信工程师的问题处理，爱立信的工程师在服务器上写了个软件发现 16 核有 2 个核占用率很高，但其他核都很闲，top 命令查看发现 hi 和 si 很高，说明 cpu 大部分时间在处理中断和软中断，而不是处理 task\_struct。解决方法是登录机器后敲命令 echo ffff 到 rps，cpu 占用率降了下来，效果很明显。

## 9.4 进程间 (TASK\_STRUCT) 负载均衡

### linux 负载均衡算法原则

linux 下所有 CPU 核会进行分布式的 PUSH 和 PULL 操作，当 CPU 核空闲时会向周围的核 PULL 任务来执行，CPU 核本身在执行任务时也会 PUSH 任务到其他核。每个核执行同样的负载均衡算法，负载均衡包括 RT 任务的负载均衡和普通任务的负载均衡。

RT 任务的负载均衡算法是将 N 个优先级最高的 RT 分布到 N 个核。

pull\_rt\_task(); push\_rt\_task()

普通任务负载均衡有三种：IDLE 式负载均衡，周期性负载均衡，FORK 和 EXEC 式负载均衡

**1 周期性负载均衡：**时钟 tick 的时间点上 CPU 核查询自己是否很闲，周围核是否很忙，是则用 PULL 将周围核的任务拉过来处理。

**2 IDLE 式负载均衡：**当 CPU 核在 IDLE 时会查询周围核是否在忙，如果旁边核比较忙时，自动 PULL 旁边核的 task\_struct 任务来执行。

**3 FORK 和 EXEC 式负载均衡：**FORK 和 EXEC 创建一个新的进程时，Linux 会自动找一个最闲的核将 FORK 和 EXEC 新创建出的进程放在上面处理。

以上处理由核与核之间分布式负载均衡处理是自动进行的。

## 应用

## 9-2

cpu 占用率 200% 的原因

在一个程序中起 2 个进程，每个进程都在做 CPU 消耗型操作（代码中调用 while(1) 死循环），在进程执行的过程中会自动将进程分配到 2 个 CPU 核上进行处理。可以通过查看 CPU 占用率和时间占用情况来验证。分配到两个 CPU 核后，CPU 占用率会上升到 200%，用 time 计算程序的占用时间，真实时间是系统时间的一半，因为系统时间是单独统计每个 CPU 核上占用的时间，2 个 CPU 核上会统计 2 次，显示的结果就是系统时间是真实时间的 2 倍。

### 设置进程在指定 CPU 上运行

要设置进程在指定 CPU 上运行，在代码里可以通过调用相关 API 实现，也可以直接在 BASH 中通过 taskset 命令实现。

```
//设置CPU task affinity api
#include<pthread.h> //注意<pthread.h>包含<sched.h>
int pthread_setaffinity_np(pthread_t thread,size_t cpusetsize,const cpu_set_t *cpuset);
int pthread_getaffinity_np(pthread_t thread,size_t cpusetsize, cpu_set_t *cpuset);
```



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

# taskset在bash下的使用方法

# 命令行形式设置CPU亲和性

```
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的亲和性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CPU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亲和性

```
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.

## 给进程指定比例的 CPU 负载-cgroup

当前的程序是按程序的需要来占用 cpu 的，这样可能会出现一些问题，比如用户 A 和 B 在同一个服务器上，如果 A 开的线程比 B 多，可能导致 A 一直占用 cpu，B 因为线程少，占用的权重比例少而得不到 cpu。于是我们想要一个分群的概率，让 A 和 B 各占有 50% 的 CPU，不管 A 线程有多少，最多只能占 50% 的 CPU，这样保证 B 即使线程数量少，也可以得到足够的 CPU 来运行。同样的道理类似于计费的网络带宽，可以根据用户缴费的情况分配 CPU，如果未交费，就算 CPU 空闲也不分配 CPU 给用户。

**cgroup** 使用方法 cgroup 主要是设置以下 3 个属性，在 `/sys/fs/cgroup/cpu` 目录 `mkdir` 一个 group 后，会出现很多属性文件，我们主要通过以下属性来查询和设置。

**cgroup.procs:** 将进程号 echo 进去

**cpu.cfs\_period\_us:** 默认是 100000 基准时间 100ms

**cpu.cfs\_quota\_us:** 配额默认是-1 最大值，设置可以比 100000 大，它与 period 的比例表示 group 内线程最高可占 cpu 的比例

**cpu.shares:** 权重，默认是 1024，调节 cpu.shares 可以调节不同 group 的 cpu 占用率

—— cgroup 操作方法 ——

```
# 1 cd /sys/fs/cgroup/cpu 目录创建不同的 group
# 2 mkdir A 创建 group A
# 3 mkdir B 创建 group B
# 4 /sys/fs/cgroup/cpu/A echo 3582 > cgroup.procs 将进程 3582 加入 A group
# 5 /sys/fs/cgroup/cpu/B echo 3581 > cgroup.procs 将进程 3581 加入 B group
# 6 /sys/fs/cgroup/cpu/A echo 50000 > cpu.cfs_quota_us
    设置 A group 权重为 50% cpu, A 内线程的 cpu 占用率最高不超过 50%
```

### 应用

9-3

安卓 5.0 之后的版本用到了 cgroup，安卓早期版本所有进程都采用调度算法公平调度，最新版本把进程分为前台交互进程和后台非交互进程，前台的权重是 1024，后台的权重是 52，这样前台可以得到更多的 CPU，用于提高前台程序的响应。

安卓的 cgroup 设计

### 10.1 REAL TIME 实时系统的含义

实时不是越快越好，是指可预期性。比如发射导弹时，必须保证在截止期限内发射出去，否则后果可能是灾难性的。硬实时强调在恶劣的情况下，从唤醒到任务真正执行之间的时间是可预期的，可以保证在预期的时间内执行到。Linux 设计不保证是可预期的，因为 Linux 在中断，软中断，及 spinlock 的区间时是不可抢占的，这些区间内的执行时间是不可预期的。linux 不是硬实时的系统，是软实时。

### 10.2 抢占：LINUX 无法硬实时的原因

要实现实时，最重要的就是要实现任何时刻都可以在预定期限内实现进程抢占，来保证进程可以在预期内执行。Linux 无法硬实时的原因就是 Linux 的 CPU 在的 4 类区间有 3 类是无法进行抢占的，如果 CPU 进入了这 3 类区间，Linux 是无法保证在这 3 类区间内的运行时间的。所以无法达到硬实时的要求。Linux 抢占的时机很多，我们主要记住不可抢占的时机：1、中断，2、软中断，3、spinlock 区间的进程。如果在这三类任务时间内唤起了高优先级的任务，任务是不能抢占的，只有当 CPU 脱离了这三类区间后，高优先级的任务才可以进行抢占。如表 9-1 所示，如果在一二三类区间唤醒了高优先级的进程，当前无法进行抢占，只有当 CPU 退回到第 4 类普通进程区间时才开始抢占。

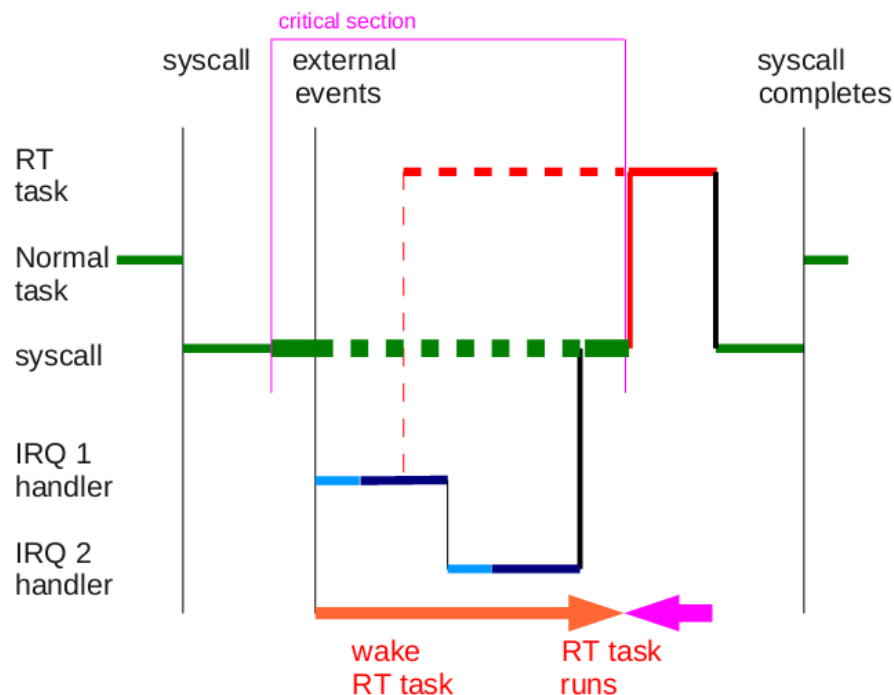


图 10-1

Linux 的 4 类抢占区间

### 10.3 LINUX 实时补丁的用法

linux 的 RT 版本, <https://wiki.linuxfoundation.org/realtime/start> 这个项目是实现 Linux 的实时版本。主要原理是将 Linux 的中断线程化, 即把一二三类区间都转换成四类区间。相应的 linux 源码 RT 版本只针对特定的几个版本, 使用方法是代码中的 RT 补丁 merge 到对应的源码中, linux 的 RT 可以做到 100us 量级的实时性能, (vxworks 是几个 us), 相应的吞吐性能也会下降。

安装 linux 的调度器的抢占模型选项

1. server 版本 (不抢占)
2. desktop 版本 (kernel 内不抢占)
3. no-latency desktop (kernel 内可抢占, 手机桌面一般用此模型)
4. completely preemption (kernel 内一二三类中断软中断都可抢占)

安装 RT 补丁后还是会有很多的代码坑需要处理, 比如内存管理方面, Linux 的内存分配是 LAZY 式, 当用到时才实际分配, 安装实时补丁后有可能出现写内存时内存实际还未分配的情形。

应用  
10-1  
的实时系统应

实时系统的另一个应用场景是安装 2 个系统：实时系统和 Linux，比如单反之前用的都是实时操作系统，现在为了增加蓝牙，wifi 等功能，用一个核执行实时系统，另一个核用 Linux 来实现蓝牙等功能

## PART VI

# 进程问题集锦

**F** AQ, 本章记录课间和课后宋老师以及同学们答疑

### CHAPTERS IN THIS PART:

**11** 课后答疑 27

## 课后答疑

以下问题为宋老师在微信答疑群中回答记录，微信群一直存在，问题也不定期提出，此部分内容会随之更新。

Q: rps 后，非多队列网卡，中断会再给每个核发中断来派发软中断？

A: 中断只发一个 core，这个 core 自己给别的 core 发核间中断

1.

Q: rt 补丁，是不是只有 RT\_FULL 支持优先级反转？

A: 不是，不需要 rt 补丁就支持优先级继承，早就 Merge 到了 mainline。不叫支持优先级反转，反转是个问题，继承是解决它的方法。反转是个现象，不存在支持不支持。你的问题是错误的。

2.

Q: 所以 softirq 的优先级都是相同的吗？

A: 不是的，它是一个 bit 的设置，检查哪个 bit 被设置，肯定是有先后顺序。挨个检查哪个 bit 的。不过这个不是关键点。你关心延迟的实时性的时候，你根本就消灭了 softirq。

3.

## PART VII

# 参考资料



考这章列举了用到的相关资料源地址

参考的文件以 PDF 附件的形式，可以双击链接打开或保存，需选择支持 PDF 附件的 PDF 阅读器，建议使用 adobe 的阅读器打开附件

### CHAPTERS IN THIS PART:

**12** 参考文献 29

**13** 相关附件 30



### 12.1 宋宝华相关网站资源

1. CSDN 视频课程打通 Linux 脉络系列：进程、线程和调度
2. linux 公众号：Linux 阅码场

### 12.2 代码网址

## CHAPTER

# 13

## 相关附件

### 13.1 [PDF 课件](#)

4 天课程的 PPT 讲义，双击打开附件或在附件中另存为处理。

1. [第 1 天进程讲义 PDF](#)
2. [第 2 天进程讲义 PDF](#)
3. [第 3 天进程讲义 PDF](#)
4. [第 4 天进程讲义 PDF](#)

### 13.2 [视频文件](#)

此文档为不带视频附件的精简版本，请用带视频附件的 PDF，直接双击 PDF 中的视频链接观看视频文件