



wangfan 的 Linux 进程笔记-精简版

L^AT_EX Web Edition

linux 进程管理 – 宋宝华

wangfanstar@163.com

2018 年 6 月 1 日

目录大纲



目 录

PART I

前言

起

因：文档是我在学习宋宝华老师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，这边用的是 ADOBE 的 PDF 阅读器打开没有问题。

文档相关：文档采用 LaTeX 在网上找了个模板修改编译而成，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 1 天

进程第 1 天课程摘要

CHAPTERS IN THIS PART:

进程的代码结构

1.1 进程控制块 **PCB** 与 **TASK_STRUCT**

1.2 **TASK_STRUCT** 的属性特点

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 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？为什么进程死亡后不可能内存泄露

CHAPTERS IN THIS PART:

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

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

本单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:

CHAPTER

6

进程分类

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

负载均衡 进程第 4 天课程摘要这里讲的是 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什么
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。

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。

CHAPTERS IN THIS PART:

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：

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

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 fffe > /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

由于负载均衡导致 `cpu` 占用率 200%

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

CHAPTER

实时系统

10

PART VI

进程问题集锦

F AQ, 本章记录课间和课后宋老师与我们之间的问题
进程问题集锦

CHAPTERS IN THIS PART:



CHAPTER

11

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:

12.1 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版，请用带视频附件版的 PDF，直接双击 PDF 中的视频链接观看视频文件。