3.2 CFS 调度器

在阅读本章之前请思考关于 Linux 内核普通进程调度的几个小问题：

1 *请简述你对进程调度器的理解，早期 Linux 内核调度器包括 O(N)和 O(1)调度器是如何工作的？*
2 *优先级、nice 以及权重之间有啥关系？*
3 *请简述 CFS 调度器是如何工作的？*
4 *CFS 调度器中 vruntime 是如何计算的？*
5 *vruntime 是什么时候更新的？*
6 *CFS 调度器里 nim_vruntime 有什么作用？*
7 *CFS 调度器对新创建的进程以及刚唤醒的进程有何关照？*
8 *普通进程的平均负载 load_avg_contrib 是如何计算的？runnable_avg_sum 和 runnable_avg_period 是什么含义？*
9 *内核代码中定义了好几个表，请说出它们的含义。prio_to_weight、prio_to_wmult、runnable_avg_yN_inv、runnable_avg_yN_sum。*
10 *如果一个普通进程在就绪队列里等待了很长时间没被调度，那么它的平均负载又该如何计算？*

Linux 内核作为一个通用操作系统，需要兼顾各种各样类型的进程，包括实时进程、交互式进程、批处理进程等。每种类型进程都有其特别的行为特征。

- 交互式进程：与人机交互的进程，与鼠标、键盘、触摸屏等相关的应用比如 vim 编辑器，他们一直在睡眠等待用户召唤它们。这类进程的特点是希望系统相应时间快，否则用户就会抱怨卡顿。
- 批处理进程：此类进程默默工作和付出，可能会占用比较多的系统资源，比如编译代码等。
- 实时进程：有些应用对整个延时有严格要求，比如现在很火的 VR 设备，从头部转动到视频显示需要控制到 20ms 以内否则人会出现眩晕。在比如有些工业控制系统，严重的延迟会导致严重的事故发生。

不同的进程采用不同的调度策略，目前 Linux 内核中默认实现了 4 种调度策略，分别是 deadline、realtime、CFS 和 idle，它们分别使用 struct sched_class 来定义成调度类。

本章主要是讲述普通进程的调度，包括交互进程和批处理进程等。在 CFS 调度器出现之前，早期 Linux 内核中曾经出现过两个调度器，分别是 O(N)和 O(1)调度器。O(N)调度器就是 Linus 在 1992 年发布的那款调度器。该调度器算法比较简洁，从就绪队列中所有进程的优先级进行比较，然后选择一个最高优先级的进程作为下一个调度进程。每个进程有一个固定时间片，当进程时间片使用完之后调度器会选择下一个调度进程，当所有进程都运行一遍之后再重新分配时间片。这个调度器的选择下一个调度进程需要遍历就绪队列，花费 O(N)时间。

在 Linux2.6.23 内核之前有一款名为 O(1)的调度器，优化了选择下一个进程的时间。它为每个 CPU 维护一组进程优先级队列，每个优先级一个队列，这样在选择下一个进程的时候，只需要查询优先级队列相应的位图就可以知道那个队列上有就绪进程，所以这个查询时间为常数 O(1)。

O(1)调度器在处理某些交互式进程的时候依然有不少问题，所以 Linux 内核社区的一位传奇

人物 Con Kolivas[①]提出了 RSDL（楼梯调度算法）来实现公平性，在社区的一番争斗之后，红帽子的 Ingo Molnar 在借鉴 RSDL 的思想憋出一个 CFS 调度算法。

这 4 种调度类通过 next 指针串联在一起。用户空间程序可以使用调度策略 API 函数（sched_setscheduler()[②]）来设定用户进程的调度策略。其中 SCHED_NORMAL 和 SCHED_BATCH 使用 CFS 调度器，SCHED_FIFO 和 SCHED_RR 使用 realtime 调度器，SCHED_IDLE 指的是 idle 调度，SCHED_DEADLINE 是 deadline 调度器。

[include/uapi/linux/sched.h]

```
/*
 * Scheduling policies
 */
#define SCHED_NORMAL            0
#define SCHED_FIFO              1
#define SCHED_RR                2
#define SCHED_BATCH             3
/* SCHED_ISO: reserved but not implemented yet */
#define SCHED_IDLE              5
#define SCHED_DEADLINE          6
```

1 权重计算

内核使用[0~139]的数值来表示进程的优先级，数值越低优先级越高。优先级 0~99 是给实时进程使用，100~139 是给普通进程使用。另外在用户空间有一个传统的变量 nice 值来映射到普通进程的优先级即[100~139]。

进程 PCB 描述符 struct task_struct 数据结构中有 3 个成员来描述进程的优先级。

```
struct task_struct {
    ...
    int prio;
    int static_prio;
    int normal_prio;
    unsigned int rt_priority;
    ...
};
```

static_prio 是静态优先级，在进程启动的时候分配的优先级，用户可以通过 nice 或者 sched_setscheduler 等系统调用来修改该值。normal_prio 是普通优先级，基于 static_prio 和调度策略计算出来的优先级，进程在创建时会继承父进程的普通优先级。prio 保存着进程的动态优先级。rt_priority 是实时进程的优先级。

内核使用一个 struct load_weight 数据结构来记录进程调度实体的权重（weight）。

```
struct load_weight {
      unsigned long weight;
      u32 inv_weight;
```


① Con Kolivas 是内核传奇的开发者，他主业是麻醉师，在内核社区中一直关注用户体验的提升，设计了相当不错的调度器算法，最终没有被社区采纳。后来他有设计了一款名为 BFS 的调度器。

② sched_setscheduler()，sched_getscheduler()– 用户空间程序系统调用 API 用来设置和获取内核调度器的调度策略和参数。