**进程管理**

Linux分时 把CPU的运行分成时间片处理不同的运算请求

3.2 CFS调度器

(1)交互进程：与人机交互的进程

(2)批处理进程：此类进程默默地工作和付出，可能会占用比较多的系统资源

(3)实时进程：有些应用对整体时延有严格要求 ，

Linux早期曾经出现两个调度器，分别是o(N)和O(1)调度器发生于，该调度器算法比较简洁。从就绪队列中比较所有进程的优先级，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的时间片，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篇 再重新分配时间片。

Linux 2.5.23 内核o(1)的调度器，优化选择下一个进程的时间，它为每个CPU维护一级进程优先级队列，每个优先级一个队列，这样在选择下一进程时，只需要查询优先级队列相应的位图，即可知道哪个队列中有就绪里程。

目前Linux内核中默认实现了**4种调度策略**，分别deadline,realtime,CFS和idle，它们分别使用strcut shced\_class 来定义调度类。

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 () 来设定用户进程的调度策略，其中SCHED\_NORMAL 0，SCHED\_BATCH 3 使用CFS调度器，SCHED\_FIFO 1，SCHED\_RR 使用realteim调度器，SCHED\_IDLE使用idle调度，SCHED\_DEADLINE使用deadline调度器

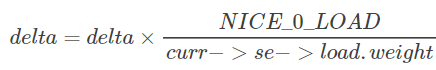
[\kernel\msm-4.4\kernel\sched\core.c]

权重计算

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

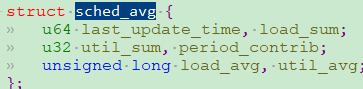
进程PCB描述符struct task\_strcut数据结构中有3个成员描述进程优先级。

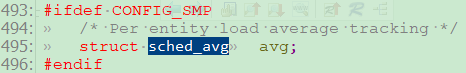
内核使用struct load\_weight数据结构来记录调度实体的权重信息。



CPU的负载，最简单的方法是计算cpu上就绪队列上所有进程的权重。仅考虑优先级权重是有问题。因为没有考虑该进程的行为。有的进程使用的cpu的突发性的。有的是恒定的。有的是cpu密集型，也有的IO密集型。进程调度考虑先级级权重的方法可行，但是如果延伸到多cpu之间的负载均衡就显得不准确。

因此从linux 3.8 以后进程，不仅考虑权重，而跟踪每个调度实体的负载情况。该 方法PELT(pre-entity load tracking )

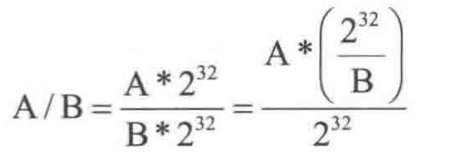




表示runnable\_sum 表示该调度实体在就绪队列里(se->on\_rq=1) 可运行状态(runable)的总时间。 就绪队列中时间包括两部分，一是正在运行的时间，称为running时间，二是就绪队列中等待的时间。

表示runnable\_period 可以理解为该调度实体在系统中总时间，之所以称为period是为因为1024 us为一周期period,last\_runnable\_update用于计算时间间隔。runable\_sum 是指统计在就绪队列里的总时间，进程进入就绪队列时(调用enqueue\_entity()), on\_rq会设置1，

内核定义了表示 runnable\_avg\_yN\_inv[]来方便使用衰减因子。



其中除以 可以使用右移32位来计算。Runable\_avg\_yN\_inv[]相当于提前计算公式中的值。Runable\_avg\_yN\_inv[]表示包括32个下标，对应过去0-32ms的负载贡献的衰减因子。

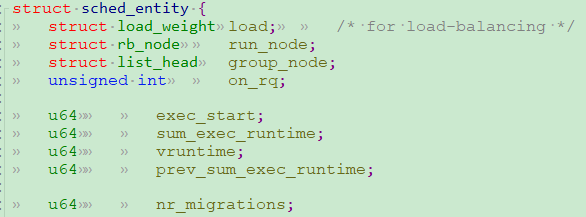
内核中的函数来计算第n个周期的衰减值。参数val表示第n个周期前的负载值，n表示第n个周期，其计算公式，即第n个周。

**3.2.2进程创建**

进程的创建通过do\_fork()函数来完成，do\_fork()来执行过程中就参与了进程调度相关的初始化。进程调度有一个非重要的数据结构struct sched\_entity ,称为调度实体，该数据结构描述进程作为一个调度实体参与调度所需要的所有信息。

例如load 表示该调度实体的权重，run\_node 表示该调度实体在红黑树中的节点，on\_rq表示该调度实体是否在就绪队列中接受调度，vruntime表示虚拟运行时间，sum\_exec\_runtime，sum\_exec\_runtime，prev\_sum\_exec\_runtime，计算虚拟时间需要所需要的信息。avg 表示 该调度实体的负载信息

[include/linux/sched.h]



\_\_sched\_fork() 函数会把新创建进程调度实体se相关成员初始化为0，因为这些值不能复用父进程，子进程将来要加入调度器中参与调度，

[do\_fork() ->sched\_fork->\_\_sched\_fork()]



[do\_fork() ->sched\_fork->

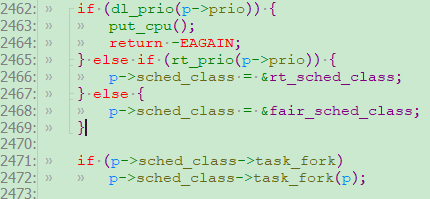


继续查看sched\_fork()函数，设置子进程状态task\_running，这里不是真正开始运行。因为没有添加调度器。

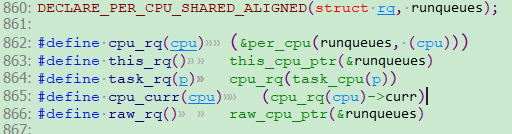
每个调度类都定义一套操作方法集。调用CFS调度器的task\_fork方法做一些fork相关初始化。CFS调度器调度类定义的操作方法集如下:

[kernel/sched/fair.c]

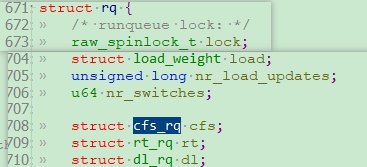
task\_fork方法实现在kernel/fairc文件中。



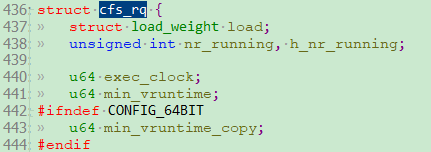
函数task\_fork\_fair()函数的参数p表示新创建的进程。进程task\_struct 数据结构中内嵌了调用实体struct sched\_entity结构体，因此由task\_struct 可以得到该进程的调度实体。smp\_processor\_id() 从当前进程thread\_info结构中的cpu成员获取当前cpu id .系统中每个cpu有一个就绪队列(runqueue).它是per-cpu类型。即每个cpu 有一个struct rq数据结构，this\_rq 可以获限当前cpu 的就绪队列数据结构struct rq ;



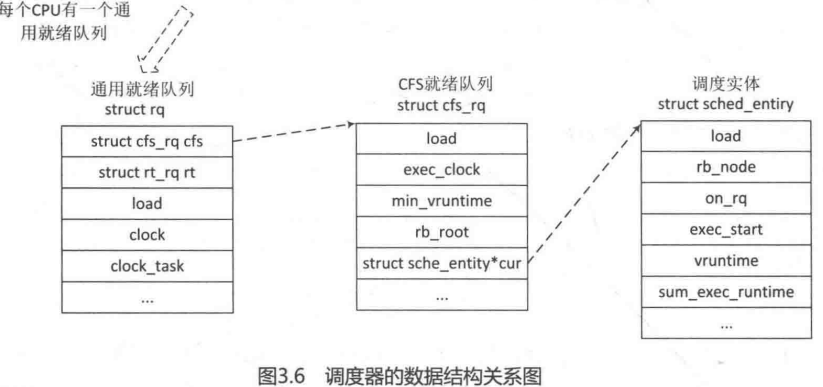
struct rq 数据结构是描述cpu 的通用就绪队列，rq数据结构中记录了一个就绪队列所需要全部信息。包括一个cfs调度器就绪队列数据结构struct cfs\_rq,一个实时进程调度就绪队列数据结构struct rt\_rq和一个deadlin调度器就绪队列数据结构struct dl\_rq,以及就绪队列的权重load等信息。



struct cfs\_rq是CFS调度器就绪队列的数据结构，定义如下



内核中调度器相关数据结构的关系如图，看起来很复杂，其实它们关闻的



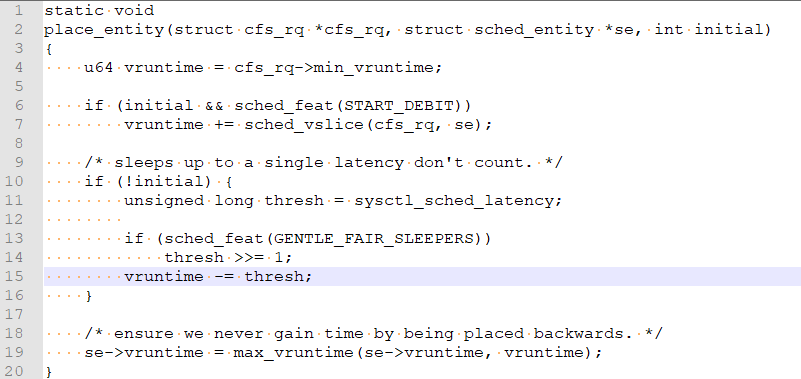


task\_cfs\_rq函数可以取出当前进程对应的cfs就绪队列。

\_\_set\_task\_cpu 把当前CPU绑定到该进程中，p->wake\_cpu在后续唤醒该进程 会用这个成员

 updae\_curr 函数是CFS调度器中比较核心的函数。 参数是当前进程对应CFS就绪队列。curr指针指向的调度实体当前进程，即父进程。rq\_clock\_task获取当前就绪队列保存的clock\_task值。该变量在每次时钟滴答tick到来时更新，delta\_exec计算该进程从上次调用update\_currr()函数到现在的时间差，calc\_dalta\_fair()使用delta\_exec时间差来计算进程虚拟时间vruntime.



place\_entity()参数cfs\_rq指父进程对应的CFS就绪队列，se是新进程的调度实体。Initial值为1.每个cfs\_rq就绪队列中有一个成员min\_vruntime.min\_vruntime其实是单步递增的。用于跟踪事个cfs就绪队列中红黑树里最小vruntime值。如果是当前进程用于fork新进程。那么这里会对新进程的vruntime做一些惩罚，因为新创建了一个进程导致CFS运行队列的权重发生了变化。惩罚值通过sched\_vslice()函数来计算。



首先\_\_sched\_period 函数会计算就绪队列中一个调度周期的长度。可以理解为一个调度周期的时间片。它根据当前运行的进程数目来计算。

函数sched\_vslice()根据 计算得到的时间来计算可以得到多少虚拟时间。

**3.2.3 进程调度**

**\_\_schedule()函数是**调度器的核心函数，其作用是让调度器选择和切换到一个合适的进程运行。调度的时机可以分为如下3种

(1)阻塞操作：互斥量，信号量，等待队列等

(2).在中断返回前和系统调用返回用户空间时，去检查TIF\_NEED\_RESCHED标志位以判断 是否需要调度

(3).将要被唤醒的进程 wakeups 不会马上调用schedule()要求被调度，而是会被添加至CFS就绪队列中，并且设置TIF\_NEED\_RESCHED 标准位，那么唤醒进程什么时候被调度呢？这要根据内核是否具有可抢占功能(CONFIG\_PREEMPT=Y)分两种情况

如果内核可以抢占，则：

* 如果唤醒动作发生在系统调用或者异常处理上下文中，在下一次调用preempt\_enable()时会检查是否需要抢占调度；
* 如果唤醒动作发生在硬中断处理上下文中，硬件中断处理返回前夕会检查是否要抢点当前进程。

如果内核不可抢占，则：

* 当前进程调用 cond\_resched()会检查是否要调度
* 主动调用调用schedule()
* 系统调用或者异常处理返回用户空间时
* 中断处理完成返回用户空间时。

前文提到的硬件中断返回前夕和硬件中断返回用户空间前夕是两个不同的概念。



**函数**\_\_schedule()函数调用pick\_next\_task()让进程调度器从就绪队列中选择一个最适合的进程next。然后context\_switch 切换到next进程运行。

prev 指当前进程。thread\_info 数据结构中的preempt\_count成员用于判断当前进程是否可以抢占，preempt\_count的低8位用于抢占引用计数(preemption count),除此之外，还有一个比待位用于PREEMPT\_ACTIVE，它只有内核抢占调度中会被置位。

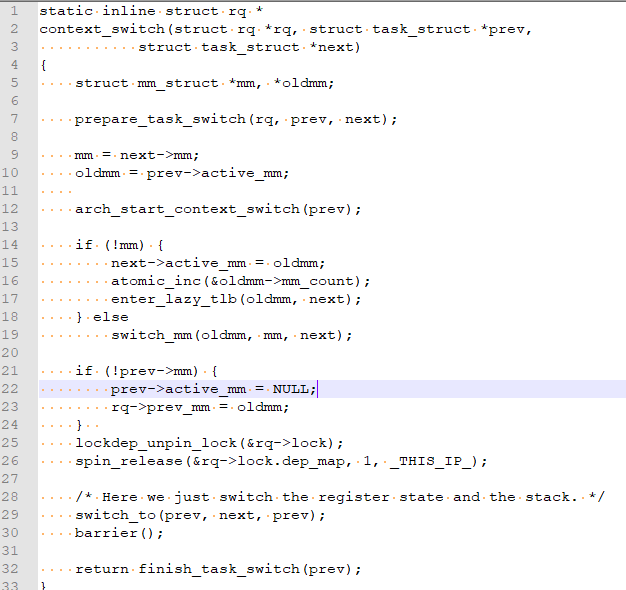
如果当前进程在之前发生过抢占调度preempt\_schedule(),那么在preempt\_scheule()->\_\_schedule()时它不应该被清除出运行队列。为什么这里做这样判断 呢？下面以睡眠等待函数wait\_event()为例，当前进程调用wait\_event函数，当条件condition 不满足时，就会把当前进程加入睡眠等待队列wq中，然后schedule()调度其他进程直到满足condition. wait\_event 函数等价于如下代码片段。

pick\_nex\_task()调用调度类中的pick\_next\_task()方法， 第X 行代码中有一个小的优先化，如果当前进程prec的调用类是CFS,并且该cpu整个就绪队列rq中的过程数量等于cfs就绪队列中数量；否则需要遍历整个调度类。调度类的优先级为stop\_sched\_class > dl\_sched\_class >rt\_shced\_class > fair\_sched\_class > idle\_sched\_class. stop\_sched\_class类似于关闭cpu,接下来是dl\_sched\_class 和rt\_sched\_class类。它是实时性进程，所以当系统有实时进程时，它们总是优先执行。

接下来看进程是如何切换。这部分涉及ARM体系结构

[\_\_schedule()->context\_switch()]





该函数涉及3个参数，其中rq表示进程切换所有的就绪队列，prev指将要被换出的进程，next 指将要被换入执行的进程。

prepare\_task\_switch(rq, prev, next); 函数设置next进程的task\_struct 结构中on\_cpu成员为1.表示next 进程马上进入执行状态。

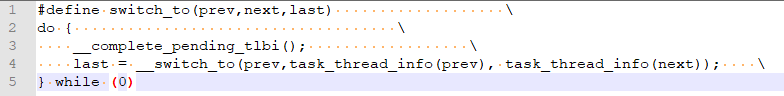
switch\_mm()首先把当前CPU设置到下个进程的cpumask位图中，然后调用check\_and\_switch\_context()函数来完成ARM体系结构相关的硬件设置。例如flush TLB.

在运行进程时，除了cache会缓存进程的数据外，cpu内部还有一个叫作TLB 硬件单元。它为了加快虚拟地址到物理的转换速度而将部分的页表项内容缓存起来，避免频繁的访问页表。

如果提高TLB的性能。从Linux 内核角度来看，地址空间可以划分为内核地址空间和用户空间。对于TLB来可以分成gobal 和process –specific

* Gobal 类型的TLB：内核空间是所有进程共享的空间。因此这部分空间的虚拟地址到物理的翻译是不会变化。可以理解为Global的。
* Process-specific类型TLB：用户地址空间是每个进程独立的地址空间。Prev进程切换到next进程时，TLB中缓存的prev进程相关数据对于next进程是无用的。因此可以冲刷掉，这就是所谓的process – specific 的TLB。

处理完TLB和页表基地址后，还需要进行栈空间的切换。Next 进程才能开始运行。下面来看 context\_switch()->switch\_to()

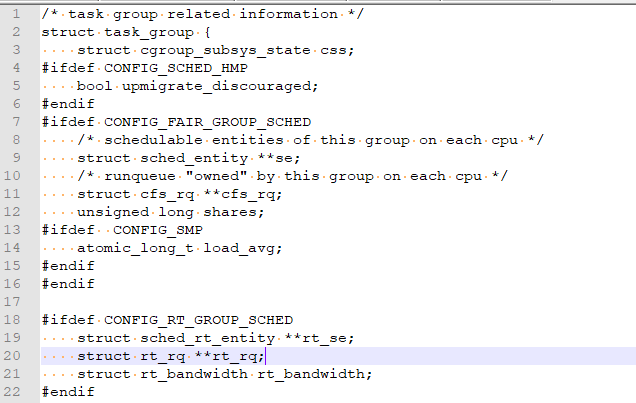


3.2.5 组调度

CFS调度器的调度粒度是进程。但是在某些应用场景中，用户希望调度的粒度是用户组。例如在一台服务器中有N个用户都可以平均分配到CPU时间。这个调度粒度为进程的CFS调度器里是很难做到的。拥有进程数据多的登录用户入将会分配到比较多的CPU资源。组调度可以解决方面的应用需求。

CFS调度器定义一个数据结构来抽象组调度struct task\_group.

[kernel/sched/sched.h]



组调度属于cgroup架构中的cpu子系统，在系统配置时需要打开CONFIG\_CGROUP\_SCHED CONFIG\_FAIR\_GROUP\_SCHED. 直接从sched\_create\_group() 函数来看如何创建和组织一个组调度。