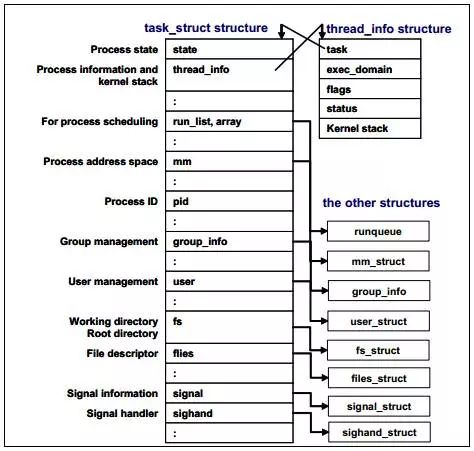
**Linux操作系统处理机管理方式**

1. **进程及其相关概念**

**进程**：进程可以理解为程序执行的一个实例，它包括可执行程序以及与其相关的系统资源，比如打开的文件、挂起的信号、内核内部数据、处理器状态、内存地址空间及包含全局变量的数据段等。从内核的角度看，进程也可以称为任务。

**进程描述符：**与进程相关的事情非常多，比如进程的状态、进程的优先级、进程的地址空间、允许该进程访问的文件等等，Linux内核为此

图1：task\_struct结构体

专门设计了一个类型为task\_struct的结构体，称之为进程描述符。进程描述符中包含了内核管理进程的所有信息，可以说，只要得到一个进程的进程描述符，就可以知道一个进程的所有信息。

**进程状态：**进程描述符task\_struct结构体中有一个state字段，表示进程当前的所处状态。从进程的创建到进程的删除，它可以经过5种不同的状态，分别是可运行状态、可中断的等待状态、不可中断的等待状态、暂停状态、跟踪状态。除此之外，当进程被终止时，还可能会变为僵死状态、僵死撤消状态。

**进程标示符：**进程描述task\_struct结构体中的pid字段可以标识唯一标识一个进程，称之为进程标识符PID。当创建一个新进程时，PID是按照顺序从小到大分配给新进程的。内核通过管理一个pidmap\_array位图来表示当前已分配的PID和闲置的PID号。注意：在多线程组中，所有的线程共享相同的PID。除了进程标识符外，内核对进程的大部分访问时通过进程描述符指针进行的。

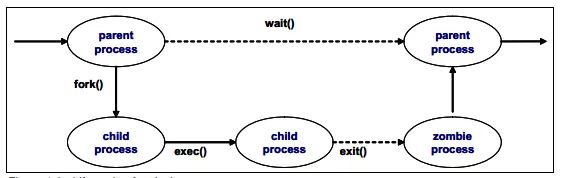


图2：经典的进程生命周期

**进程关系：**进程之间的关系有亲属关系和非亲属关系。亲属关系包括父子关系和兄弟关系等。其中由task\_struct结构体中的parent/children/sibling等字段描述。

**进程资源：**为了防止进程过度的使用系统资源，内核为每个进程使用资源的数量进行了一些限制。其中包括进程地址空间的最大数、进程使用CPU的最大时间、堆的最大值、文件大小的最大值、文件锁数量的最大值、消息队列的最大字节数、打开文件描述符的最大数、进程拥有的页框最大数等。

1. **进程的创建、切换、撤销**
2. **进程的创建**

Linux进程的创建有三个函数，fork、vfork和clone，exec系列函数常用于三者之后，用于创建一个新的程序运行环境。便于理解，先从大体来谈谈三者的区别：

* **fork()** 子进程复制父进程的所有资源。
* **clone()** 轻量级进程使用clone。
* **vfork()** 子进程和父进程共享数据段，子进程每次都优先于父进程执行。

整个进程创建过程可能涉及到如下函数：

fork() / vfork() / clone() ------> clone() ------> do\_fork() ------> copy\_process()上面的创建过程结束之后，就有了处于可运行状态的完整的子进程，新的子进程有了PID、进程描述符等各种数据结构，要想实际运行它，还需要调度程序把CPU交给新创建的子进程。

1. **进程的切换**

进程切换又称为任务切换、上下文切换。它是这样一种行为，为了控制进程的执行，内核挂起当前在CPU上运行的进程，并恢复以前挂起的某个进程的执行。进程的切换有两种方法，一种是硬件切换，一种是软件切换。软件切换就是利用程序逐步执行切换，它的优点是，可以对切换时装入的数据进行合法性检查，执行时间虽与硬件切换大致相同，但仍有可改进的地方。

1. **进程的撤销**

进程终止时，必须通知内核释放进程的资源，包括内存、打开文件以及其他零碎的东西，比如信号量。进程终止一般通过调用exit()库函数，该函数释放C函数库所分配的资源，执行编程者所注册的每个函数，并结束从系统回收进程的那个系统调用。

* **do\_group\_exit()** 内核函数实现了这个系统调用，它对应C库的exit()。
* **exit()** 终止某一个线程，而不管线程所属线程组中的所有其他线程。
* **do\_exit()** 实现这个系统调用的内核函数，它对应Linux线程库的pthread\_exit()。

1. **进程的调度**

**（一）2个调度器**

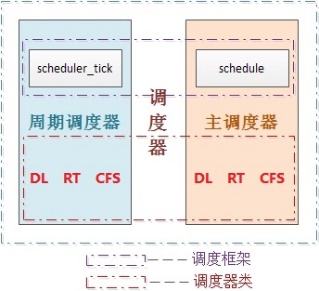
* **主调度器：**比较直接，比如进程打算睡眠或出于其他原因放弃CPU。
* **周期性调度器：**通过周期性的机制，以固定的频率运行，不时的检测是否有必要。
* Linux 调度器是以模块的方式提供的，这是为了允许不同类型的进程可以有针对性的选择调度算法。而提供不同调度算法的模块即是调度器类。每个调度器都有一个优先级，内核会先选择优先级最高的调度器，然后由该调度器调度进程并执行。

图3：Linux调度器

**（二）6种调度策略**

linux内核目前实现了6中调度策略(即调度算法), 用于对不同类型的进程进行调度, 或者支持某些特殊的功能。

|  |  |  |
| --- | --- | --- |
| **字段** | **描述** | **所在调度器类** |
| SCHED\_NORMAL | 用于普通进程，通过CFS调度器实现。SCHED\_BATCH用于非交互的处理器消耗型进程。SCHED\_IDLE是在系统负载很低时使用 | CFS |
| SCHED\_BATCH | CHED\_NORMAL普通进程策略的分化版本。采用分时策略，根据动态优先级，分配CPU运算资源。注意：这类进程比上述两类实时进程优先级低，换言之，在有实时进程存在时，实时进程优先调度。但针对吞吐量优化, 除了不能抢占外与常规任务一样，允许任务运行更长时间，更好地使用高速缓存，适合于成批处理的工作 | CFS |
| SCHED\_IDLE | 优先级最低，在系统空闲时才跑这类进程(如利用闲散计算机资源跑地外文明搜索，蛋白质结构分析等任务，是此调度策略的适用者） | CFS-IDLE |
| SCHED\_FIFO | 先入先出调度算法（实时调度策略），相同优先级的任务先到先服务，高优先级的任务可以抢占低优先级的任务 | RT |
| SCHED\_RR | 轮流调度算法（实时调度策略），后者提供 Roound-Robin 语义，采用时间片，相同优先级的任务当用完时间片会被放到队列尾部，以保证公平性，同样，高优先级的任务可以抢占低优先级的任务。不同要求的实时任务可以根据需要用sched\_setscheduler() API设置策略 | RT |
| SCHED\_DEADLINE | 新支持的实时进程调度策略，针对突发型计算，且对延迟和完成时间高度敏感的任务适用。基于Earliest Deadline First (EDF) 调度算法 | DL |

**（三）5个调度器类**

依据其调度策略的不同实现了5个调度器类, 一个调度器类可以用一种或者多种调度策略调度某一类进程, 也可以用于特殊情况或者调度特殊功能的进程。

其所属进程的优先级顺序为：

stop\_sched\_class -> dl\_sched\_class -> rt\_sched\_class -> fair\_sched\_class -> idle\_sched\_class

|  |  |  |
| --- | --- | --- |
| **调度器类** | **描述** | **对应调度策略** |
| stop\_sched\_clss | 优先级最高的线程，会中断所有其他线程，且不会被其他任务打断 作用1.发生在cpu\_stop\_cpu\_callback 进行cpu之间任务migration 2.HOTPLUG\_CPU的情况下关闭任务 | 无, 不需要调度普通进程 |
| dl\_sched\_class | 采用EDF最早截至时间优先算法调度实时进程 | SCHED\_DEADLINE |
| rt\_sched\_class | 采用提供 Roound-Robin算法或者FIFO算法调度实时进程，具体调度策略由进程的task\_struct->policy指定 | SCHED\_FIFO,SCHED\_RR |
| fair\_sched\_clas | 采用CFS算法调度普通的非实时进程 | SCHED\_NORMAL  SCHED\_BATCH |
| idle\_sched\_class | 采用CFS算法调度idle进程, 每个cup的第一个pid=0线程：swapper，是一个静态线程。调度类属于：idel\_sched\_class，所以在ps里面是看不到的。一般运行在开机过程和cpu异常的时候做dump | SCHED\_IDLE |

1. **进程的调度**

系统发生调度的时机如下：

* 调用cond\_resched()时。
* 显式调用schedule()时。
* 从系统调用或者异常中断返回用户空间时。
* 从中断上下文返回用户空间时。

当开启内核抢占(默认开启)时，会多出几个调度时机，如下

* 在系统调用或者异常中断上下文中调用preempt\_enable()时。
* 在中断上下文中，从中断处理函数返回到可抢占的上下文时。

而在系统启动调度器初始化时会初始化一个调度定时器，调度定时器每隔一定时间执行一个中断，在中断会对当前运行进程运行时间进行更新，如果进程需要被调度，在调度定时器中断中会设置一个调度标志位，之后从定时器中断返回。 每次调度时，会先在实时进程运行队列中查看是否有可运行的实时进程，如果没有，再去普通进程运行队列找下一个可运行的普通进程，如果也没有，则调度器会使用idle进程进行运行。

1. **调度所用到的函数**

调度程序依靠几个函数来完成调度工作，其中最重要的函数如下：

* try\_to\_wake\_up()函数通过把进程状态设置为TASK\_RUNNING,并把该进程插入本地CPU的运行队列来唤醒睡眠或停止的进程。
* recalc\_task\_prio()函数更新进程的平均睡眠时间和动态优先级。
* schedule()按时实现调度程序，它的任务时从运行队列的链表中找到一个进程，并随后将CPU分配给这个进程。schedule()可以由几个内核控制路径调用，可以采用直接调用或延迟调用的方式

1. **死锁产生条件及避免死锁**
2. **产生死锁的必要条件**

* 互斥条件：一个资源每次只能被一个进程（线程）使用。
* 请求与保持条件：一个进程（线程）因请求资源而阻塞时，对已获得的资源保持不放。
* 不剥夺条件 : 此进程（线程）已获得的资源，在末使用完之前，不能强行剥夺。
* 循环等待条件 : 多个进程（线程）之间形成一种头尾相接的循环等待资源关系。

1. **产生死锁的原因：**

* 竞争不可抢占资源引起死锁
* 竞争可消耗资源引起死锁
* 进程推进顺序不当引起死锁

1. **检测死锁**

Linux 提供了检测死锁的机制，主要分为 D 状态死锁和 R 状态死锁。

* **D 状态死锁：**进程等待I/O资源无法得到满足，长时间处于 TASK\_UNINTERRUPTIBLE 睡眠状态，这种状态下进程不响应异步信号。如：进程与外设硬件的交互，通常使用这种状态来保证进程与设备的交互过程不被打断，否则设备可能处于不可控的状态。对于这种死锁的检测Linux提供的是hung task机制，MTK也提供hang detect机制来检测Android系统hang机问题。触发该问题成因比较复杂多样，可能因为synchronized\_irq、mutex lock、内存不足等。D 状态死锁只是局部多进程间互锁，一般来说只是 hang 机、冻屏，机器某些功能没法使用。
* **R 状态死锁**：进程长时间处于TASK\_RUNNING状态垄断CPU而不发生切换，一般情况下是进程关抢占或关中断后长时候执行任务、死循环，此时往往会导致多CPU间互锁，整个系统无法正常调度。该问题多为原子操作，spinlock等CPU间并发操作处理不当造成。

1. **死锁的预防**

* **破坏“互斥条件”：**若资源不被一个进程独占使用，那么死锁是肯定不会发生的。但一般“互斥”条件是无法破坏的。因此，在死锁预防里面主要是破坏其他三个必要条件。
* **破坏“请求保持条件”：**所有进程在运行之前，必须一次性地申请在整个运行过程中所需的全部资源。这样，该进程在整个运行期间，便不会再提出资源请求，从而破坏了“请求”条件。
* **破坏“不可抢占”条件：**允许对资源实行抢夺。
* **破坏“循环等待条件”：**将系统中的所有资源统一编号，进程可在任何时刻提出资源申请，但所有申请必须按照资源的编号顺序（升序）提出。

1. **死锁的解除**

* **资源剥夺法：**挂起某些死锁进程，并抢占它的资源，将这些资源分配给其他的死锁进程。但应防止被挂起的进程长时间得不到资源，而处于资源匮乏的状态。
* **撤销进程法：**强制撤销部分、甚至全部死锁进程并剥夺这些进程的资源。撤销的原则可以按进程优先级和撤销进程代价的高低进行。
* **进程回退法：**让一（多）个进程回退到足以回避死锁的地步，进程回退时自愿释放资源而不是被剥夺。要求系统保持进程的历史信息，设置还原点。

1. **参考文献**

[1]吴振亚.Linux实时调度研究及改进[D].西安：西安电子科技大学，2014.

[2]张修琪.基于Linux多核进程调度的研究[D].成都：电子科技大学，2012.

[3]戴红.嵌入式Linux的预防死锁算法[N].吉林工程技术师范学院学报，2004，20(6)：1-7.

[4]嵌入式Linux内核解析[D].河北：河北工业大学，2007.

[5]Linux内核分析及实时性改造[D].成都：电子科技大学，2007.