操作系统原理与实践教程笔记

第3章 处理器管理

进程的状态

3种基本状态

就绪状态：分配处理机到运行状态

运行状态：时间片已用完到就绪状态；某时间发生（如等待I/O完成）到阻塞状态

阻塞状态：正在等待某个实践的发生（如等待I/O的完成）而暂停执行 。某时间被解除到就绪状态

具有挂起状态的进程状态

当系统资源不能满足进程运行需要时，系统必须把某些进程交换到磁盘上，不让其参与进程调度，以达到平衡系统负荷的目的，这个过程称为“挂起”。

就绪状态：被挂起到静止就绪状态

静止就绪：被激活到就绪状态

阻塞状态：被挂起到静止阻塞状态；

静止阻塞：被激活到阻塞状态；某时间被解除到静止就绪状态

进程控制块(Process Control Block,PCB)

包含：进程标识符；进程当前你状态；进程相应的程序和数据地址；进程资源清单；进程优先级；CPU现场保护区；进程同步与通信机制；进程所在队列PCB的链接字（PCB链接字指出该进程所在队列中下一个进程PCB的首地址）；御锦城有关的其他信息；

进程控制

进程控制机构

进程的控制机构是由操作系统内核实现的。

大多数操作系统内核都包括支撑功能和资源管理功能

支撑功能

终端处理

时钟处理

原语操作

进程控制原语操作主要有：创建原语、撤销原语、挂起原语、激活原语、阻塞原语以及唤醒原语

资源管理功能

进程管理

存储器管理

设备管理

进程创建

进程撤销

进程阻塞

进程唤醒

线程

线程的概念

传统的进程有两个基本属性：拥有资源的独立单位、处理器调度和分配的基本单位。由于在进程的创建、撤销和切换中，系统必须为之付出较大的时空开销，限制了并发程度的提高。引入线程，将两个基本属性分开，线程作为处理器调度和运行的基本单位，进程作为分配资源的基本单位，可以通过创建线程来完成任务，以减少程序并发执行的时空开销。

线程的实现

用户级线程和内核级线程

用户及线程仅存在于用户空间中。用户级线程的创建、撤销、线程间的同步和通信等都无需通过系统调用来实现。用户级线程是与内核无关的。由于这些线程的PCB设置在用户空间，所有内核完全不知道用户级线程的存在。在设置了用户级线程的操作系统中，任然是以进程为单位进行调度的。

内核级线程是在内核的支持下运行的。内核线程的创建和管理要鳗鱼用户级线程。

多线程的模型

在支持用户级和内核级线程的系统中：

多对一模型：许多用户级线程映射到一个内核线程。线程管理实在用户空间进行，效率高，但是如果一个线程执行了阻塞系统调用，那么整个进程就会阻塞，因此讴歌线程不能并行运行在多处理器上。

一对一模型：

多对多模型：克服了上面模型的缺点，可以创建任意多个必要的线程，线程执行阻塞系统调用时，内核能够调度另一个线程来执行。

线程池

传统多线程发难中，采用的服务器模型是即时创建即时销毁的策略。对于执行次数及其频繁的任务，会耗费大量的CPU时间。

线程池的实现原理类似于操作系统中的缓冲区概念：预先创建若干数量的线程，并让这些线程都处于睡眠状态，不消耗CPU资源，当个苦短有一个新请求时，就会唤醒线程池中的某一个睡眠线程来处理请求，当处理完后，线程又处于睡眠状态。

处理器调度

处理器调度的层次

高级调度：又称宏观调度或作业调度，需要决定：从驻留在外存后备队列中调入多少个作业，以及要调入哪几个作业；最后把新创建的进程放入就绪队列中。

中级调度：进程在内存和外存之间的交换。就是挂起和激活。中级调度其实就是存储器管理中的对换功能。

低级调度：又称进程调度，按照一定的算法从就绪队列中选择一个进程，然后将处理器分配给它。

作业调度

从后备作业队列中选择作业装入内存运行，并在作业运行结束后做善后处理。

4个状态：

提交状态；后备状态；运行状态；完成状态；

工作：

选择作业；分配资源；建立作业进程；简历有关表格；作业善后处理

进程调度

非抢占方式：调度程序一旦把CPU分配给某一进程后便让其一直运行下去，知道进程完成或发生某时间而不能运行时，才将CPU分给其他进程

抢占方式

需要遵循的原则：时间片原则；优先级原则；短进程优先原则

非抢占优点：简单、系统开销小、用在批处理系统；

抢占优点：调度方式能够及时响应各进程的请求，用在分时系统和实时系统中

调度算法

先来先服务（First-Come, First-Served, FCFS）,是非抢占式

短作业（进程）优先（Shortest-Job-First, SJF），估计进程的运行时间，每次选择所需时间最短的进程

优先级调度（Priority-Scheduling Algorithm），每次选择优先级最高的进程，如果优先级相同就按FCFS；分为：非抢占式和抢占式；确定优先级的方法：静态优先级和动态优先级

时间片轮转法（Round-Robin, RR）

多级队列调度算法（Multilevel-Queue-Scheduling Alogorithm）,将就绪进程队列分成多个独立队列，每个队列有自己的调度算法，队列之间有优先级差异。

多级反馈队列调度：UNIX，Windows NT、OS/2都采用了类似的调度算法。又称队列轮转法。

例如假设采用抢占式调度：

设置多个就绪队列，并且队列间有不同的优先级，每个队列分配不同的时间片，优先级越低时间片越长

新进程就绪后，现插入队列1，按FCFS调度。如果一个时间片未能执行完，就插入到队列2，以此类推

进程由于等待时间而放弃CPU后，进入等待队列，一旦等待时间发生，则回到原来的就绪队列

当高优先级的队列为空时，才调度低优先级队列中的进程

高响应比优先调度

响应比=周转时间/运行时间=1+（等待时间/运行时间）

多处理器调度和实时调度

多处理器调度

实时调度

硬实时：应用的时间需求能够得到完全的满足；军工、航天、核工业

软实时：即时通信和信息采集系统

Linux的进程管理

Linux的进程描述符

Linux中没有明确的数据结构表示线程，其通过扩展进程机制创建轻权进程并设置轻权进程组来支持多进程并行

在Linux中，每一个进程（包括轻权进程）用一个task\_struct数据结构来表示，就是进程的PCB，也称进程描述符

|  |  |  |  |
| --- | --- | --- | --- |
| 项 | 名称 | 类型 | 描述 |
| 进程标识信息 | pid | pid\_t | 进程标识（唯一） |
| pgrp | pid\_t | 进程组标识 |
| uid,suid | uid\_t | 进程用户标识 |
| gid,sgid | gid\_t | 进程用户组标识 |
| 进程运行环境信息 | thread\_info | thread\_info\* | 当前进程运行的环境信息 |
| 进程地址空间信息 | mm | mm\_struct\* | 指向进程的内存描述符 |
| 进程窗口相关信息 | tty | tty\_struct\* | 指向终端窗口描述符 |
| 文件相关信息 | files | files\_struct\* | 指向文件资源描述符 |
| 进程调度信息 | state | long | 进程运行状态 |
| rt\_priority | long | 实时进程的优先级 |
| nice | long | 普通进程的优先级 |
| counter | long | 时间片 |
| 信号处理信息 | signal | long | 进程接受到的信号 |
| blocked | long | 进程接受信号的掩码 |
| signal | signal\_struct\* | 指定进程接受信号处理函数 |
| 处理机执行现场 | policy | long | 进程调度策略 |
| 描述进程间关系的指针 | next\_task | task\_struct\* | 进程链的向前指针，向后指针 |
| prev\_task |
| thread\_group | list\_head\* | 轻权进程组链的前、后指针 |
| p\_pptr | task\_struct\* | 指向父进程的指针 |
| p\_cptr | task\_struct\* | 指向孩子进程的指针 |

在Linux中，用户程序通过系统调用fork()创建一个进程，调用clone()创建一个轻权进程LWP。进程和轻权进程都有唯一的pid，但轻权进程和其父进程有相同的进程组标识符pgrp(即父进程的pid)，轻权进程可以和父进程共享地址空间、终端窗口、打开文件表等信息

Linux有两种运行模式----核心态和用户态。进程一般运行在用户态，只有出发终端异常或系统调用时才前环岛核心态。进程拥有两个栈：用户模式栈和核心模式栈。进程描述符的运行唤醒信息thread\_info和进程的核心栈的空间分配在一起，而task\_struct对象利用slab分配器进行分配。

Linux的进程状态及转换

基本状态：七种

运行状态(TASK\_RUNNING):进程在运行或具备运行资格，等待被调度执行的进程。挂入运行队列(run\_queue)中。

可中断睡眠状态(TASK\_INTERRUPTIBLE):等待资源的阻塞或睡眠状态，知道所需资源满足时，进程被唤醒或被信号终止睡眠状态

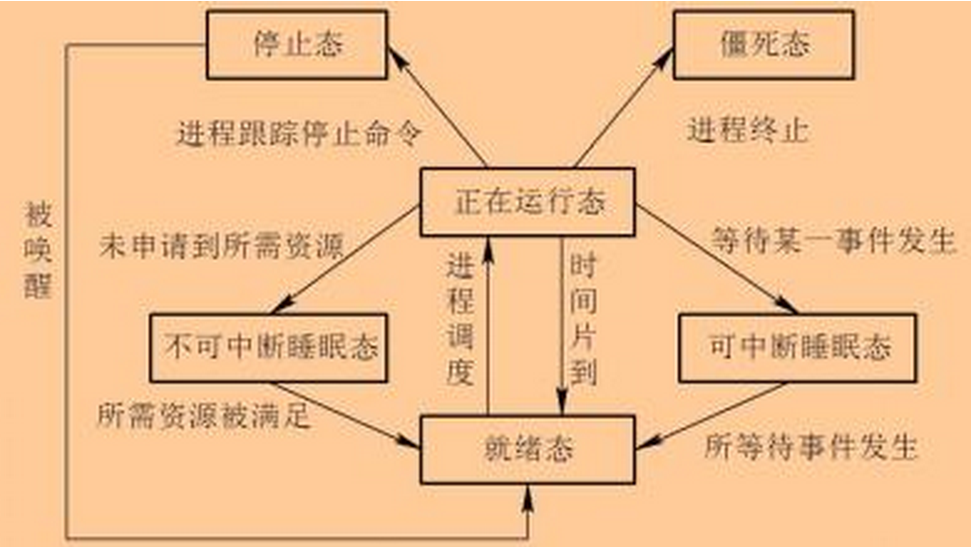
不可终端睡眠状态(TASK\_UNINTERRUPTIBLE):。。。不能被信号唤醒

暂停状态(TASK\_STOPPED):进程处于暂停或挂起状态，当进程处于SIGSTOP、SIGTSTP、SIGTTIN或SIGTTOU信号时进入该状态

跟踪状态(TASK\_TRACED):当调试进程收到信号时进入该状态并通知调试器，等待调试器发出继续的命令

僵死状态(TASK\_ZOMBIE):进程已结束，并释放了大部分占用的资源，等待内核收回进程的task\_struct结构

退出状态(TASK\_DEAD):与僵死状态类似，只是不需要等待内核回收进程描述符和核心栈



对状态的解释：<http://blog.chinaunix.net/uid-20539088-id-2610497.html> 。

Linux进程的调度

Linux是一个分时操作系统，Linux支持实时进程的概念

在Linux中，进程通常称为任务(task)。对进程的调度有进程控制模块实现，进程task\_struct结构中有下列调度信息

policy:指定进程的调度策略，Linux进程有实时进程和普通进程两种。SCHED\_RR(轮转调度策略)SCHED\_FIFO(先进先出)用户实时进程，实时进程比普通进程优先级高，不同的实时进程间的优先级相同。SCHED\_OTHER用于普通进程，SCHED\_YIELD表示进程主动放弃CPU资源

priority:进程调度优先级

rt\_priority:实时进程专用调度优先级

counter：进程被调度室运行时间的长短

在Linux中没有一个专门的系统进程处理进程调度事宜，进程完成系统的初始化后，有核心提供的schedule()函数来实现任务调度

Linux进程的创建和终止

进程的创建

Linux系统启动时系统中只有一个初始化进程，该进程运行在核心态，系统初始化结束时，初始化进程创建一个init进程(标识号为1)，而进程处于空转状态，一二进程在Linux中也成为空闲进程。系统中所有其他进程都是有init进程派生的。当一个进程创建其子进程是，子进程的task\_struct是动态生成的，只有空闲进程的task\_struct是金泰分配的。

Linux提供了系统调用for()和clone()来创建子进程，而系统调用for()和clone()都是调用内核函数do\_fork()，通过克隆当前进程来创建子进程。该函数所做工作：

1. 分配一个task\_struct内存空间，并将当前进程的task\_struct内容复制其中；
2. 查看当前进程树是否超过系统准许，并从tarray\_freelist分配一个指向新分配的task\_struct的指针；
3. 设置task\_struct中各成员的值，获取一个可以使用的进程标识符；
4. 从父进程中继承文件表、文件系统信息信号、处理函数等信息；
5. 赋值内存管理数据，为新进程建立新的页目录、页表等；
6. 建立进程上下文环境，唤醒新进程，即设置新进程的状态为TASK\_RUNNING，并放入运行队列中，等待调度；
7. 系统总进程数加1；

Linux允许两个进程共享相同的资源，如文件、信号处理程序等。当某个资源被共享时，该资源的引用数会增加1，从而只有当使用资源的进程都终止时，内核才会释放这些资源

进程的终止

当进程正常结束时，进程调用exit()函数，或在main函数中执行return；或者main函数执行完，进程结束。当进程在运行期间出现错误或故障被批结束时，Linux通过调用abort()或收到进程外所发来的信号而结束，属于异常结束

无论进程是正常结束还是非正常结束，最终都要调用内核函数do\_exit()。进程释放占用的资源，并调用调度程序是其他进程被调度运行。do\_exit()的工作：

1. 释放进程的数据结构和内存等资源
2. 将进程状态设置为TASK\_ZOMBIE
3. 通知父进程等相关进程，将所有子进程的父进程设置为init进程
4. 调用schedule()，调度就绪队列中的进程到CPU运行

因为父进程要通过wait()之类的系统调用来确定子进程是否结束，当子进程结束时，子进程描述符和其关联的部分数据结构也不能立即被释放，子进程处于EXIT\_ZOMBIE状态，此时父进程释放子进程task\_struct，进程结束

Linux的线程管理

Linux支持多线程、多任务，没有单独定义线程，而是将线程定义为“执行上下文”，实际是进程的另外一个执行上下文，Linux基本调度单位任然是进程。

Linux线程分为：用户线程和内核线程。

用户线程：是用户使用的处理程序中多个控制流的单元，不需要内核支持，使用POXIS的Pthreads线程库提供的函数实现线程的创建、同步、调度和管理。

内核线程：管理调度由内核完成，可被系统的其他线程访问，线程之间可以用系统调用cloe()来共享地址空间

Linux线程的调度非抢先式调度，线程之间的切换只发送在线程自动放弃对CPU控制时。但是内核线程的调度是抢先式的。调度策略有三种：SCHED\_OTHER(普通分时调度策略);SCHED\_FIFO(先进先出调度策略)；SCHED\_RR(循环调度策略)；

Linux当多个线程同时访问一个资源时，会产生不可余量的后果，同时，几个相互配合的线程也存在如何同步、协调运行的问题。所以，线程必须有一定的同步机制。

Linux同步对象：自旋锁、信号量、条件变量

自旋锁：通过一个时刻只允许一个线程访问临界区共享资源的方式来确保资源的一致性。0表示解锁，1表示上锁

信号量

条件变量：用来保证线程实现阻塞，知道某个事件发送或条件成立。

条件变量用来发送信号表示某一个操作完成，相对于等待资源锁定来说，其更适合于等待事件；而自旋锁和信号量主要用于控制对数据的访问