操作系统

进程与线程及其区别

- 进程是对运行时程序的封装，是系统进行资源调度和分配的的基本单位，实现操作系统的并发；
- 线程是进程的子任务，是CPU调度和分派的基本单位，用于保证程序的 实时性，实现进程内部的并发；
- 一个程序至少有一个进程，一个进程至少有一个线程，线程依赖于进程而存在；
- 进程在执行过程中拥有独立的内存单元，而多个线程共享进程的内存。

进程间通信方式

- 管道（pipe）及命名管道（named pipe）：管道可用于具有亲缘关系的父子进程间的通信，有名管道除了具有管道所具有的功能外，它还允许无亲缘关系进程间的通信；
- 信号（signal）：信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生；
- 消息队列：消息队列是消息的链接表，它克服了上两种通信方式中信号量有限的缺点，具有写权限得进程可以按照一定得规则向消息队列中添加新信息；对消息队列有读权限得进程则可以从消息队列中读取信息；
- 共享内存：可以说这是最有用的进程间通信方式。它使得多个进程可以访问同一块内存空间，不同进程可以及时看到对方进程中对共享内存中数据得更新。这种方式需要依靠某种同步操作，如互斥锁和信号量等；
- 信号量：主要作为进程之间及同一种进程的不同线程之间得同步和互斥手段；
- 套接字：这是一种更为一般得进程间通信机制，它可用于网络中不同机器之间的进程间通信，应用非常广泛。

线程同步方式

- 互斥量Mutex：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以可以保证公共资源不会被多个线程同时访问。
- 信号量 Semphare：它允许同一时刻多个线程访问同一资源，但是需要控制同一时刻访问此资源的最大线程数量
- 事件(信号)Wait/Notify：通过通知操作的方式来保持多线程同步，还可以方便的实现多线程优先级的比较操作（感觉类似条件变量）
- 临界区：保证在某一时刻只有一个线程能访问数据的简便办法。

死锁概念、死锁产生的条件与死锁的基本应对方法

- 概念：在两个或者多个并发进程中，如果每个进程持有某种资源而又等待其它进程释放它或它们现在保持着的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁。通俗的讲，就是两个或多个进程无限期的阻塞、相互等待的一种状态。
- 四个必要条件
  - 互斥：至少有一个资源必须属于非共享模式，即一次只能被一个进程使用；若其他申请使用该资源，那么申请进程必须等到该资源被释放为止；
  - 占有并等待：一个进程必须占有至少一个资源，并等待另一个资源，而该资源为其他进程所占有；
  - 非抢占：进程不能被抢占，即资源只能被进程在完成任务后自愿释放
  - 循环等待：若干进程之间形成一种头尾相接的环形等待资源关系
- 死锁的处理基本策略和常用方法

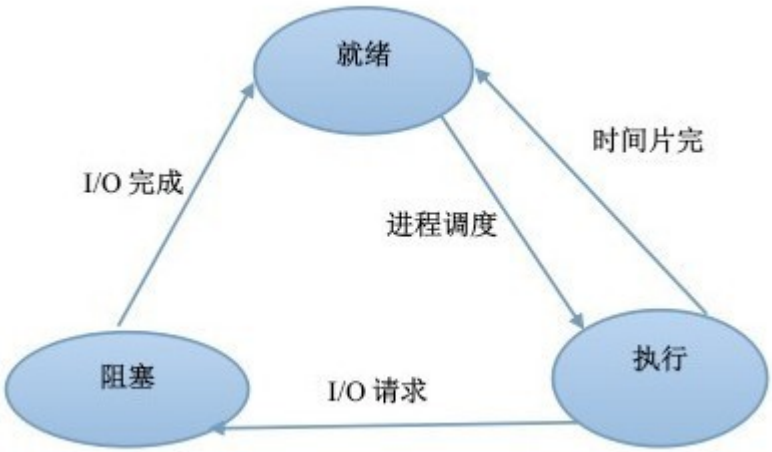
  解决死锁的基本方法主要有 预防死锁、避免死锁、检测死锁、解除死锁 、鸵鸟策略 等。
  - 死锁预防：基本思想是 只要确保死锁发生的四个必要条件中至少有一个不成立，就能预防死锁的发生
  - 避免死锁：基本思想是动态地检测资源分配状态，以确保循环等待条件不成立，从而确保系统处于安全状态。所谓安全状态是指：如果系统能按某个顺序为每个进程分配资源（不超过其最大值），那么系统状态

是安全的，换句话说就是，如果存在一个安全序列，那么系统处于安全状态。资源分配图算法和银行家算法是两种经典的死锁避免的算法，其可以确保系统始终处于安全状态。其中，资源分配图算法应用场景为每种资源类型只有一个实例(申请边，分配边，需求边，不形成环才允许分配)，而银行家算法应用于每种资源类型可以有多个实例的场景。

- 死锁解除：常用两种方法为进程终止和资源抢占。所谓进程终止是指简单地终止一个或多个进程以打破循环等待，包括两种方式：终止所有死锁进程和一次只终止一个进程直到取消死锁循环为止；所谓资源抢占是指从一个或多个死锁进程那里抢占一个或多个资源，此时必须考虑三个问题：1、选择一个牺牲品；2、回滚到安全状态；3、饥饿

进程状态转移

- 就绪状态：进程已获得除处理机以外的所需资源，等待分配处理机资源；
- 运行状态：占用处理机资源运行，处于此状态的进程数小于等于CPU数；
- 阻塞状态： 进程等待某种条件，在条件满足之前无法执行；



Linux进程状态转移

- 运行状态（TASK_RUNNING）：当进程正在被CPU执行，或已经准备就绪随时可由调度程序执行，则称该进程为处于运行状态（running）。进程可以在内核态运行，也可以在用户态运行。当系统资源已经可用时，进程就被唤醒而进入准备运行状态，该状态称为就绪态。这些状态（图中中间一列）在内核中表示方法相同，都被成为处于TASK_RUNNING状态。
- 可中断睡眠状态（TASK_INTERRUPTIBLE）：当进程处于可中断等待状态时，系统不会调度该进程执行。当系统产生一个中断或者释放了进程正在等待的资源，或者进程收到一个信号，都可以唤醒进程转换到就绪状态（运行状态）。
- 不可中断睡眠状态（TASK_UNINTERRUPTIBLE）：与可中断睡眠状态类似。但处于该状态的进程只有被使用wake_up()函数明确唤醒时才能转换到可运行的就绪状态。
- 暂停状态（TASK_STOPPED）：当进程收到信号SIGSTOP、SIGTSTP、SIGTTIN或SIGTTOU时就会进入暂停状态。可向其发送SIGCONT信号让进程转换到可运行状态。在Linux 0.11中，还未实现对该状态的转换处理。处于该状态的进程将被作为进程终止来处理。
- 僵死状态（TASK_ZOMBIE）：当进程已停止运行，但其父进程还没有询问其状态时，则称该进程处于僵死状态。 当一个进程的运行时间片用完，系统就会使用调度程序强制切换到其它的进程去执行。另外，如果进程在内核态执行时需要等待系统的某个资源，此时该进程就会调用sleep_on()或sleep_on_interruptible()自愿地放弃CPU的使用权，而让调度程序去执行其它进程。进程则进入睡眠状态（TASK_UNINTERRUPTIBLE或TASK_INTERRUPTIBLE）。 只有当进程从“内核运行态”转移到“睡眠状态”时，内核才会进行进程切换操作。在内

核态下运行的进程不能被其它进程抢占，而且一个进程不能改变另一个进程的状态。为了避免进程切换时造成内核数据错误，内核在执行临界区代码时会禁止一切中断。

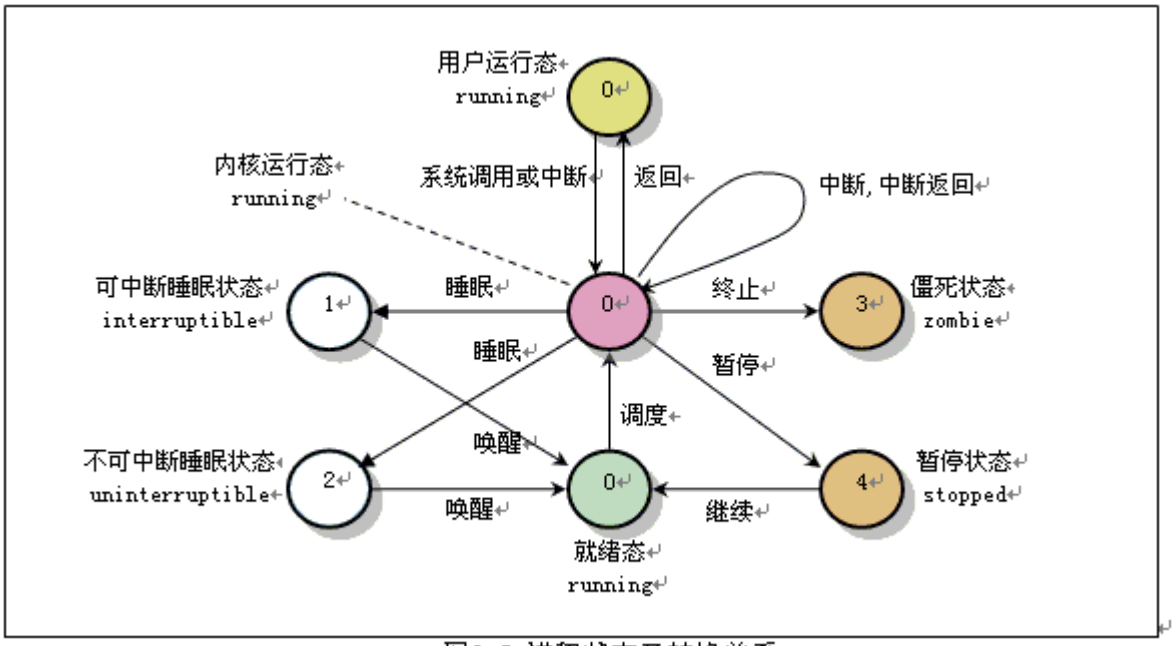


图2.6 进程状态及转换关系

页面置换算法

- FIFO先进先出算法：在操作系统中经常被用到，比如作业调度（主要实现简单，很容易想到）；
- LRU（Least recently use）最近最少使用算法：根据使用时间到现在的长短来判断；
- LFU（Least frequently use）最少使用次数算法：根据使用次数来判断；
- OPT（Optimal replacement）最优置换算法：理论的最优，理论；就是要保证置换出去的是不再被使用的页，或者是在实际内存中最晚使用的算法。

进程调度策略

- FCFS(先来先服务，队列实现，非抢占的)：先请求CPU的进程先分配到CPU
- SJF(最短作业优先调度算法)：平均等待时间最短，但难以知道下一个CPU区间长度
- 优先级调度算法(可以是抢占的，也可以是非抢占的)：优先级越高越先分配到CPU，相同优先级先到先服务，存在的主要问题是：低优先级进程无穷等待CPU，会导致无穷阻塞或饥饿；解决方案：**老化**
- 时间片轮转调度算法(可抢占的)：队列中没有进程被分配超过一个时间片的CPU时间，除非它是唯一可运行的进程。如果进程的CPU区间超过了一个时间片，那么该进程就被抢占并放回就绪队列。
- 多级反馈队列调度算法：与多级队列调度算法相比，其允许进程在队列之间移动：若进程使用过多CPU时间，那么它会被转移到更低的优先级队列；在较低优先级队列等待时间过长的进程会被转移到更高优先级队列，以防止饥饿发生。
- linux CFS：其基本思路很简单，他把CPU当做一种资源，并记录下每一个进程对该资源使用的情况，在调度时，调度器总是选择消耗资源最少的进程来运行。这就是所谓的“完全公平”。但这种绝对的公平有时也是一种不公平，因为有些进程的工作比其他进程更重要，我们希望能按照权重来分配CPU资源。为了区别不同优先级的进程，就是会根据各个进程的权重分配运行时间，分配给进程的运行时间 = 调度周期 * 进程权重 / 所有进程权重之和。
- linux CFS实现原理：

  分配给进程的运行时间 = 调度周期 * 进程权重 / 所有进程权重之和

vruntime = 实际运行时间 * 1024 / 进程权重

vruntime = (调度周期 * 进程权重 / 所有进程总权重) * 1024 / 进程权重 = 调度周期 * 1024 / 所有进程总权重， 所有进程的vruntime增长速度宏观上看应该是同时推进的。

1024等于nice为0的进程的权重，代码中是NICE_0_LOAD。也就是说，所有进程都以nice为0的进程的权重1024作为基准，计算自己的vruntime增加速度，权重跟进程nice值之间有一一对应的关系。

CFS的思想就是让每个调度实体（没有组调度的情形下就是进程，以后就说进程了）的vruntime互相追赶，而每个调度实体的vruntime增加速度不同，权重越大的增加的越慢，这样就能获得更多的cpu执行时间。

第一个是调度实体sched_entity，它代表一个调度单位，在组调度关闭的时候可以把他等同为进程。**每一个task_struct中都有一个sched_entity，进程的vruntime和权重都保存在这个结构中。那么所有的sched_entity怎么组织在一起呢？红黑树。所有的sched_entity以vruntime为key(实际上是以vruntime-min_vruntime为key，是为了防止溢出，反正结果是一样的)插入到红黑树中，同时缓存树的最左侧节点，也就是vruntime最小的节点，这样可以迅速选中vruntime最小的进程。**

注意只有等待CPU的就绪态进程在这棵树上，睡眠进程和正在运行的进程都不在树上。

Linux中断系统

- 硬中断与上下部分（任务队列、软中断和tasklet）
- 软中断信号机制：
  - 设置时机：内核给进程发送软中断信号的方法，是在进程所在的进程表项的信号域设置对应于该信号的位。若进程处于休眠状态，需看该进程进入睡眠的优先级，若可被中断的优先级上，则唤醒进程；否则仅设置进程表中信号域相应的位，而不唤醒进程。
  - 处理时机：内核处理一个进程收到的信号的时机是在一个进程从内核态返回用户态时。所以，当一个进程在内核态下运行时，软中断信号并不立即起作用，要等到将返回用户态时才处理。进程只有处理完信号才会返回用户态，进程在用户态下不会有未处理完的信号。内核处理一个进程收到的软中断信号是在该进程的上下文中，因此，进程必须处于运行状态。

fork、vfork、clone

- fork：创建的子进程是父进程的完整副本，采用写时复制的方式，其实际开销即复制父进程的页表以及给子进程创建唯一的进程描述符。
- vfork：创建的子进程与父进程共享数据段,而且由vfork()创建的子进程将先于父进程运行
- clone：更细粒度地控制与子进程共享的资源，底层线程的创建（共享内存空间、文件系统信息、打开的文件、信号处理函数）
- 子进程从父进程拷贝的内容主要：用户号和用户组号、环境、堆栈、共享内存、打开的文件描述符、执行时关闭标志、信号设定、进程组号、工作目录、文件方式创建屏蔽字
- 多线程独有：线程ID、寄存器组的值、线程的堆栈、错误返回码、信号屏蔽字、线程优先级、程序计数器。

  共享：代码空间、全局变量、打开的文件描述符、信号的处理函数、进程的当前目录和进程用户ID与进程组ID

僵尸进程与孤儿进程

- 孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init进程(进程号为1)所收养，并由init进程对它们完成状态收集工作，无危害。
- 僵尸进程：一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。称为僵死进程。

unix提供了父进程获取子进程结束时的状态信息的机制，: 在每个进程退出的时候,内核释放该进程所有的资源,包括打开的文件,占用的内存等。 但是仍然保留一定的信息(包括进程号,退出状态,运行时间等)。直到父进程通过wait / waitpid来取时才释放。 但这样就导致了问题，**如果进程不调用wait / waitpid的话， 那么保留的那段信息就不会释放，其进程号就会一直被占用，但是系统所能使用的进程号是有限的，如果大量的产生僵死进程，将因为没有可用的进程号而导致系统不能产生新的进程. 此即为僵尸进程的危害，应当避免。**任何一个子进程(init除外)在exit()之后，并非马上就消失掉，而是留下一个称为僵尸进程(Zombie)的数据结构，等待父进程处理。

- 僵尸进程处理方法：1、子进程退出时向父进程发送SIGCHILD信号，父进程处理SIGCHILD信号；2、**fork两次**：原理是将子进程成为孤儿进程，从而其的父进程变为init进程，通过init进程可以处理僵尸进程。

- linux内存管理方式
- linux文件管理方式
- 虚拟内存
- 多线程与多进程

堆与栈的区别

- 管理方式不同：栈是由编译器自动分配和释放,使用方便;而对于堆来说,分配和释放都必须由程序员来手动完成,不易管理,容易造成内存泄漏和内存碎片；
- 可用空间不同：对于栈来说,它可用的内存空间比较小;而对于堆来说它可以使用的空间比栈要大的多；
- 能否产生碎片：由于栈采用的是后进先出的机制,所以栈空间没有内存碎片的产生;而对于堆来说,由于频繁的使用new/delete势必会造成内存空间分配的不连续,从而造成大量的碎片,使程序的效率降低；
- 生长方式不同：对于堆来说,它一般是向上的;即是向着地址增加的方向增长;对于栈来说,它一般是向下的,即向着地址减小的方向增长.
- 分配效率不同：栈是机器系统提供的数据结构,计算机会在底层对栈提供支持:为栈分配专门的寄存器.压栈和出栈都由专门的指令进行.因此它的效率会很高;而堆则是由c/c++库函数实现的,机制是非常的负责的;例如要分配一块内存的时候,库函数会利用特定的算法在堆内存中搜索可用大小的内存空间;如果没有足够大的内存空间,就会调用系统功能去增加数据段的内存空间.这样才能得到足够大的可用的内存空间,因此堆内存的分配的效率比栈要低得多.

- 软链接与硬链接
- 常用linux命令
- 进程优雅退出
- mmap
- 协程
- 自旋锁与互斥锁
- linux进程状态转移