进程调度

基本概念

- CPU-I/o 执行周期
  - CPU 的调度成功取决于如下观察到的进程属性： 进程执行包括周期 cycle) 进行 CPU 执行和I/O 等待。
- CPU 调度程序
  - 每当 CPU 空闲时， 操作系统就应从就绪队列中选择一个进程来执行。 进程选择采用短期调度程序 或 CPU 调度程序，调度程序从内存中选择一个能够执行的进程，并为其分配 CPU。
- 抢占调度
  - CPU 调度的情况可分为以下四种
    - 当一个进程从运行状态切换到等待状态时（ 例如， I/O 请求， 或 waito 调用以便等待一个子进程的终止）。
    - 当一个进程从运行状态切换到就绪状态时（ 例如， 当出现中断时）。
    - 当一个进程从等待状态切换到就绪状态时（ 例如， I/O 完成）。
    - 当一个进程终止时。
  - 在非抢占调度下， 一旦某个进程分配到 CPU, 该进程就会一直使用 CPU, 直到它终止或切换到等待状态。
- 调度程序
  - 调度程序是一个模块， 用来将 CPU 控制交给由短期调度程序选择的进程。
    - 切换上下文。
    - 切换到用户模式。
    - 跳转到用户程序的合适位置， 以便重新启动程序。
  - 调度程序停止一个进程而启动另一个所需的时间称为调度延迟

调度准则

- CPU 使用率： 应使 CPU 尽可能地忙碌。
- 吞吐量： 如果 CPU 忙于执行进程， 那么工作就在完成。 一种测量工作的方法称为吞吐量， 它是在一个时间单元内进程完成的数量。
- 周转时间： 从一个特定进程的角度来看， 一个重要准则是运行这个进程需要多长时间。 从进程提交到进程完成的时间段称为周转时间
- 等待时间： CPU 调度算法并不影响进程运行和执行 I/O 的时间， 它只影响进程在就绪队列中因等待所需的时间。 等待时间为在就绪队列中等待所花时间之和。
  - 最大化 CPU 使用率和吞吐量， 并且最小化周转时间、 等待时间和响应时间， 这是可取的。 在大多数情况下， 优化的是平均值。 然而， 在有些情况下， 优化的是最小值或最大值，而不是平均值。
- 响应时间： 对于交互系统， 周转时间不是最佳准则。 通常， 进程可以相当早地产生输出， 并且继续计算新的结果同时输出以前的结果给用户。 因此， 另一时间是从提交_请求到产生第一响应的时间。 这种时间称为响应时间，

调度算法

- 先到先服务调度
  - FCFS 调度代码编写简单并且理解容易。FCFS 策略的缺点是， 平均等待时间往往很长。
- 最短作业优先调度
  - 当 CPU变为空闲时它会被赋给具有最短 CPU 执行的进程。 如果两个进程具有同样长度的 CPU 执行， 那么可以由 FCFS 来处理。一个更为恰当的表示是最短下次 CPU 执行
  - 抢占 SJF 调度有时称为最短剩余时间优先
- 优先级调度
  - 优先调度可以是抢占的或非抢占的。 当一个进程到达就绪队列时， 比较它的优先级与当前运行进程的优先级。 如果新到达进程的优先级高于当前运行进程的优先级， 那么抢占优先级调度算法就会抢占 CPU 非抢占优先级调度算法只是将新的进程加到就绪队列的头部。
  - 优先级调度算法的一个主要问题是无穷阻塞 或饥饿就绪运行但是等待 CPU 的进程可以认为是阻塞的。 优先级调度算法可让某个低优先级进程无穷等待 CPU
- 轮转调度
  - 轮转 调度算法是专门为分时系统设计的。 它类似于 FCFS 调度，但是增加了抢占以切换进程。 将一个较小时间单元定义为时间量 或时间片 时间片的大小通常为 10 100ms 就绪队列作为循环队列。 CPU 调 度 程 序 循_环整个就绪队列， 为每个进程分配不超过一个时间片的 CPU
- 多级队列调度
  - 多级队列 调度算法将就绪队列分成多个单独队列。 根据进程属性， 如内存大小、进程优先级、 进程类型等， 一个进程永久分到一个队列。 每个队列有自己的调度算法。
- 多级反馈队列调度
  - 多级反馈队列 调度算法允许进程在队列之间迁移。 这种想法是， 根据不同 CPU 执行的特点来区分进程。 如果进程使用过多的 CPU 时间， 那么它会被移到更低的优先级队列。

线程调度

- 竞争范围
  - 对于实现多对一 和多对多 模型的系统线程库会调度用户级线程。 以便在可用 LWP 上运行。 这种方案称为进程竞争范围 PCS), 因为竞争 CPU 是发生在同一进程的线程之间。
  - 内核采用系统竞争范围 SCS) 采用 SCS调度来竞争 CPU, 发生在系统内的所有线程之间。 采用一对一模型的系统，如Windows Linux 和 Solaris, 只采用 SCS 调度。
- Pthreads 调度

多处理器调度

有多个 CPU, 则负载分配成为可能， 但是调度问题就相应地更为复杂。

- 多处理器调度的方法
  - CPU 调度的一种方法是让一个处理器（ 主服务器） 处理所有调度决定、 I/O 处理以及其他系统活动， 其他的处理器只执行用户代码。 这种非对称多处理
  - 第二种方法是使用对称多处理 SMP), 即每个处理器自我调度。 所有进程可能处于一个共同的就绪队列中， 或每个处理器都有它自己的私有就绪进程队列。
- 处理器亲和性
  - 大多数 SMP 系统试图避免将进程从一个处理器移到另一个处理器， 而是试图让一个进程运行在同一个处理器上。 这称为处理器亲和性, 即一个进程对它运行的处理器具有亲和性。
  - 当一个操作系统试图保持进程运行在同一处理器上时(但不保证它会这么做)， 这种情况称为软亲和性
  - 相反， 有的系统提供系统调用以便支持硬亲和性, 从而允许某个进程运行在某个处理器子集上。
  - 系统的内存架构可以影响处理器的亲和性。
- 负载平衡
  - 负载平衡设法将负载平均分配到 SMP 系统的所有处理器。
  - 对于推迁移， 一个特定的任务周期性地检查每个处理器的负载， 如果发现不平衡， 那么通过将进程从超载处理器推到 空闲或不太忙的处理器， 从而平均分配负载。 当空闲处理器从一个忙的处理器上拉一个等待任务时， 发生拉迁移。
  - 负载平衡通常有两种方法： 推迁移 和拉迁移。
- 多核处理器
  - 一个处理器访问内存时， 它花费大量时间等待所需数据。 这种情况称为内存停顿
  - 对于粗粒度的多线程， 线程一直在处理器上执行， 直到一个长延迟事件(如内存停顿） 发生。 由于长延迟事件造成的延迟， 处理器应切换到另一个线程来开始执行。然而， 线程之间的切换成本是高的， 因为在另一个线程可以在处理器核上开始执行之前， 应刷新指令流水线。 一旦这个新的线程开始执行， 它会开始用指令来填充流水线。
  - 一般来说， 处理器核的多线程有两种方法： 粗粒度和细粒度 的多线程。
    - 细粒度（ 或 交错） 的多线程在更细的粒度级别上（通常在指令周期的边界上） 切换线程。 而且， 细粒度系统的架构设计有线程切换的逻辑。 因此， 线程之间的切换成本很小

实时CPU调度

软实时系统 不保证会调度关键实时进程； 而只保证这类进程会优先于非关键进程(， 硬实时系统有更严格的要求。 一个任务应在它的截止期限之前完成； 在截止期限之后完成， 与没有完成， 是完全一样的。

- 最小化延迟
  - 中断延迟 是从 CPU 收到中断到中断处理程序开始的时间。
    - 显然， 对实时操作系统来说， 至关重要的是： 尽量减少中断延迟， 以确保实时任务得到立即处理。 事实上， 对于硬实时系统， 中断延迟不只是简单地最小化， 而是要有界以便满足这些系统的严格要求
  - 调度程序从停止一个进程到启动另一个进程所需的时间量称为调度延迟
    - 提供实时任务立即访问 CPU 要求， 实时操作系统最大限度地减少这种延迟。 保持调度延迟尽可能低的最有效技术是， 提供抢占式内核。
- 优先权调度
  - 提供抢占的、 基于优先级的调度程序仅保证软实时功能。 硬实时系统应进一步保证实时任务应在截止期限内得到服务， 做出这样的保证需要附加的调度特征。
  - 这种形式调度的不寻常之处在于， 进程可能应向调度器公布其截止期限要求。 然后， 使用一种称为准入控制算法的技术， 调度程序做两件事之一：它承认进程，保证进程完成； 如果它不能保证任务能在截止期限前得以服务，拒绝请求
- 单调速率调度
  - 单调速率 调度算法采用抢占的 、 静态优先级的策略， 调度周期性任务。在进入系统时， 每个周期性任务会分配一个优先级， 它与其周期成反比。
    周期越短， 优先级越高； 周期越长， 优先级越低。
  - 单调速率调度假定： 对于每次 CPU 执行， 周期性进程的处理时间是相同的。 也就是说， 在每次进程获取 CPU 时， 它的 CPU 执行长度是相同的。
  - 尽管是最优的， 然而单调速率调度有一个限制： CPU 的利用率是有限的， 并不总是可能完全最大化 CPU 资源。 调度 iV个进程的最坏情况下的 CPU 利用率为 N（2^(1/n)-1)
- 最早截止期限优先调度
  - 最早截止期限优先EDF) 调度根据截止期限动态分配优先级。截止期限越早， 优先级越高； 截止期限越晚，优先级越低
  - 与单调速率调度不一样， EDF 调度不要求进程应是周期的， 也不要求进程的 CPU 执行的长度是固定的。 唯一的要求是： 进程在变成可运行时， 应宣布它的截止期限。
  - EDF 调度具有吸引力的地方是： 它是理论上最佳的。然而， 在实际中， 由于进程的上下文切换和中断处理的代价， 这种级别的 CPU 利用率是不可能的
- 比例分享调度
  - 比例分享 调度程序在所有应用之间分配 T 股。 如果一个应用程序接收 N 股的时间， 那么确保了它将有 N/T的总的处理器时间。
- POSIX 实时调度

算法评估

最大化 CPU 使用率， 同时要求最大响应时间为 1 秒。
• 最大化吞吐量， 比如要求（ 平均） 周转时间与总的执行时间成正比。

- 确定性模型
  - 一种主要类别的评估方法称为分析评估法 分析评估法使用给定算法和系统负荷，生成一个公式或数字， 以便评估在该负荷下的算法性能。
  - 确定性模型为一种分析评估类型。 这种方法采用特定的预先确定的负荷， 计算在给定负荷下每个算法的性能。
- 排队模型
  - CPU 是具有就绪队列的服务器， 而 I/O 系统是具有设备队列的服务器。 已知到达率和服务率， 可以计算使用率、 平均队列长度、 平均等待时间等。 这种研究方法称为排队网络分析
- 仿真
  - 为了获得更为精确的调度算法评价， 可以使用仿真。 仿真涉及对计算机系统进行建模。软件数据结构代表了系统的主要组成部分。 仿真程序有一个代表时钟的变量； 随着这个变量值的增加， 模拟程序修改系统状态以便反映设备 、 进程和调度程序的活动。 随着仿真的运行， 表明算法性能的统计数据被收集并打印。
- 实现
  - 即使仿真的精确度也是有限的。 用于评估一个调度算法的唯一完全精确方式是：对它进行编程， 放在操作系统内， 并且观测它如何工作。 这种评价方法采用实际算法、 实际系统及实际操作条件来进行。