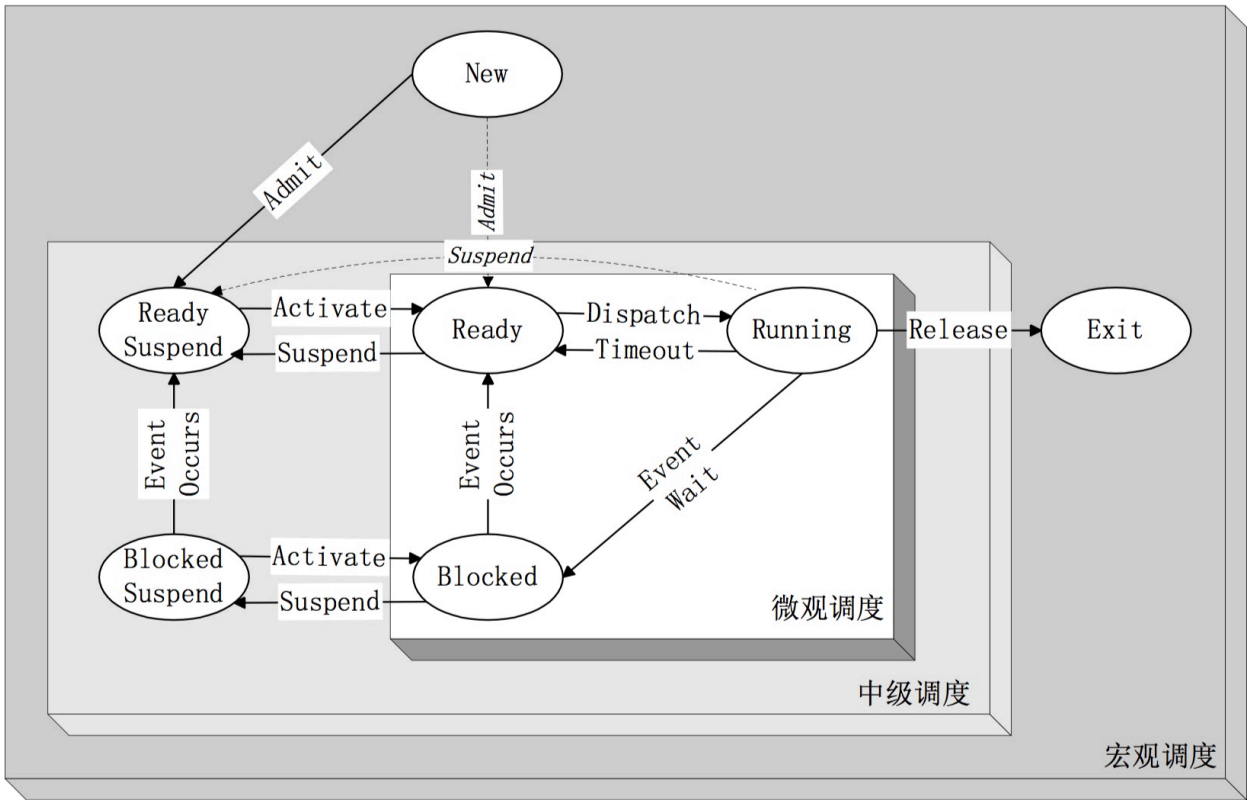
第九章 单处理器调度

1. 处理机管理
    - 处理机管理的工作是对 CPU 资源进行合理的分配使用，以提高处理机利用率，并使各用户公平地得到处理机资源。这里的主要问题是处理机**调度算法**和**调度算法特征分析**。
        - 调度类型
        - 性能原则
        - 调度算法
        - 调度算法性能分析

2. **调度的类型(scheduling)**

    从处理机调度的对象、时间、功能等不同角度，我们可把处理机调度分成不同类型。

    i. 按照**调度的层次**
        - **作业**：又称为 "**宏观调度**" 、 " 高级调度 " 。从用户工作流程的角度，一次提交的若干个流程，其中每个程序按照进程调度。时间上通常是分钟、小时或天。
        - **内外存交换**：又称为 "**中级调度**" 。从存储器资源的角度。将进程的部分或全部换出到外存上，将当前所需部分换入到内存。指令和数据必须在内存里才能被 CPU 直接访问。
        - **进程或线程**：又称为 "**微观调度** " 、 " 低级调度 " 。从 CPU 资源的角度，执行的单位。时间上通常是毫秒。因为执行频繁，要求在实现时达到高效率。
        - 处理器调度的层次



ii. 按照**调度的时间周期**

- **长期 (long-term)**：将进程投入 " 允许执行 " 进程缓冲池中，或送到 " 退出 " 进程缓冲池中。进程状态： New ->Ready suspend, Running ->Exit
- **中期 (medium-term)**：将进程的部分或全部加载到内存中。进程状态： Ready <->Ready suspend, Blocked <->Blocked suspend
- **短期 (short-term)**：选择哪个进程在处理机上执行。 进程状态： Ready <->Running
- **I/O 调度**：选择哪个 I/O 等待进程，使其请求可以被空闲的 I/O 设备进行处理。

iii. 按照OS的分类

- 批处理调度－－应用场合：大中型主机集中计算，如工程计算、理论计算（流体力学）
- 分时调度、实时调度：通常没有专门的作业调度
- 多处理机调度

3. **调度的性能准则**

> 我们可从不同的角度来判断处理机调度算法的性能，如用户的角度、处理机的角度和算法实现的角度。实际的处理机调度算法选择是一个综合的判断结果。

i. 面向用户的调度性能准则

- **周转时间**：作业从提交到完成（得到结果）所经历的时间。 包括：*在收容队列中等待， CPU上执行，就绪队列和阻塞队列中等待，结果输出等待*－－批处理系统
  - 平均周转时间 T
  - 平均带权周转时间（**带权周转时间W是T（周转)/T（CPU执行）**）

- **响应时间**：用户输入一个请求（如击键）到系统给出首次响 应（如屏幕显示）的时间－－分时系统
- **截止时间**：**开始截止时间和完成截止时间**－－实时系统，与周转时间有些相似。
- **公平性**：不因作业或进程本身的特性而使上述指标过分恶化。 如长作业等待很长时间。
- **优先级**：可以使关键任务达到更好的指标。

ii. 面向系统的调度性能准则
- 吞吐量：单位时间内所完成的作业数，跟作业本身特性和调度算法都有关系－－批处理系统
  - 平均周转时间不是吞吐量的倒数，因为并发执行的作业在时间上可以重叠。如：在 2 小时内完成 4 个作业，而每个周转时间是 1 小时，则吞吐量是 2 个作业 / 小时
- 处理机利用率：－－大中型主机
- 各种设备的均衡利用：如 CPU 繁忙的作业和 I/O 繁忙 （指次数多，每次时间短）的作业搭配－－大中型主机

iii. 调度算法本身的调度性能准则
- 易于实现
- 执行开销比

4. 进程调度

- 功能：**调度程序(dispatcher)**
  - 记录所有进程的运行状况（静态和动态）
  - 当进程出让 CPU 或调度程序剥夺执行状态进程占用的 CPU 时，选择适当的进程分派 CPU
  - 完成上下文切换
- 进程的上下文切换过程：
  - 用户态执行进程 A 代码－－进入 OS 核心（通过时钟中断或系统调用）
  - 保存进程 A 的上下文，恢复进程 B 的上下文（ CPU 寄存器和一些表格的当前指针）
  - 用户态执行进程 B 代码
- **注：上下文切换之后，指令和数据快速缓存 cache 通常需要更新，执行速度降低**

5. 调度算法

> 通常将作业或进程归入各种就绪或阻塞队列。有的算法适用于作业调度，有的算法适用于进程调度，有的两者都适应。
> **先来先服务**
> **短作业优先**
> **时间片轮转算法**

> **多级队列算法**
> **优先级算法**
> **多级反馈队列算法**

- **FCFS 算法** *First in first service*
  - 按照作业提交或进程变为就绪状态的先后次序，分派 CPU ；
  - 当前作业或进程占用 CPU ，直到执行完或阻塞， 才出让 CPU **(非抢占方式)**。
  - 在作业或进程唤醒后（如 I/O 完成），并不立即恢复执行，通常等到当前作业或进程出让 CPU 。 最简单的算法。
- FCFS 的特点
  - 比较有利于长作业，而不利于短作业。
  - **有利于 CPU 繁忙的作业**，而不利于 I/O 繁忙的作业。
- **SJF 算法** *Short job first*
  - 对预计执行时间短的作业（进程）优先分派处理机。通常后来的短作业不抢先正在执行的作业。
- SJF 的特点
  - 优点：
    - 比 FCFS 改善平均周转时间和平均带权周转时间，缩短作业的等待时间；
    - **提高系统的吞吐量；**
  - 缺点：
    - **对长作业非常不利**，可能长时间得不到执行；
    - 未能依据作业的紧迫程度来划分执行的优先级；
    - 难以准确估计作业（进程）的执行时间，从而影响调度性能。
- SJF 的变型
  - " 最短剩余时间优先 "**SRT(Shortest Remaining Time)**
    - 允许比当前进程剩余时间更短的进程来抢占
  - " 最高响应比优先 "**HRRN(Highest Response Ratio Next)**
    - 响应比 **R = (等待时间 + 要求执行时间) / 要求执行时间**
  - 是 FCFS 和 SJF 的折衷

- **时间片轮转(Round Robin) 算法**

> 前两种算法主要用于**宏观调度**，说明怎样选择一个进程或作业开始运行，开始运行后的作法都相同， 即运行到结束或阻塞，阻塞结束时等待当前进程放弃 CPU 。本算法主要用于**微观调度**，说明怎样并发运行，即切换的方式；设计目标是提高资源利用率。
> 其基本思路是通过时间片轮转，提高进程并发性和响应时间特性，从而提高资源利用率；

- 将系统中所有的就绪进程按照FCFS原则排成一个队列。
- 每次调度时将 CPU 分派给队首进程，让其执行一个时间片。时间片的长度从几个 ms 到几百 ms 。
- 在一个时间片结束时，发生时钟中断。
- 调度程序据此暂停当前进程的执行，将其送到就绪队列的末尾，并通过上下文切换执行当前的队首进程。
- 进程可以未使用完一个时间片，就出让 CPU（如阻塞）。

- 时间片长度的确定

  - 时间片长度变化的影响
    - 过长－ > 退化为 FCFS 算法，进程在一个时间片内都执行完，响应时间长。
    - 过短－ > 用户的一次请求需要多个时间片才能处理完，上下文切换次数增加，响应时间长。
  - 对响应时间的要求：
    - **T(响应时间)=N(进程数目)*q(时间片)**
  - 时间片长度的影响因素：
    - 就绪进程的数目：数目越多，时间片越小（当响应时间一定时）
    - 系统的处理能力：应当使用户输入通常在一个时间片内能处理完，否则使响应时间，平均周转时间和平均带权周转时间延长。

- 多级队列算法 (Multiple-level Queue)

  > 本算法引入多个就绪队列，通过各队列的区别对待，达到一个综合的调度目标；

  - 根据作业或进程的性质或类型的不同，将就绪队列再分为若干个子队列。
  - 每个作业固定归入一个队列。
  - 各队列的不同处理：不同队列可有不同的优先级、 时间片长度、调度策略等。如：系统进程、用户交互进程、批处理进程等。

- 优先级算法 (Priority Scheduling)

  > 本算法是多级队列算法的改进，平衡各进程对响应 时间的要求。适用于作业调度和进程调度，可分成 抢先式和非抢先式；

  ```
  静态优先级
  动态优先级
  线性优先级调度算法（ SRR, Selfish Round Robin ）
  ```

  - 静态优先级

> 创建进程时就确定，直到进程终止前都不改变。通常是一个整数。

  - 依据：
    - 进程类型（系统进程优先级较高）
    - 对资源的需求（对 CPU 和内存需求较少的进程， 优先级较高）
    - 用户要求（紧迫程度和付费多少）

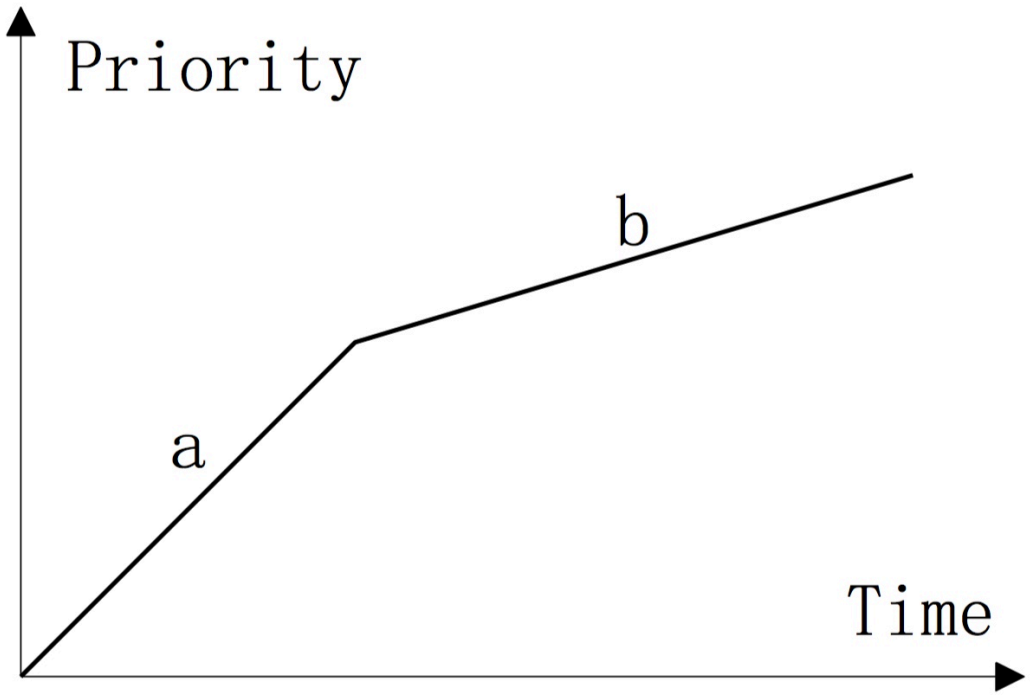
- 动态优先级

> 在创建进程时赋予的优先级，在进程运行过程中可以自动改变，以便获得更好的调度性能。如：

  - 在就绪队列中，**等待时间延长则优先级提高**，从而使优先级较低的进程在等待足够的时间后，其优先级提高到可被调度执行；
  - 进程每执行一个时间片，就降低其优先级，从而一个进程持续执行时，其优先级降低到出让 CPU 。

- **线性优先级调度算法 （ SRR, Selfish Round Robin ）**

  - 本算法是优先级算法的一个实例，它通过进程优先级的递增来改进长执行时间进程的周转时间特征。
  - SRR算法
    - 就绪进程队列分成两个：
      - **新创建进程队列**：按 FCFS 方式排成；进程优先级按速率 a 增加；
      - **享受服务队列**：已得到过时间片服务的进程按 FCFS 方式 排成；进程优先级按速率 b 增加；
    - 新创建进程等待时间的确定：**当新创建进程优先级与享受服务队列中最后一个进程优先级相同时**，转入享受服务队列；
    - SSR算法优先级的变化



- SRR 算法与 FCFS 算法和时间片轮转算法的关系
  - **当 b>a>0 时，享受服务队列中永远只有一个进程； SRR 算法退化成 FCFS 算法；**
  - **当 a>b=0 时， SRR 算法就是时间片轮转算法；**

- 多级反馈队列算法 (Round Robin with Multiple Feedback)
  - 多级反馈队列算法是时间片轮转算法和优先级算法的综合和发展。优点：
    - 为提高系统吞吐量和缩短平均周转时间而照 顾短进程
    - 为获得较好的 I/O 设备利用率和缩短响应时间而照顾 I/O 型进程
    - 不必估计进程的执行时间，动态调节
  - 设置多个就绪队列，分别赋予不同的优先级，如逐级降低，队列 1 的优先级最高。每个队列执行时间片的长度也不同，规定优先级越低则时间片越长，如逐级加倍

- 具体实现
  - 新进程进入内存后，先投入队列 1 的末尾，按 FCFS 算法调度；若按队列 1 一个时间片未能执行完，则降 低投入到队列 2 的末尾，同样按 FCFS 算法调度；如此下去，降低到最后的队列，则按 " 时间片轮转 " 算法调度直到完成。
  - 仅当较高优先级的队列为空，才调度较低优先级的队列中的进程执行。如果进程执行时有新进程进入 较高优先级的队列，则抢先执行新进程，并把被抢先的进程投入原队列的末尾。

- 几点说明

        - I/O 型进程：让其进入最高优先级队列，以及时响应 I/O 交互。通常执行一个小时间片，要求可处理完一 次 I/O 请求的数据，然后转入到阻塞队列。
        - 计算型进程：每次都执行完时间片，进入更低级队 列。最终采用最大时间片来执行，减少调度次数。
        - I/O 次数不多，而主要是 CPU 处理的进程：在 I/O 完 成后，放回优先 I/O 请求时离开的队列，以免每次都 回到最高优先级队列后再逐次下降。
        - 为适应一个进程在不同时间段的运行特点， I/O 完成 时，提高优先级；时间片用完时，降低优先级；

6. 调度算法性能分析

    调度算法的性能通常是通过实验或计算得到的。

    - FCFS, Round Robin, 线性优先级算法 SRR(Selfish Round Robin)
    - 周转时间
        - 长作业时： T(FCFS) < T(SRR) < T(RR) （运行时间是主要因素）
        - 短作业时： T(RR) < T(SRR) < T(FCFS) （等待时间是主要因素）