第4章 中断与处理机调度

在多道程序环境下，主存中存在着多个进程，其数目往往多于处理机的数目。这就要求系统能够按照某种算法，动态地将处理机分配给处于就绪状态的一个进程，并使之运行。分配处理机的任务是由处理机调度程序完成的。

调度问题是操作系统设计的核心问题。

主要内容

- 4.1中断概述
- 4.2三级调度体系
- 4.3进程调度目标和调度方式
- 4.4调度算法的评价准则
- 4.5进程调度算法
- 4.6线程的调度
- 4.8本章小结

本章要点

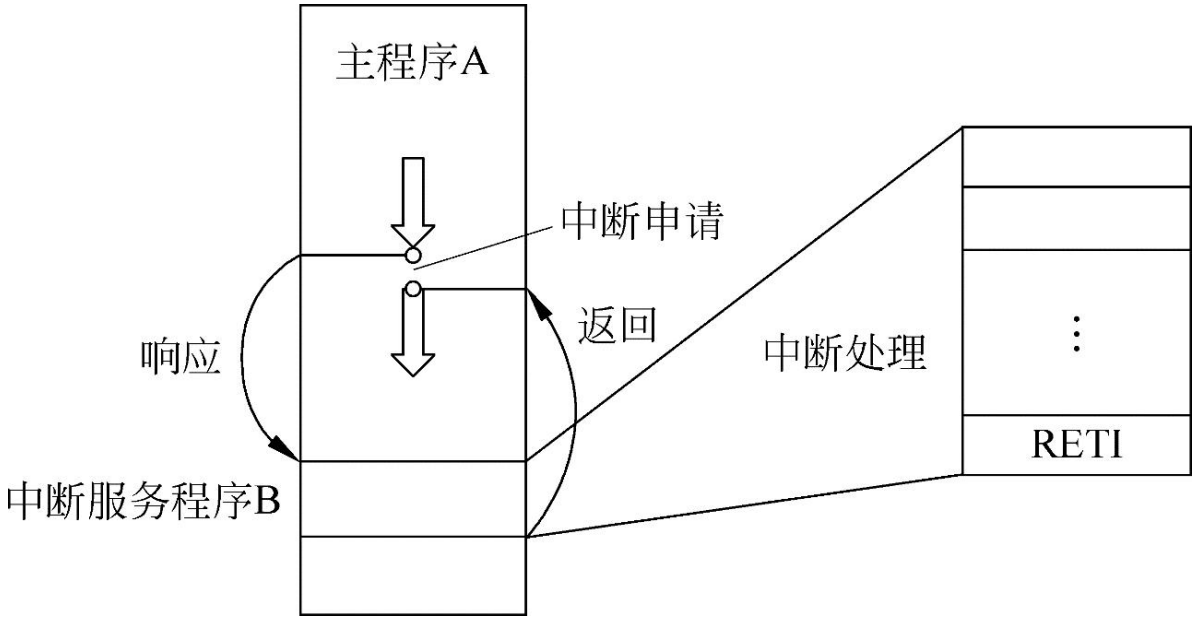
- 本章首先介绍中断的基本概念，接着阐述调度的类型与方式，并且重点介绍常用的的进程调度算法，最后介绍线程调度。本章重点掌握以下要点：
- 了解中断技术的基本概念；了解调度算法的评价准则；了解线程的调度的实现方式。
- 理解中断的处理过程；理解进程调度的目标和调度方式。
- 掌握处理机调度的类型与方式；掌握常用进程调度算法及其特点。

4.1 中断概述

- 中断在操作系统中有着特殊重要的地位，它是多道程序得以实现的基础，没有中断，就不可能实现多道程序，因为进程之间的切换是通过中断来完成。中断能充分发挥处理机的使用效率，提供高系统的实时处理能力。

4.1.1 中断的概念

- 中断是指CPU对系统发生的某个事件作出的一种反应，它使CPU暂停正在执行的程序，保留现场后自动执行相应的处理程序，处理该事件后，如被中断进程的优先级最高，则返回断点继续执行被“打断”程序。



4.1.1 中断的概念

- 引起中断的事件或发出中断请求的来源称为**中断源**。中断源向CPU提出的处理请求称为**中断请求**。发生中断时，被打断程序的暂停点称为**断点**。
- 中断的分类有外中断和内中断。

4.1.2 中断优先级和中断屏蔽

- 中断优先级
  - 为使系统能及时响应并处理发生的所有中断，系统根据引起中断事件的重要性和紧迫程度，硬件将中断源分为若干个级别，称作**中断优先级**。
  - 一般情况下，优先级的高低顺序依次为：**硬件故障中断**、**自愿中断**、**程序性中断**、**外部中断**和**输入输出中断**。

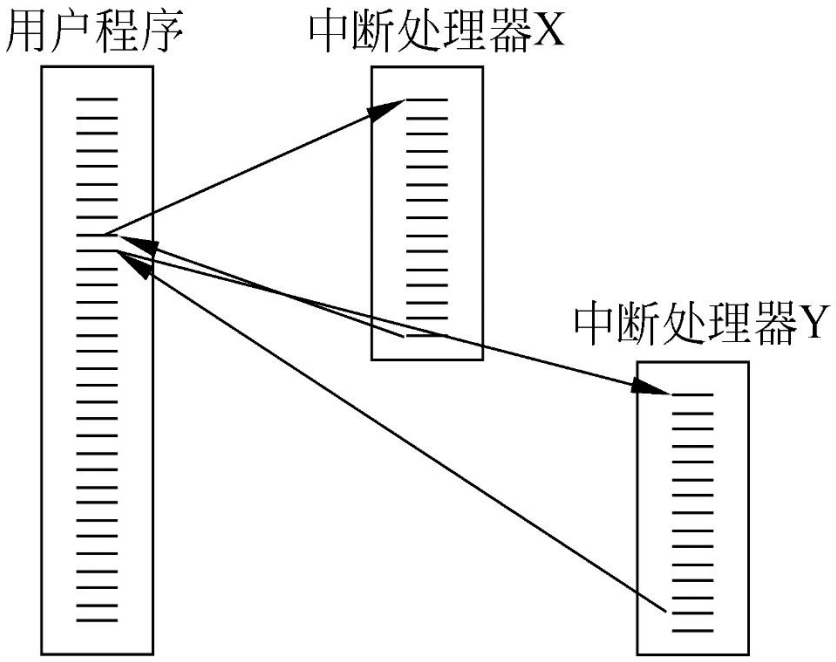
4.1.2 中断优先级和中断屏蔽

- 中断屏蔽
  - 为了防止低优先级的中断事件处理打断优先级高的中断事件的处理，以及防止中断多重嵌套处理，计算机系统采用中断屏蔽技术。中断屏蔽根据是否可以被屏蔽，可将中断分为两大类：不可屏蔽中断(又叫非屏蔽中断)和可屏蔽中断。不可屏蔽中断源一旦提出请求，CPU必须无条件响应，而对可屏蔽中断源的请求，CPU可以响应，也可以不响应。

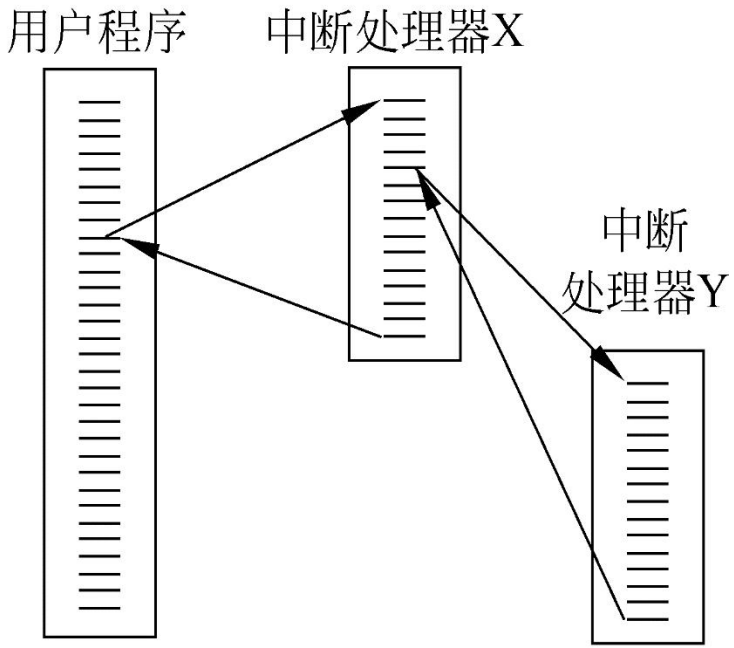
4.1.2 中断优先级和中断屏蔽

- 对中断的处理方式
  - 顺序处理方式
    - 当处理机正在处理一个中断时，将屏蔽掉所有的中断，即处理机对任何新到的中断请求，都暂时不予理睬，而让它们等待。直到处理机已完成本次中断的处理后，处理机再去检查是否有中断发生。若有，再去处理新到的中断，若无，则返回被中断的程序。在该方法中，所有中断都将按顺序依次处理。
  - 嵌套处理方式
    - 在设置了中断优先级的系统中，通常按这样的规则来进行优先级控制：
    - （1） 当同时有多个不同优先级的中断请求时，处理机优先响应最高优先级的中断请求；
    - （2） 高优先级的中断请求，可以抢占正在运行低优先级中断的处理机，该方式类似于基于优先级的抢占式进程调度。

4.1.2 中断优先级和中断屏蔽



(a) 顺序中断处理

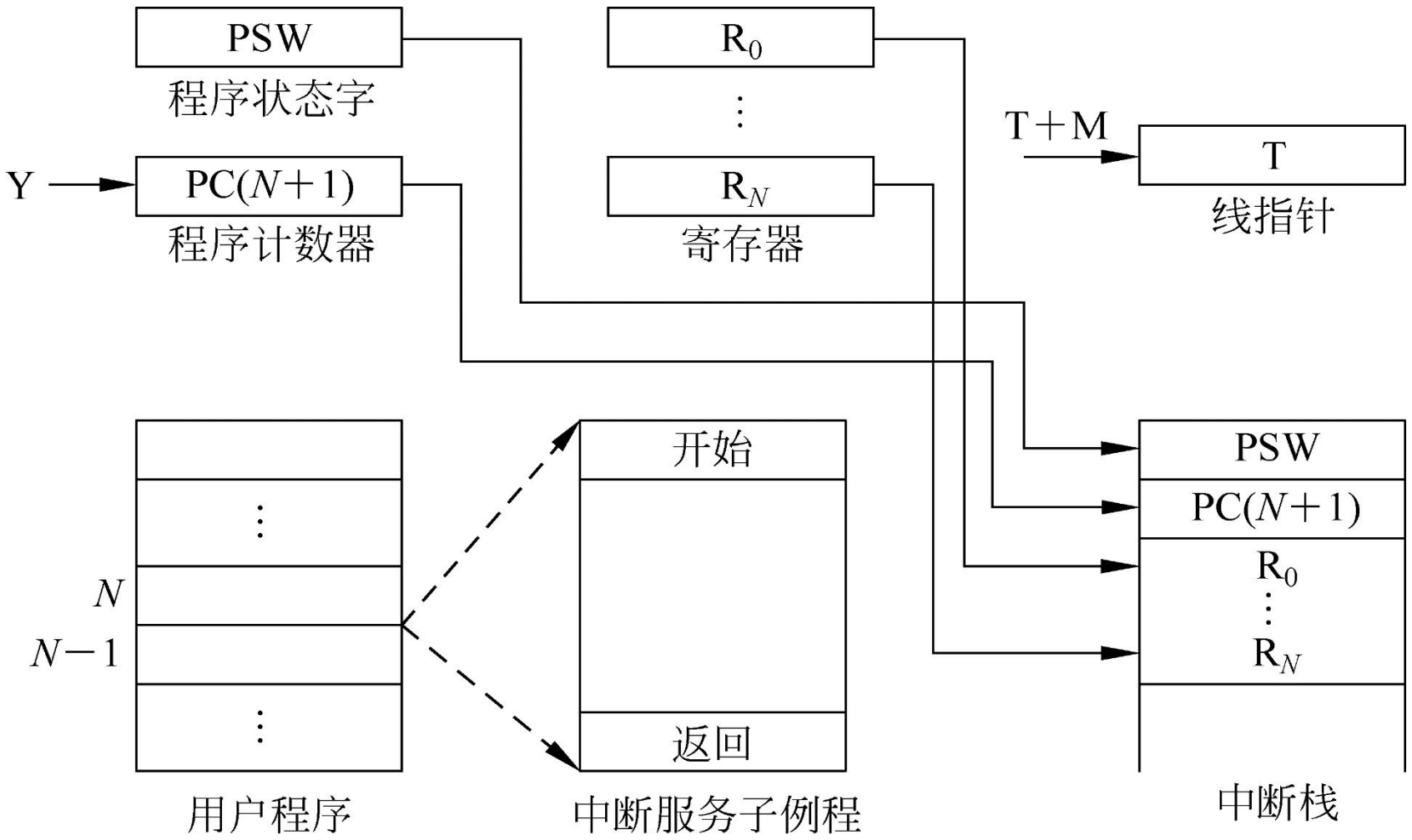


(b) 嵌套中断处理

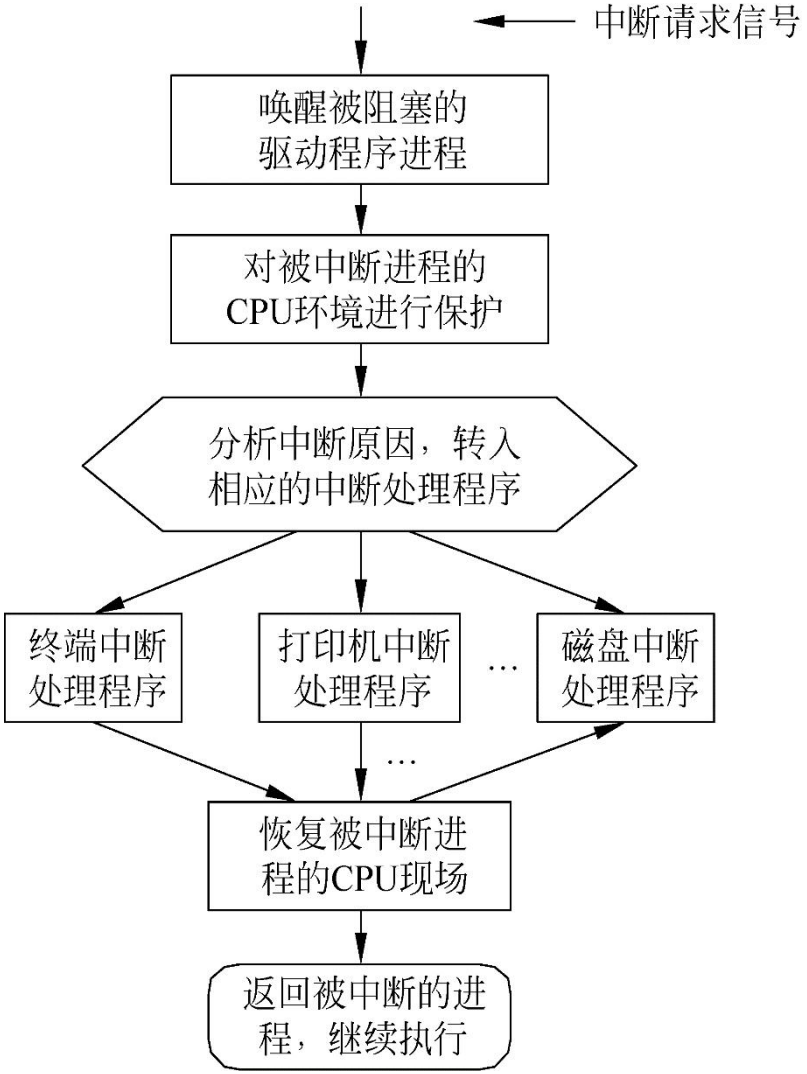
4.1.3 中断处理过程

1. 唤醒被阻塞的驱动（程序）进程
2. 保护被中断进程的CPU环境
3. 转入相应的设备处理程序
4. 中断处理
5. 恢复被中断进程的现场

4.1.3 中断处理过程


PSW
程序状态字
Y
PC(N+1)
程序计数器
R0
RN
寄存器
T+M
T
线指针
N
N−1
开始
返回
用户程序
中断服务子例程
PSW
PC(N+1)
R0
RN
中断栈

4.1.3 中断处理过程


中断请求信号
唤醒被阻塞的
驱动程序进程
对被中断进程的
CPU环境进行保护
分析中断原因，转入
相应的中断处理程序
终端中断
处理程序
打印机中断
处理程序
…
磁盘中断
处理程序
…
恢复被中断进
程的CPU现场
返回被中断的进
程，继续执行

4.2 三级调度体系

- 调度的作用
  - 处理机调度的主要目的就是分配处理机。
  - 调度的功能是组织和维护就绪进程队列。包括确定调度算法、按调度算法组织和维护就绪进程队列。
  - 分派的功能是指当处理机空闲时，从就绪队列队首中移出一个PCB，并将该进程投入运行。

- 习惯上往往把上述功能统称为进程调度。
- 调度程序还要关注CPU的利用效率。

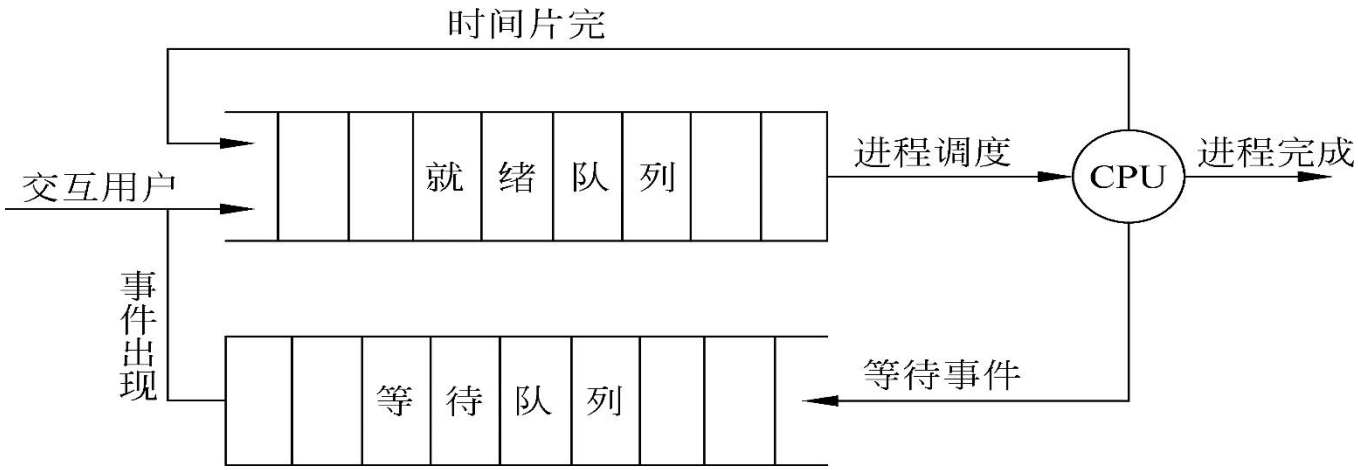
4.2 三级调度体系

- 调度级别
  - 分为高级调度、中级调度和低级调度

  - 高级调度：又称作业调度或长期调度
  - 中级调度：又称内存调度或中期调度
  - 低级调度：又称进程调度或短期调度

4.2.1 低级调度

- 低级调度也称为进程调度或短程调度，它所调度的对象是进程（内核级线程）。其主要功能是根据某种算法，决定就绪队列中的哪个进程获得处理机，并由分派程序把处理机分配给被选中的进程。**进程调度是最基本的一种调度**。执行频率几毫秒钟或几十毫秒一次。



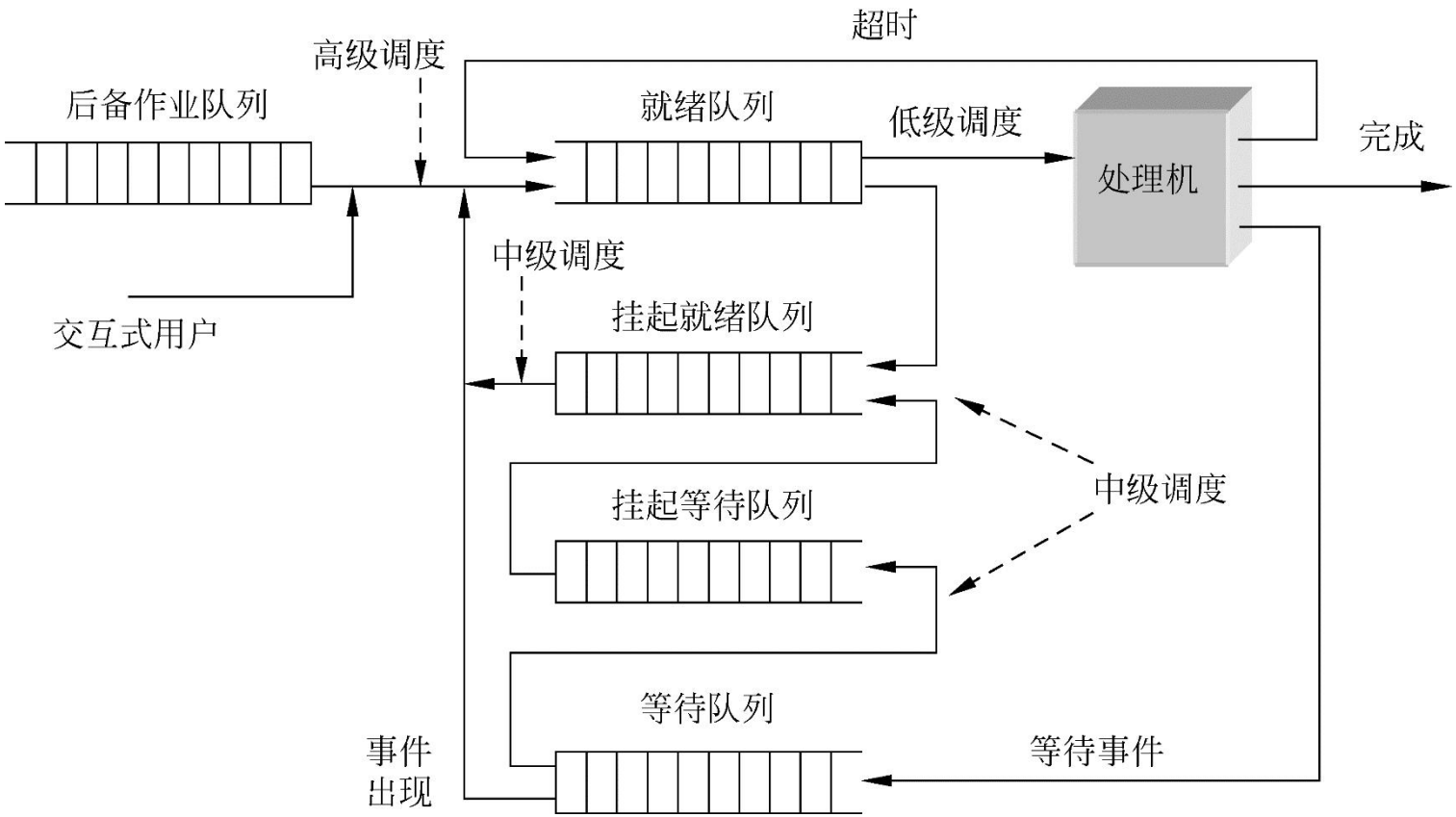
4.2.2 中级调度

- 中级调度也称为内存调度。它主要目的提高内存利用率和系统吞吐量。负责进程在内存和外存对换区之间换进/换出。是内存对换功能的一部分。执行频率介于作业调度和进程调度之间。

4.2.3 高级调度

- 高级调度又称作业调度或长程调度，主要功能是根据某种算法，把外存上处于后备队列中的那些作业调入内存，也就是说，**高级调度的主要对象是作业。执行频率几分钟一次**。

4.2.4 三级调度关系


超时
高级调度
后备作业队列
就绪队列
低级调度
处理机
完成
中级调度
交互式用户
挂起就绪队列
中级调度
挂起等待队列
等待队列
事件
出现
等待事件

4.3 进程调度目标和调度方式

- **4.3.1进程调度目标**
  - **公平性**。保证每个进程得到合理的处理机时机和执行速度。比如，不能由于采用某种调度算法而使得某些进程长时间得不到处理机的执行，出现“饥饿”现象。要在保证某些进程优先权的基础上，最大限度地实现进程执行的公平性。
  - **高效率**。保证处理机得到充分利用，不让处理机由于空闲等待而浪费大量的时间，力争使处理机的绝大部分时间都在“忙碌”地执行有效指令。其中处理机的利用率可以用以下方法计算：

$$\text{CPU 的利用率} = \frac{\text{CPU 有效工作时间}}{\text{CPU 有效工作时间} + \text{CPU 空闲等待时间}}$$

4.3 进程调度目标和调度方式

- **4.3.1进程调度目标**
  - **平衡性**。由于在系统中可能具有多种类型的进程，有的属于计算型作业，有的属于I/O型。为使系统中的CPU和各种外部设备都能经常处于忙碌状态，调度算法应尽可能保持系统资源使用的平衡性。
  - **高吞吐量**。要实现系统的高吞吐量，减小每个进程的等待时间。
  - **策略强制执行**。对所制订的策略其中包括安全策略，只要学要，就必须予以准确地执行，即使会造成某些工作的延迟也要执行。

4.3 进程调度目标和调度方式

- **4.3.1进程调度目标**
  - 不同类型的操作系统有不同的调度目标。下面介绍一下常见操作系统的调度目标。设计操作系统时，设计者选择哪些调度目标在很大程度上取决于操作系统自身的特点。
  - 多道批处理系统。多道批处理系统强调高效利用系统资源、系统吞吐量大和平均周转时间短。进程提交给处理机后就不再与外部进行交互，系统按照调度策略安排它们运行，直到诸进程完成为止。

4.3 进程调度目标和调度方式

- **4.3.1进程调度目标**
  - **分时操作系统**。分时系统更关心多个用户的公平性和及时响应性，它不允许某个进程长时间占用处理机。分时系统多采用时间片轮转调度算法或在其基础上改进的其他调度算法。但处理机在各个进程之间的频繁切换会增加系统时空开销，延长各个进程在系统中的存在时间。分时系统最关注的是交互性和各个进程的均衡性，对进程的执行效率和系统开销并不苛刻。
  - **实时操作系统**。实时系统必须保证实时进程的请求得到及时响应，往往不考虑处理机的使用效率。实时系统采取的调度算法和其他类型系统采取的调度算法相比有很大不同，其调度算法的最大特点是可抢占性。
  - **通用操作系统**。通用操作系统中，对进程调度没有特殊限制和要求，选择进程调度算法时主要追求处理机的使用公平性以及各类资源使用的均衡性。

4.3.2 进程调度方式

- **非抢占方式(Nonpreemptive Mode)**
- 在采用这种调度方式时一旦把处理机分配给某进程后，就一直让它运行下去，决不会因为时钟中断或任何其它原因去抢占当前正在运行进程的处理机，直至该进程完成，或发生某事件而被阻塞时，才把处理机分配给其它进程。

- **优点**：实现简单、系统开销小，适用于大多数的批处理系统环境。
- **缺点**：难以满足紧急任务的要求。在要求比较严格的实时系统中，不宜采用这种调度方式。

4.3.2 进程调度方式

- **抢占方式(Preemptive Mode)**
- 这种调度方式允许调度程序根据某种原则，去暂停某个正在执行的进程，将已分配给该进程的处理机重新分配给另一进程。

- 抢占原则
  - 优先权原则
  - 短进程优先原则
  - 时间片原则

4.4 调度算法的评价准则

- 操作系统设计者在设计进程调度算法时，往往有很多种调度算法可供选择，哪种方法更优秀，必须有一个明确的评价准则。
- 通常可以从用户角度、系统角度和调度算法实现角度来考察算法的优劣，经过综合考虑做出最终判断。

4.4.1 面向用户的评价准则

- 平均周转时间短。周转时间是指从作业被提交给系统开始，到作业完成为止这段时间间隔，即完成时间减去到达时间。
- 平均周转时间描述为：$$T = \frac{1}{n}\left[\sum_{i=1}^{i} T_i\right]$$
- 作业的周转时间T与系统为它提供服务的时间（运行时间）T_S之比，即$W=T/T_S$，称为带权周转时间，而平均带权周转时间则可表示为：

$$W = \frac{1}{n}\left[\sum_{i=1}^{n} \frac{T_i}{T_{Si}}\right]$$

4.4.1 面向用户的评价准则

- **响应时间快**。响应时间是指从进程输入第一个请求到系统给出首次响应的时间间隔。用户请求的响应时间越短，用户的满意度越高。响应时间通常由三部分时间组成：进程请求传送到处理机的时间、处理机对请求信息进行处理的时间、响应信息回送到显示器的时间。其中，第一、三部分时间很难减少，只能通过合理的调度算法缩短第二部分时间。

4.4.1 面向用户的评价准则

- **截止时间的保证**。截止时间是指用户或其他系统对运行进程可容忍的最大延迟时间。在实时系统中，通常用该准则衡量一个调度算法是否合格。在实际系统评价中，主要考核的是开始截止时间和完成截止时间。

- **优先权准则**。在批处理、分时和实时系统中选择调度算法时，为保证某些紧急作业得到及时处理，必须遵循优先权准则。因此，系统对不同进程设立优先级，高优先级进程优先获得处理机的使用权。

4.4.2 面向系统的评价准则

- **系统吞吐量**：单位时间内完成的进程数目
- **处理机利用率**：CPU有效工作时间与总的运行时间之比
- **各类资源均衡使用**：资源的均衡利用
- **调度算法实现准则**：有效性和易实现性

4.5 进程调度算法

- 作业调度算法和进程调度算法非常相似，对进程调度算法稍加改动就可以转换为作业调度。

- 进程调度的主要问题是采用某种调度算法能合理、有效地把处理机分配给各个进程。

4.5.1 先来先服务调度法FCFS

- FCFS: First Come First Service
- 实现思想：排队买票
  - 按照进程进入就绪的先后顺序选择可以占用处理机的进程。当有进程就绪时，把该进程排入就绪队列的末尾，而进程调度总是把处理机分配给就绪队列中的第一个进程。一旦一个进程占用了处理机，它就一直执行下去，直到因等待某事件或进程完成了工作才让出了处理机分配给其他进程。

4.5.1 先来先服务法FCFS

- 非抢占式
- 优点
  - 简单，易于理解，容易实现。
  - 有利于长作业（进程）,有利于CPU繁忙型作业（进程）。
- 缺点
  - 不利于短作业（进程），不利于I/O繁忙型作业（进程）。
  - 效率较低。
- 适用于作业调度、进程调度。通常与其他算法结合起来使用。

4.5.1 先来先服务法FCFS

- 【例1】，系统中有5个进程P_1、P_2、P_3、P_4和P_5并发执行，5个并发进程的创建时间、执行时间、优先级以及时间片个数如下表所示：

进程名	进程创建时间	要求执行时间	优先级	时间片个数
P_1	0	3	3	3
P_2	1	6	5	6
P_3	2	1	1	1
P_4	3	4	4	4
P_5	4	2	2	2

4.5.1 先来先服务法FCFS

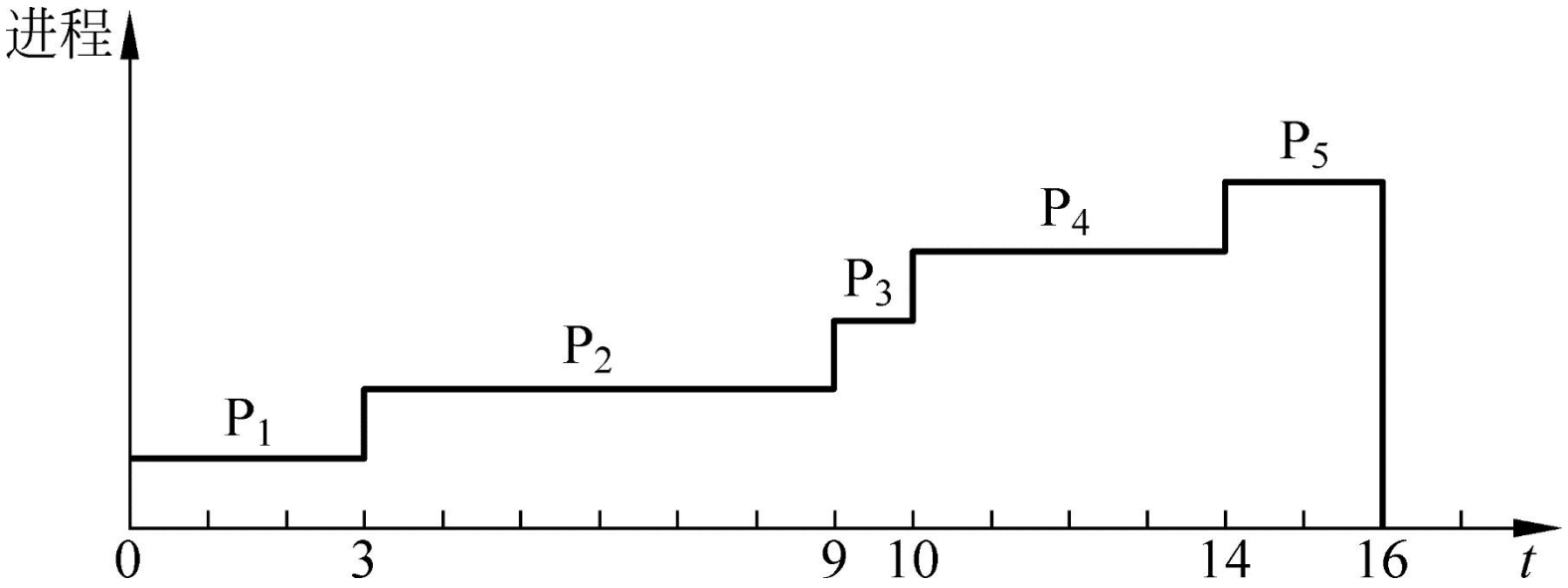


图4-7 先来先服务调度算法时序图

FCFS调度算法性能指标

表4-3 先来先服务调度的评价结果

进程	创建时间	运行时间	开始时间	完成时间	周转时间	带权周转时间
P_1	0	3	0	3	3	1
P_2	1	6	3	9	8	1.33
P_3	2	1	9	10	8	8
P_4	3	4	10	14	11	2.75
P_5	4	2	14	16	12	6
平均周转时间 T =8.4 带权平均周转时间 W =3.82					42	19.08

4.5.2 短进程优先调度算法SPF

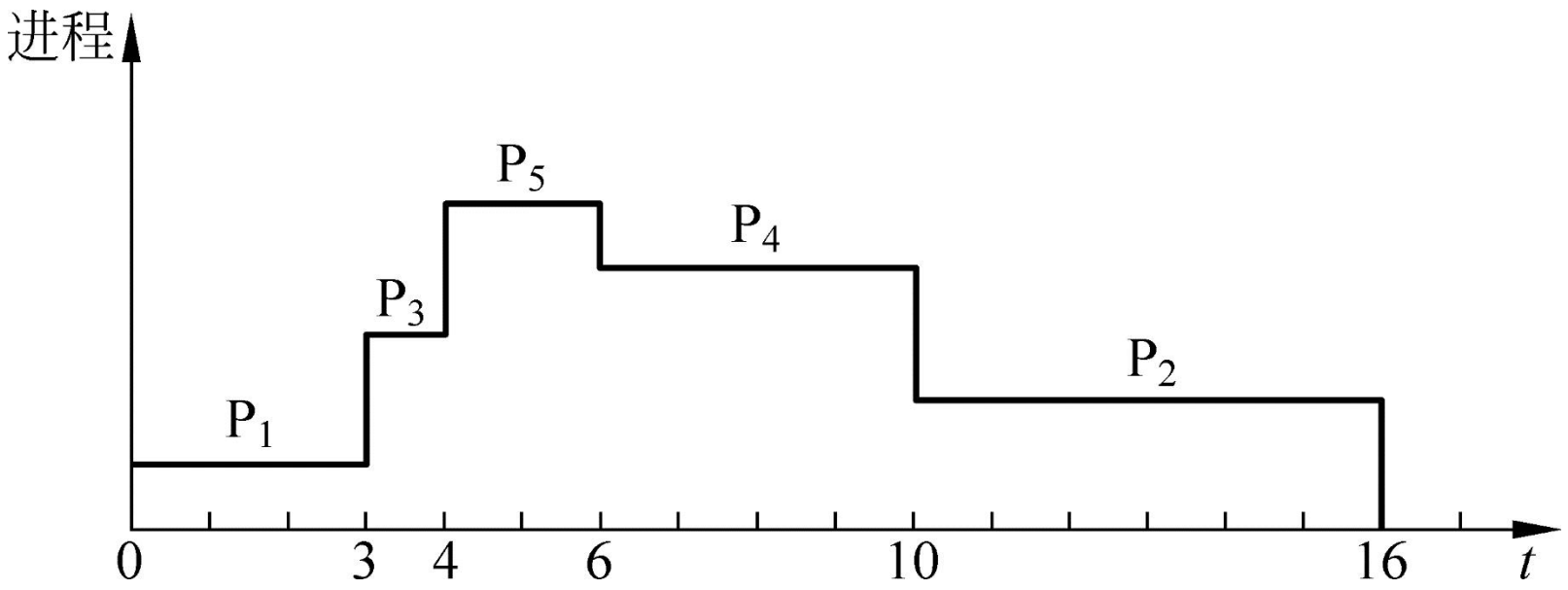
- **短进程优先（Short Process First，SPF）**
  - 所谓进程的长短是指进程要求运行时间的多少。
  - 当分派时，SPF算法把CPU优先分给最短的进程。
- 实现思想
  - 当分配CPU时，选择所需处理时间最短的进程。短进程将越过长进程，跳到队列头。一个进程一旦分得处理机，便执行下去，直到该进程完成或阻塞时，才释放处理机。
- 这是对FCFS算法的改进，其目标是减少平均周转时间。

4.5.2 短进程优先SPF

- 非抢占式
- 优缺点
  - 对于一组给定的进程，SPF算法能给出较小的平均等待时间，提高了系统的吞吐量。
  - 对长进程非常不利，使进程的周时间明显地增长，有可能出现“饥饿”现象。
  - 不能保证紧迫性进程会被及时处理。
  - 进程的运行时间很难精确估计，进程在运行前不一定能真正做到短进程被优先调度。
  - 在采用该算法时，人-机无法实现交互。

4.5.2 短进程优先SPF

- 【例2】对FCFS算法中的实例采用SPF调度算法重新调度。



4.5.2 短进程优先SPF

表4-4 短进程优先调度的评价结果

进程	创建时间	运行时间	开始时间	完成时间	周转时间	带权周转时间
P_1	0	3	0	3	3	1
P_2	1	6	10	16	15	2.5
P_3	2	1	3	4	2	2
P_4	3	4	6	10	7	1.75
P_5	4	2	4	6	2	1
平均周转时间 T =5.8 带权平均周转时间 W =1.65					29	8.25

4.5.3 最短剩余时间优先法SRTF

- SRTF：Shortest Remaning Time First
- 实现思想
  - 当新进程进入就绪队列时，如果它需要的运行时间比当前运行的进程所需的剩余时间还短，则执行切换，当前运行进程被剥夺CPU的控制权，使新进程获得CPU并运行。

4.5.3 最短剩余时间优先法SRTF

- 抢占式
- 优点
  - 能保证新的短进程一进入系统就能很快得到服务，平均等待时间短。
- 缺点
  - 需要不断统计进程剩余时间且进程切换较为频繁，系统开销较大。

4.5.3 最短剩余时间优先法SRTF

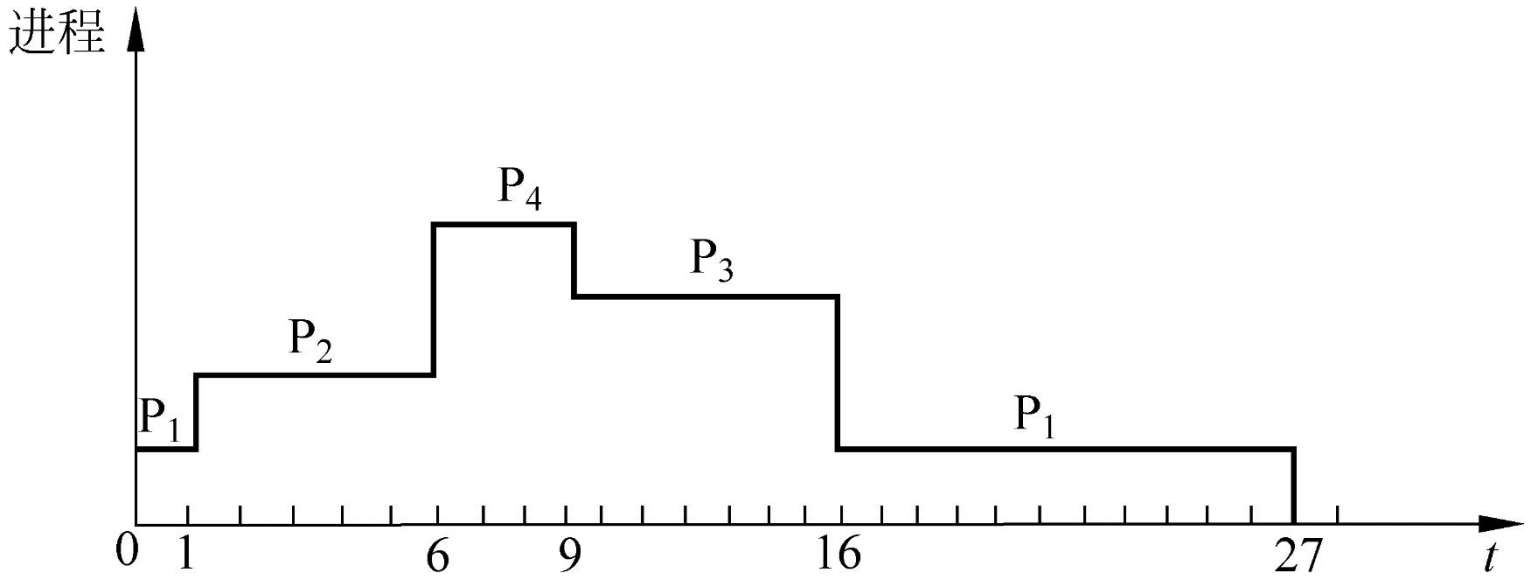
- 【例3】P1、P2、P3、P4四个进程到达系统的时间和运行时间如表4-5所示。请采用最短剩余时间优先调度算法求出各个进程的执行情况，并计算平均周转时间和平均带权周转时间。

表4-5 最短剩余时间优先各进程到达时间和运行时间

进程	到达时间	运行时间
P_1	0	12
P_2	1	5
P_3	3	7
P_4	5	3

4.5.3 最短剩余时间优先法SRTF

- 【例3】P_1、P_2、P_3、P_4四个进程到达系统的时间和运行时间如表4-5所示。请采用最短剩余时间优先调度算法求出各个进程的执行情况，并计算平均周转时间和平均带权周转时间。



4.5.3 最短剩余时间优先法SRTF

- **采用最短剩余时间优先调度算法，每个进程的周转时间和带权周转时间如表4-6所示。**

表4-6 最短剩余时间优先调度的评价结果

进程	到达时间	运行时间	开始时间	完成时间	周转时间	带权周转时间
P_1	0	12	0	27	27	2.25
P_2	1	5	1	6	5	1
P_3	3	7	9	16	13	1.86
P_4	5	3	6	9	4	1.33
平均周转时间 T =12.25 带权平均周转时间 W =1.61					49	6.44

4.5.4 时间片轮转调度算法RR

- RR：Round Robin
- 实现思想
  - 依据公平服务的原则，将处理机的运行时间划分成等长的时间片，轮转式分配给各个就绪进程使用。采用此算法的系统中，所有就绪进程按照先来先服务的原则排成一个队列，每次调度时将处理机分派给队首进程。如果进程在一个时间片内没执行完，那么调度程序强行将该进程中止，进程由执行态变为就绪态并把处理机分配给下一个就绪进程。该算法能保证就绪队列中的所有进程在一给定的时间段内均能获得处理机运行。

4.5.4 时间片轮转调度算法RR

- 进程切换时机
  - 若一个时间片尚未用完，正在运行的进程便已经完成，就立即激活调度程序，调度就绪队列中队首进程运行。
  - 在一个时间片用完时，计时器中断处理程序被激活，如果进程尚未运行完毕，调度程序将把它送往就绪队列末尾。
- 时间片的大小
  - 过长：退化为先来先服务
  - 过短：进程切换次数大大增加，系统开销大
  - 一般：$q=R/N$

4.5.4 时间片轮转调度算法RR

- 【例4】仍然采用FCFS算法中的实例，改用时间片轮转调度算法对其重新调度。5个进程分别需要运行3、6、1、4、2个时间片，5个并发执行进程的运行时序图如图4-9所示。

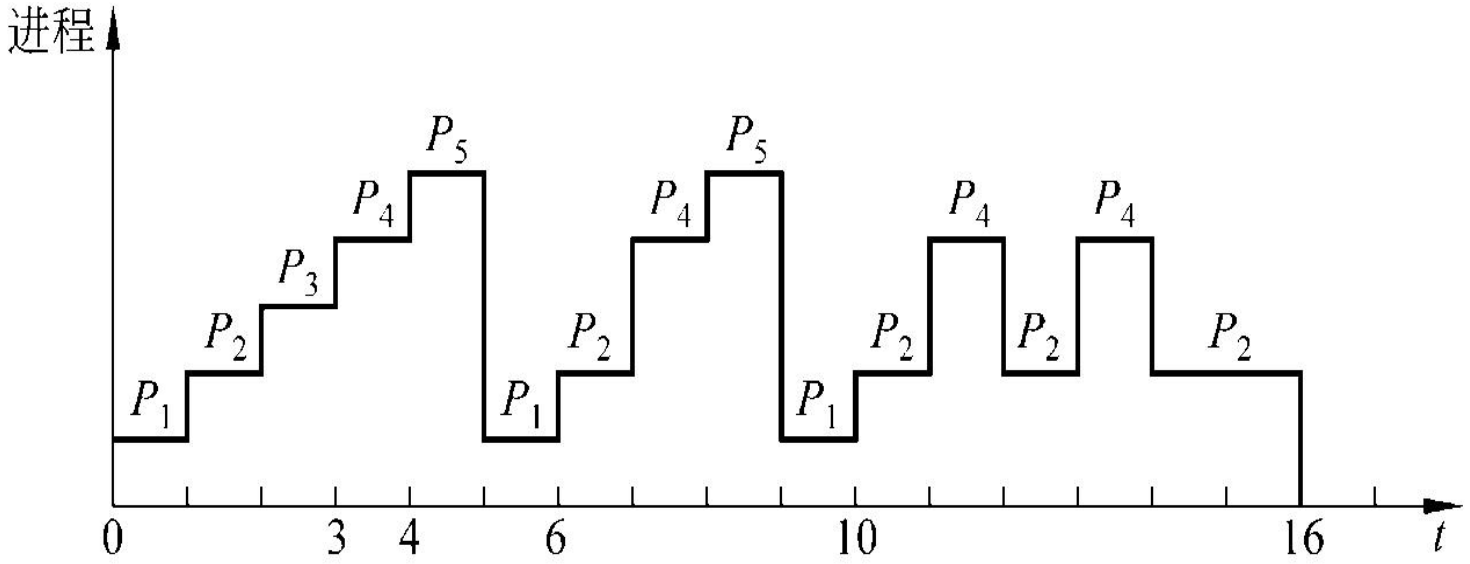


图4-9 时间片轮转调度算法时序图

4.5.4 时间片轮转调度算法RR

- 采用时间片轮转调度算法，每个进程的完成时间、周转时间和带权周转时间如表4-7所示。

表4-7 时间片轮转调度的评价结果

进程	创建时间	运行时间	开始时间	完成时间	周转时间	带权周转时间
P_1	0	3	0	10	10	3.33
P_2	1	6	1	16	15	2.5
P_3	2	1	2	3	1	1
P_4	3	4	3	14	11	2.75
P_5	4	2	4	9	5	2.5
平均周转时间 T =8.4 带权平均周转时间 W =2.42					42	12.08

4.5.5 优先级调度算法PS

- **优先级法（Priority Scheduling，PS）**
  - 从就绪队列中选出优先级最高的进程，让它在CPU上运行。
- 处理方式
  - 非抢占式优先级法
  - 抢占式优先级法
- 优先级的确定
  - 优先级确定：可由系统内部定义或由外部指定
  - 确定进程优先级的方式——静态与动态。

4.5.5 优先级调度算法PS

- 静态优先级
  - 在创建进程时就确定下来的，而且在进程的整个运行期间保持不变。
  - 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）

4.5.5 优先级调度算法PS

- 动态优先级
  - 随进程的推进而不断改变。
  - 在就绪队列中，等待时间延长则优先级提高。(解决饥饿问题)
  - 进程每执行一个时间片，就降低其优先级。(实现负反馈，防止长期占用CPU)

4.5.5 优先级调度算法PS

- 【例5】仍然采用FCFS算法中的实例，分别改用不可抢占静态优先级调度算法和可抢占静态优先级调度算法重新调度。不可抢占静态优先级调度算法具体分析如下：

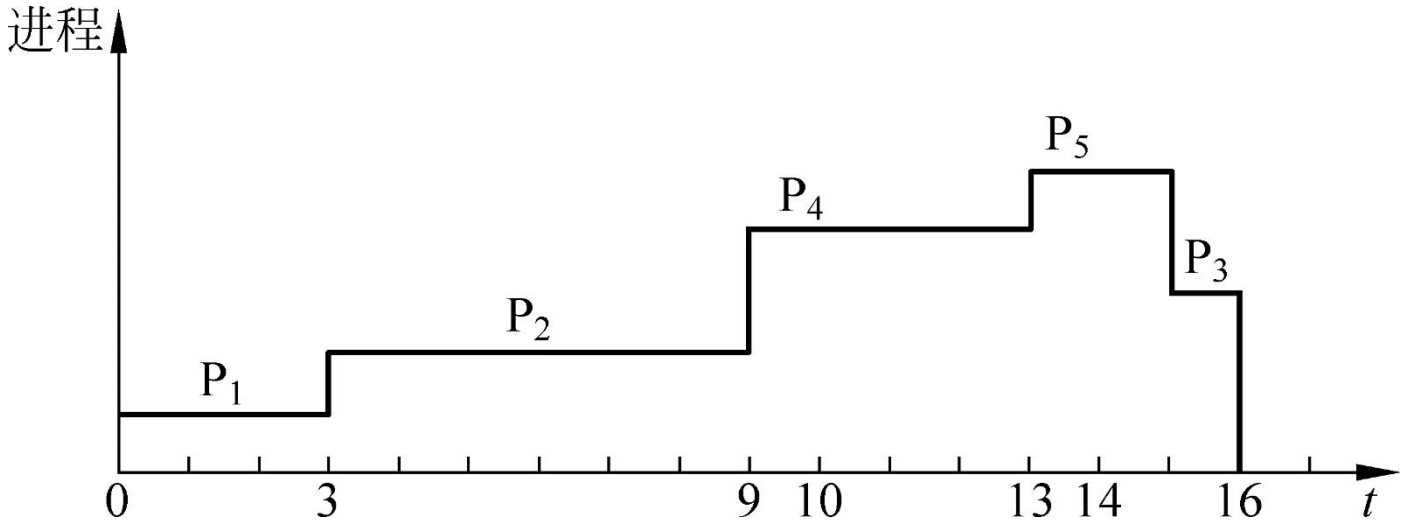


图4-11不可抢占的静态优先级调度算法时序图

4.5.5 优先级调度算法PS

- 【例5】仍然采用FCFS算法中的实例，分别改用不可抢占静态优先级调度算法和可抢占静态优先级调度算法重新调度。可抢占静态优先级调度算法具体分析如下：

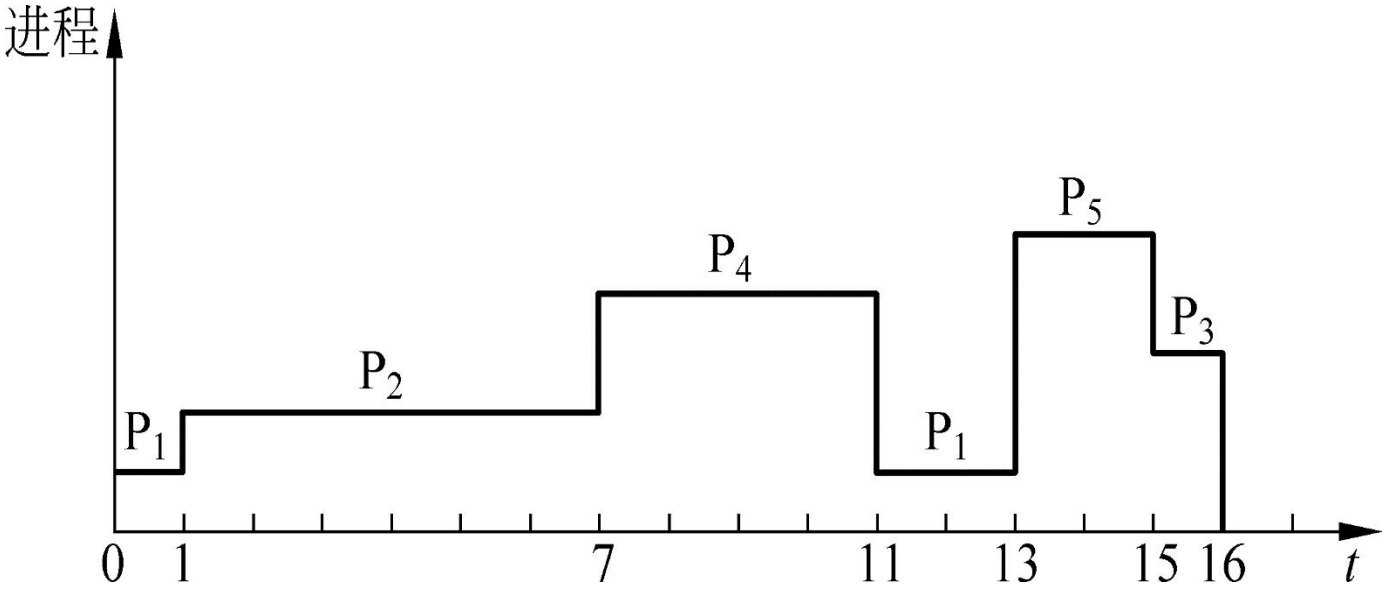


图4-12可抢占的静态优先级调度算法时序图

4.5.5 优先级调度算法PS

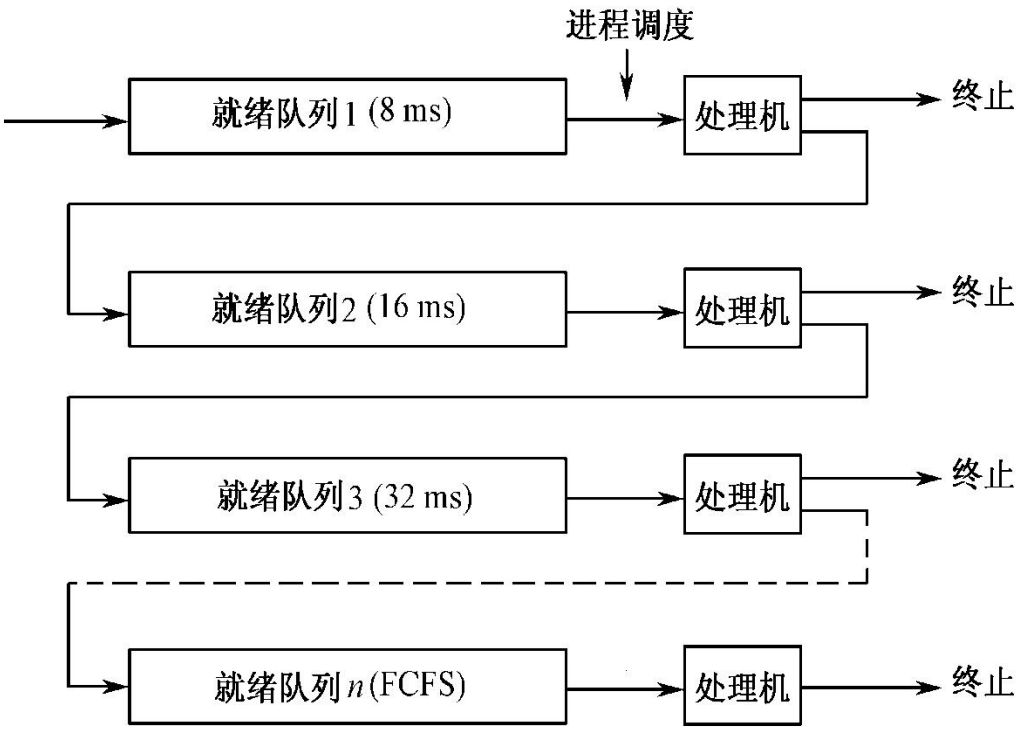
- 采用不可抢占和可抢占静态优先级调度算法，每个进程的调度结果如下表所示。

不可抢占式静态优先级调度算法						
进程	创建时间	运行时间	开始时间	完成时间	周转时间	带权周转时间
P_1	0	3	0	3	3	1
P_2	1	6	3	9	8	1.33
P_3	2	1	15	16	14	14
P_4	3	4	9	13	10	2.5
P_5	4	2	13	15	10	5
平均周转时间 T =9 带权平均周转时间 W =4.77					45	23.83
可抢占式静态优先级调度算法						
进程	创建时间	运行时间	开始时间	完成时间	周转时间	带权周转时间
P_1	0	3	0	13	13	4.33
P_2	1	6	1	7	6	1
P_3	2	1	15	16	14	14
P_4	3	4	7	11	8	2
P_5	4	2	13	15	11	5.5
平均周转时间 T =10.4 带权平均周转时间 W =5.37					52	26.83

4.5.6 多级反馈队列法MFQ

- MFQ：Multilevel Feedback Queue
- 实现思想
  - 系统中设置多个就绪队列，每个队列对应一个优先级；
  - 各就绪队列中进程的运行时间片不同，高优先级队列的时间片小，低优先级队列的时间片大；
  - 新进程进入系统后，先放入第1个队列的末尾，如果在时间片内工作未完成，则转入下一个队列尾，依此类推；
  - 系统先运行第1个队列中的进程，若第1队列为空，才运行第2队列，依次类推。

4.5.6 多级反馈队列法MFQ



- 抢占式调度，使用动态优先级机制。
- 是时间片轮转和优先级调度算法的综合和发展。
- 通过动态调整进程优先级和时间片大小，兼顾多方面系统指标。

4.5.6 多级反馈队列法MFQ

- 性能优秀，能较好满足各种类型用户的需要

① **保证了短进程优先，可让终端型用户满意。**终端型用户进程所需CPU时间不长，在高优先级队列中运行。

② **满足输入/输出型进程的要求。**让正在使用输入/输出设备的进程优先执行，可以充分提高设备的利用率，保证输入/输出型用户能及时得到响应。

4.5.6 多级反馈队列法MFQ

- 性能优秀，能较好满足各种类型用户的需要

③ **照顾了计算型长进程的执行。**长进程不断向优先级低的队列转换，而优先级低的队列时间片越长，保证了长进程能够较快执行完毕。

④ **系统开销小。**不需要动态计算时间片和优先级，进程的优先级和时间片等于它所在调度队列的优先级和时间片。

典型调度算法比较

名称	先来先服务（FCFS）	时间片轮转（RR）	短进程优先（SPF）	最短剩余时间（SRTF）	优先级（PSA）	多级反馈队（MFQ）
调度方式	非抢占式	抢占式	非抢占式	抢占式	非抢占式/抢占式	抢占式
吞吐量	不突出	如时间片太小，可能变低	高	高	低	不突出
响应时间	可能很高，特别在进程执行时间有很大变化时	对于短进程提供良好的响应时间	对于短进程提供良好的响应时间	提供良好的响应时间	对于紧迫性进程提供良好的响应时间	不突出
开销	最小	低	可能高	可能高	可能高	可能高
对进程的作用	有利于长进程和CPU繁忙型进程	公平对待	有利于短进程	有利于短进程	有利于紧迫性进程	有利于短进程和I/O繁忙型进程

4.6 线程调度

- 多线程系统中，提供了进程和线程两级并行机制。

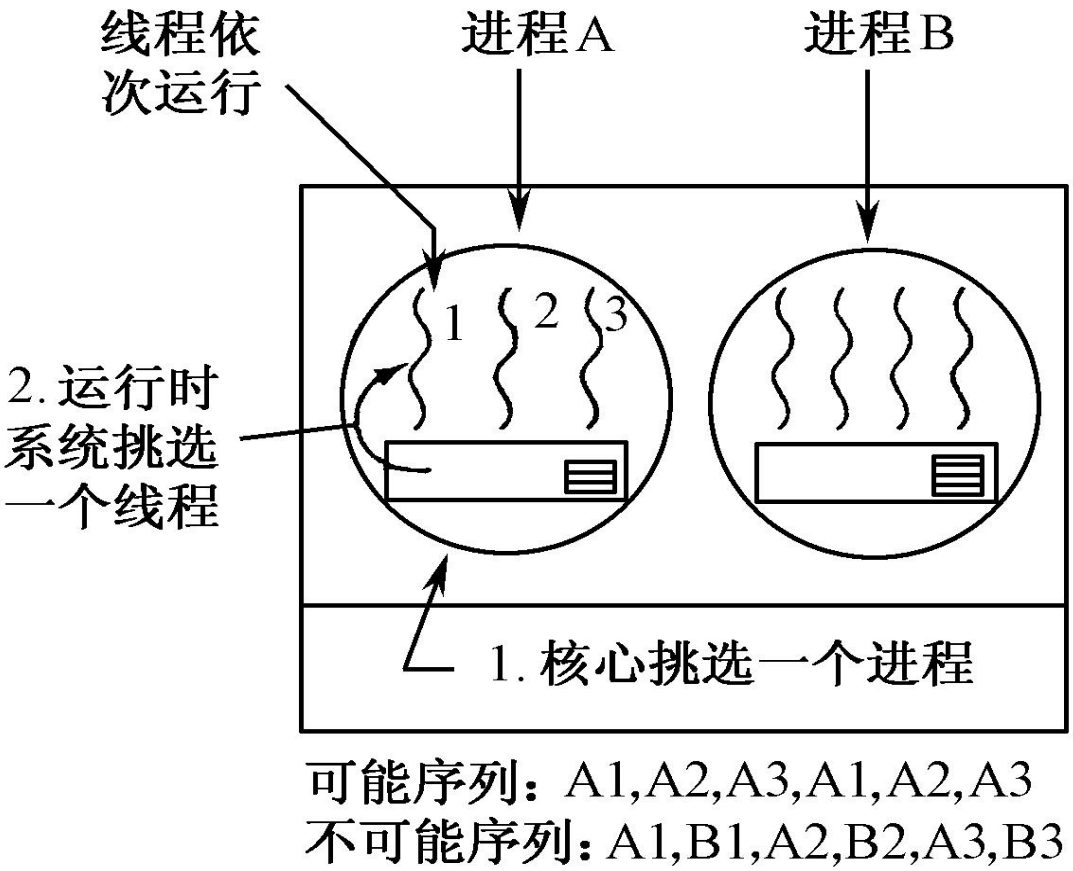
- 因为线程的实现分为用户级和核心级，所以在多线程系统中，调度算法主要依据线程的实现而不同。

4.6 线程调度

- **用户级线程调度**
  - 核心不负责线程的调度。核心只为进程提供服务，即从就绪队列中挑选一个进程（例如A），为它分配一个时间片，然后由进程A内部的线程调度程序决定让A的哪一个线程（如A1或A2）运行。
  - 核心切换到其它进程之前，可能发生A进程的多个线程的切换。

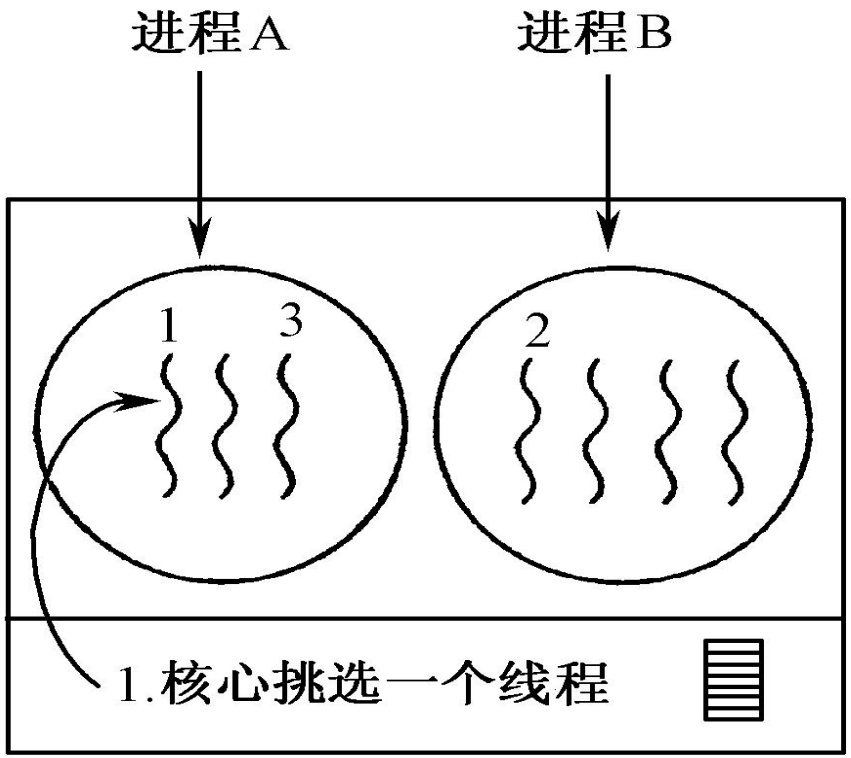
- **核心级线程调度**
  - 由核心调度线程，不同进程的线程之间可能发生切换。

4.6 线程调度



图示：用户级线程可能的调度

4.6 线程调度



可能序列：A1,A2,A3,A1,A2,A3
也可能序列：A1,B1,A2,B2,A3,B3

图示：核心级线程可能的调度

4.6 线程调度

- 用户级线程调度与核心级线程调度的特点
- 性能
  - 用户级线程切换可用机器指令，速度快；核心级线程切换需要全部上下文切换，速度慢。
- 阻塞
  - 核心级线程方式下，一个线程因等待I/O而阻塞时，不会挂起整个进程；而用户级方式下会挂起整个进程。

4.8 本章小结

- 中断是现代计算机系统中的重要概念之一，它是指CPU对系统发生的某个事件做出的处理过程。
- 三级调度：高级调度（作业调度）、中级调度（主存调度）、低级调度（进程调度）。
- 在设计进程调度时使用了面向用户和面向系统的准则。从用户的角度看，作业周转时间尽快短，响应时间要快，并且要保证截止时间，还要考虑优先权等。从系统的角度来看，系统的吞吐量和处理机的利用率是最重要，其次要考虑各类资源要均衡利用，最后才是调度算法的实现。

4.8 本章小结

- 典型的调度算法
  - 先来先服务算法
  - 短进程优先算法
  - 最短剩余时间优先算法
  - 时间片轮转调度算法
  - 优先级调度算法
  - 多级反馈队列调度算法
- 线程调度
  - 用户级线程调度
  - 核心级线程调度