操作系统
第六讲

张涛

Review

处理机调度的基本概念

调度算法

实时调度

多处理机调度

3.4.3 实时调度

- 实时系统（real-time system）
  - 能够实现在指定或者确定的时间内完成系统功能和对外部或内部、同步或异步时间做出响应的系统
  - 在实时计算中，系统的正确性不仅仅依赖于计算的逻辑结果，而且依赖于结果产生的时间。
- 实时任务（real-time task）
  - 周期性实时任务
  - 非周期性实时任务
  - 硬实时任务
  - 软实时任务

实现实时调度的基本条件

- 1. 提供必要的信息
  - 就绪时间
  - 开始截止时间和完成截止时间
  - 处理时间
  - 资源要求
  - 优先级
- 2. 可调度的实时系统(Schedulable real-time system)
  - 给定m个周期性事件(periodic events)
  - 事件i的周期为P_i，需要的处理时间为 C_i
  - 可调度的标准(schedulable)

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

- 不可调度的例子：
  - 系统中有6个硬实时任务，周期时间都是 50 ms，每次的处理时间为 10 ms。
- 解决的方法：提高系统的处理能力。途径有二：
  - 采用单处理机系统，但须增强其处理能力，以显著地减少对每一个任务的处理时间；
  - 采用多处理机系统。假定系统中的处理机数为N，则应将上述的限制条件改为：

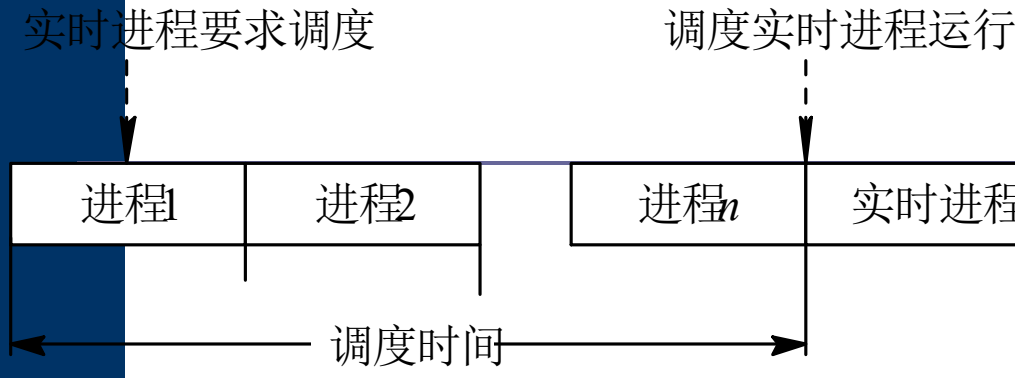
$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq N$$

实现实时调度的基本条件

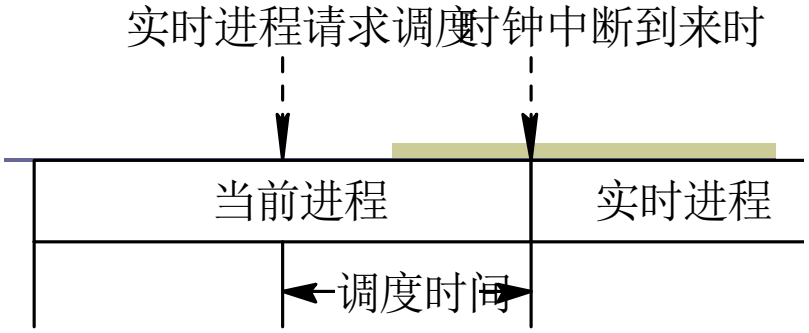
- 3. 采用抢占式调度机制
  - 当一个优先权更高的任务到达时，允许将当前任务暂时挂起，而令高优先权任务立即投入运行，可满足该实时任务对截止时间的要求。
- 4. 具有快速切换机制
  - 对外部中断的快速响应能力。
  - 快速的任务分派能力。

实时调度算法的分类

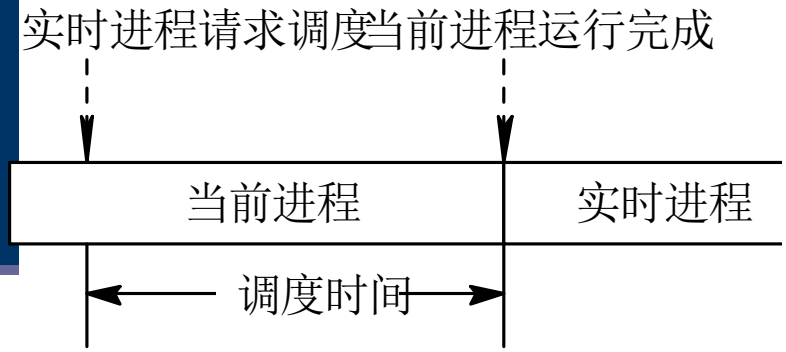
- 非抢占式调度算法
  - 非抢占式轮转调度算法
  - 非抢占式优先调度算法
- 抢占式调度算法
  - 基于时钟中断的抢占式优先权调度算法
  - 立即抢占(Immediate Preemption)的优先权调度算法



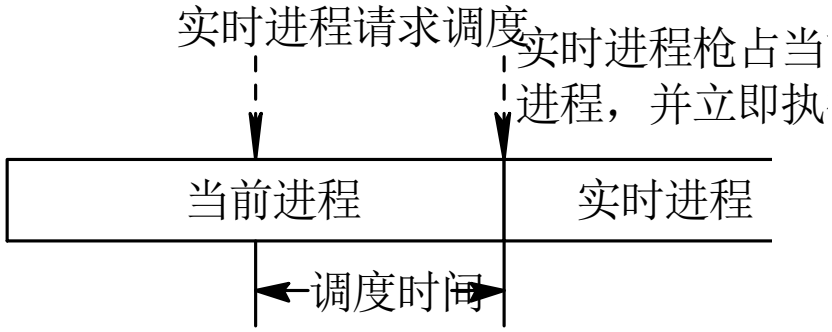
(*a*) 非抢占轮转调度



(*c*) 基于时钟中断抢占的优先权抢占调度

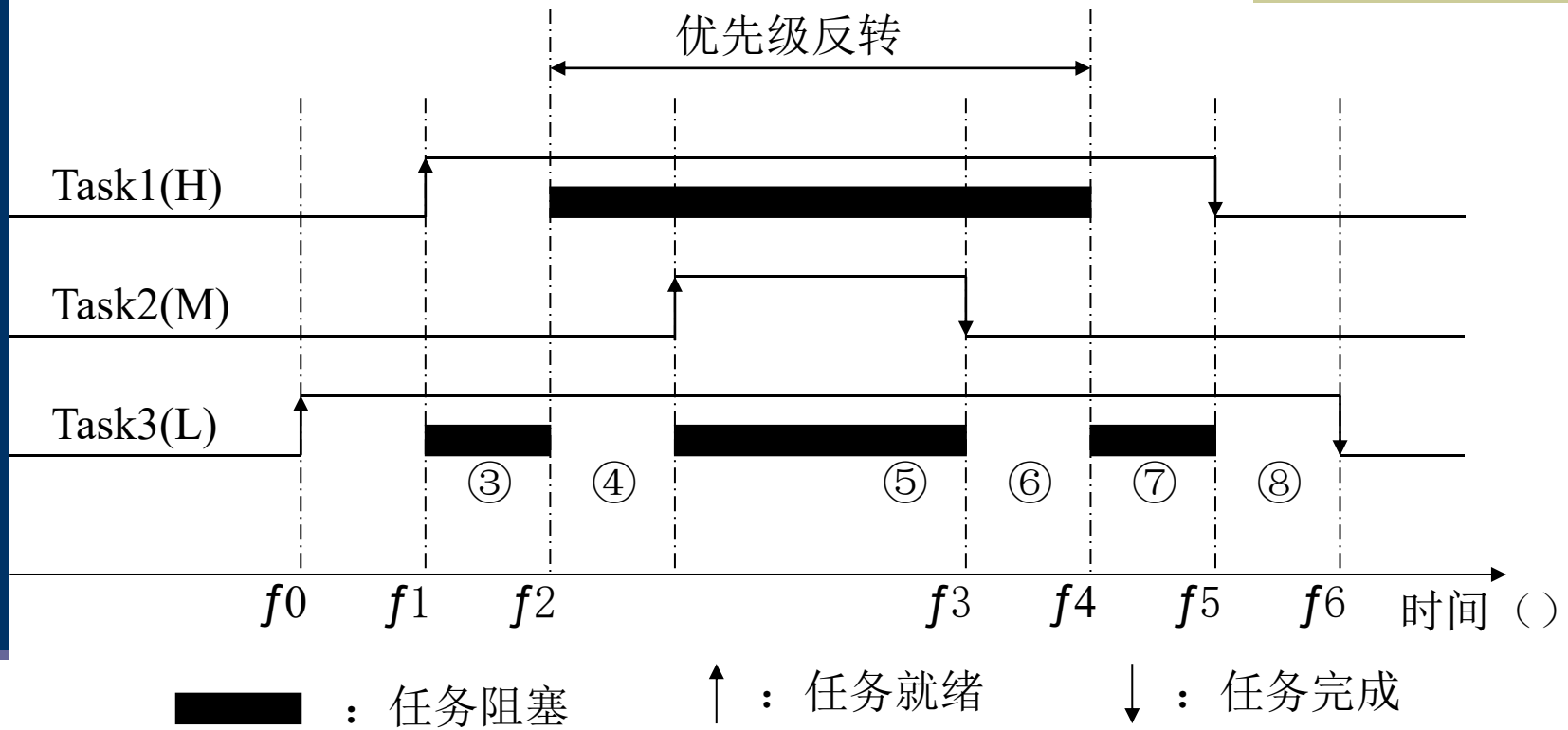


(*b*) 非抢占优先权调度



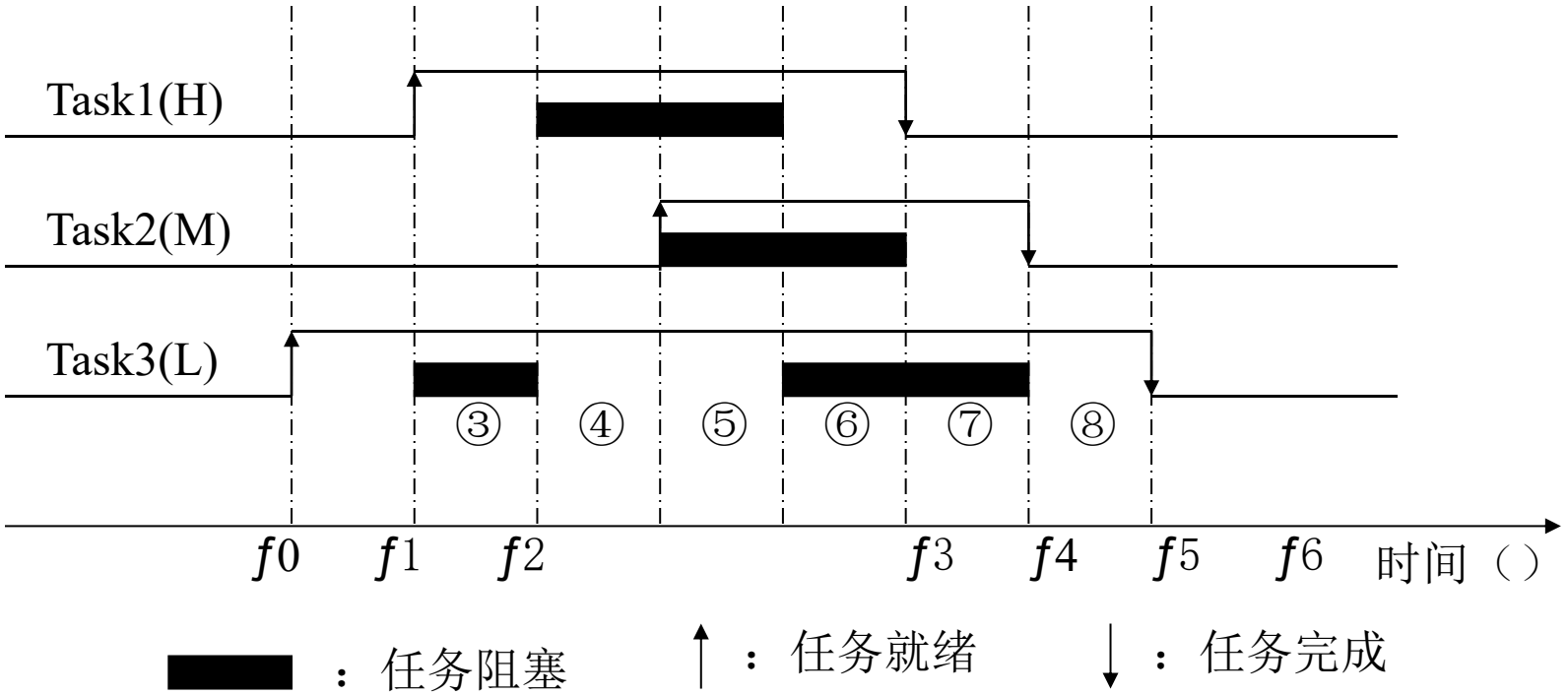
(*d*) 立即抢占的优先权调度

- **优先级反转：** 高优先级的进程必须等待低优先级的进程完成



优先级反转示意图

- **优先级继承：** 所有使用到高优先级进程所需资源的进程，继承高优先级直到用完竞争资源，再回到原来的优先级。



采用优先级继承消除优先级反转

常用实时调度算法

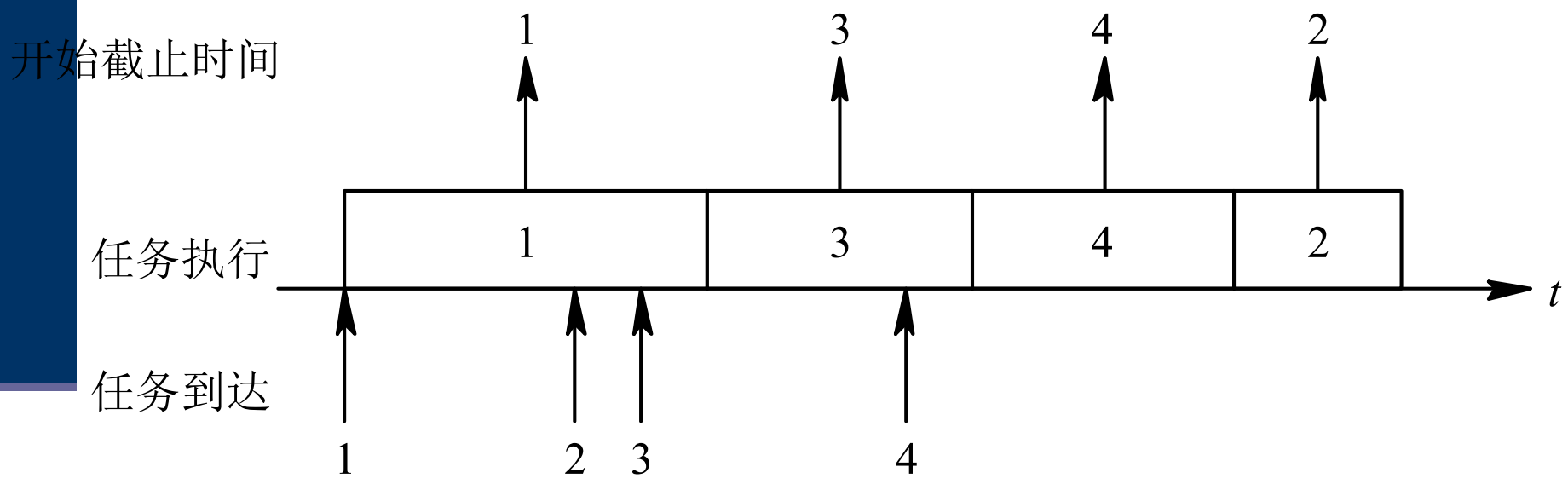
- 静态表调度算法（Static table-driven scheduling）
  - 适用于周期性的实时应用。通过对所有周期性任务的分析预测（到达时间、运行时间、结束时间、任务间的优先关系），事先确定一个固定的调度方案。这种方法的特点是有效但不灵活。
- 静态优先级调度算法（Static priority-driven scheduling）
  - 把通用的优先级调度算法用于实时系统，但优先级的确定是通过静态分析（运行时间、到达频率）完成的。

实时调度算法

- 动态分析调度算法（Dynamic planning-based scheduling）
  - 在任务下达后执行前进行调度分析，要求满足实时性要求。
- 无保障动态调度算法（Dynamic best effort scheduling）
  - 在任务下达时分配优先级，开始执行，在时限到达时未完成的任务被取消。用于非周期性任务的实时系统。

常用的几种实时调度算法(1)

1. 最早截止期优先EDF（Earliest Deadline First）算法

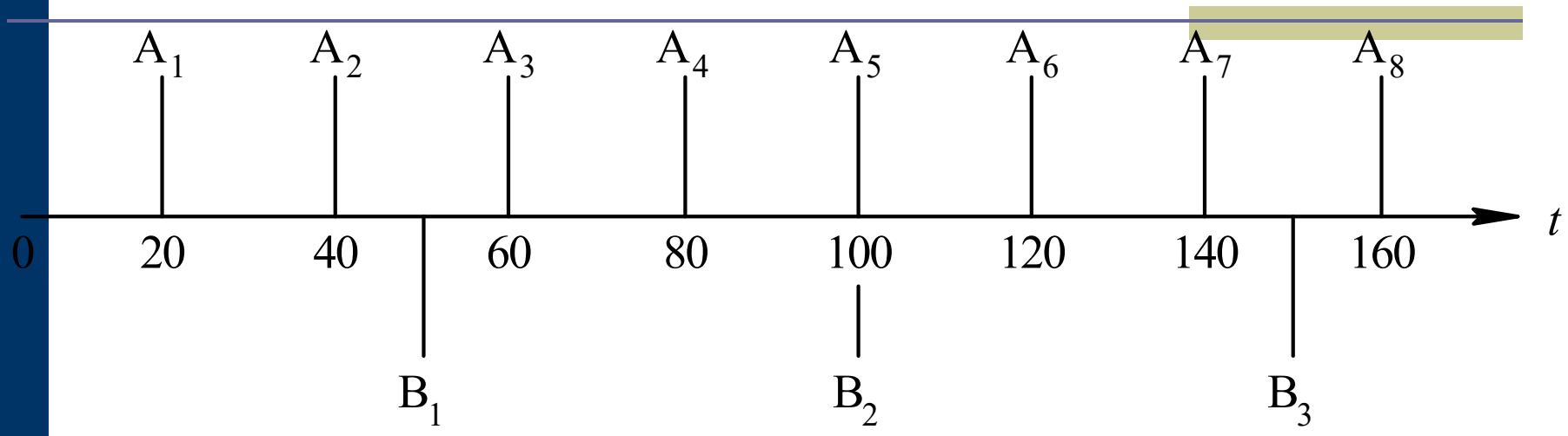


EDF算法用于非抢占调度方式

常用的几种实时调度算法(2)

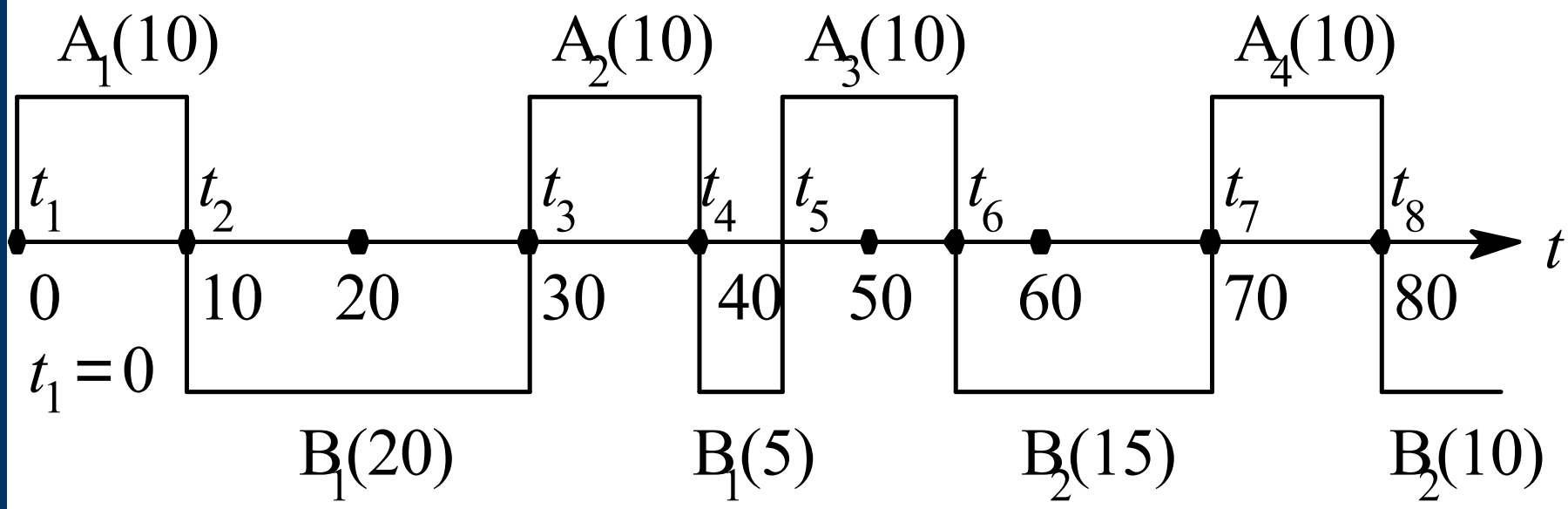
2. 最低松弛度优先即LLF（Least Laxity First）算法

- 根据任务紧急（或松弛）的程度，来确定任务的优先级；
- 实现该算法时要求系统中有一个按松弛度排序的实时任务就绪队列，松弛度最低的任务排在队列最前面，调度程序总是选择就绪队列中的队首任务执行；
- 主要用于可抢占调度方式中。



A和B任务每次必须完成的时间

假如在一个实时系统中，有两个周期性实时任务A和B，任务A要求每 20 ms执行一次，执行时间为 10 ms；任务B只要求每50 ms执行一次，执行时间为 25 ms。



利用LLF算法进行调度的情况

3.4.4 多处理机调度

- 与单处理机调度的区别
- 对称式多处理系统(SMP)
- 非对称式多处理系统(ASMP)的处理机调度
- 成组调度(gang scheduling)
- 专用处理机调度(dedicated processor assignment)

与单处理机调度的区别

- 注重整体运行效率（而不是个别处理机的利用率）
- 更多样的调度算法
- 多处理机访问OS数据结构时的互斥（对于共享内存系统）
- 调度单位广泛采用线程

对称式多处理系统(SMP)

- 按控制方式，SMP调度算法可分为：
  - 集中控制
  - 分散控制
- 集中控制
  - 静态分配(static assignment)：每个CPU设立一个就绪队列，进程从开始执行到完成，都在同一个CPU上。
    - 优点：调度算法开销小。
    - 缺点：容易出现忙闲不均。
  - 动态分配(dynamic assignment)：各个CPU采用一个公共就绪队列，队首进程每次分派到当前空闲的CPU上执行。

分散控制

- 自调度(self-scheduling)：各个CPU采用一个公共就绪队列，每个处理机都可以从队列中选择适当进程来执行。需要对就绪队列的数据结构进行互斥访问控制。
  - 最简单的一种调度方式，直接由单处理机环境下的调度方式演变而来
  - 系统中的公共就绪队列可按照单处理机系统中所采用的各种方式加以组织；其调度算法也可沿用单处理机系统所用的算法
  - 只要系统中有任务，只要公共就绪队列不空，就不会出现处理机空闲的情况，也不会发生处理器忙闲不均的现象，有利于提高处理器的利用率。
- 优点：简单，处理机利用率高

非对称式多处理系统(ASMP)

- 主－从处理机系统，由主处理机管理一个公共就绪队列，并分派进程给从处理机执行。
- 从机空闲时，便向主机发送一索求进程的信号，然后，便等待主机为它分配进程。
- 各个处理机有固定分工，如执行OS的系统功能，I/O处理，应用程序。

成组调度(gang scheduling)

- 将一个进程中的一组线程，每次分派时同时到一组处理机上执行，在剥夺处理机时也同时对这一组线程进行。
- 优点
  - 通常这样的一组线程在应用逻辑上相互合作，成组调度提高了这些线程的执行并行度，有利于减少阻塞和加快推进速度，最终提高系统吞吐量。
  - 每次调度可以完成多个线程的分派，在系统内线程总数相同时能够减少调度次数，从而减少调度算法的开销

专用处理机调度 (dedicated processor assignment)

- 为进程中的每个线程都固定分配一个CPU，直到该线程执行完成。
  - 缺点：线程阻塞时，造成CPU的闲置。
  - 优点：线程执行时不需切换，相应的开销可以大大减小，推进速度更快。
- 适用场合：CPU数量众多的高度并行系统，单个CPU利用率已不太重要。

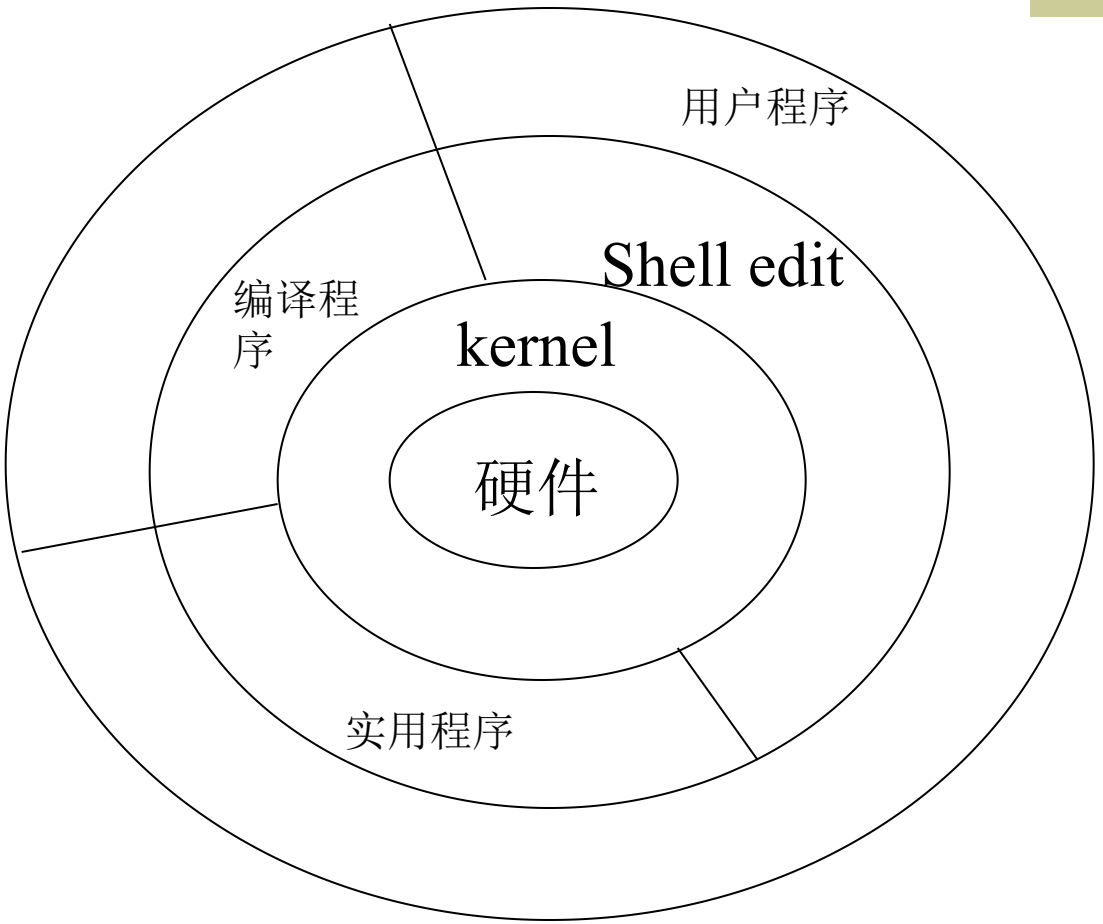
Example：UNIX进程

UNIX概述

UNIX的特征：

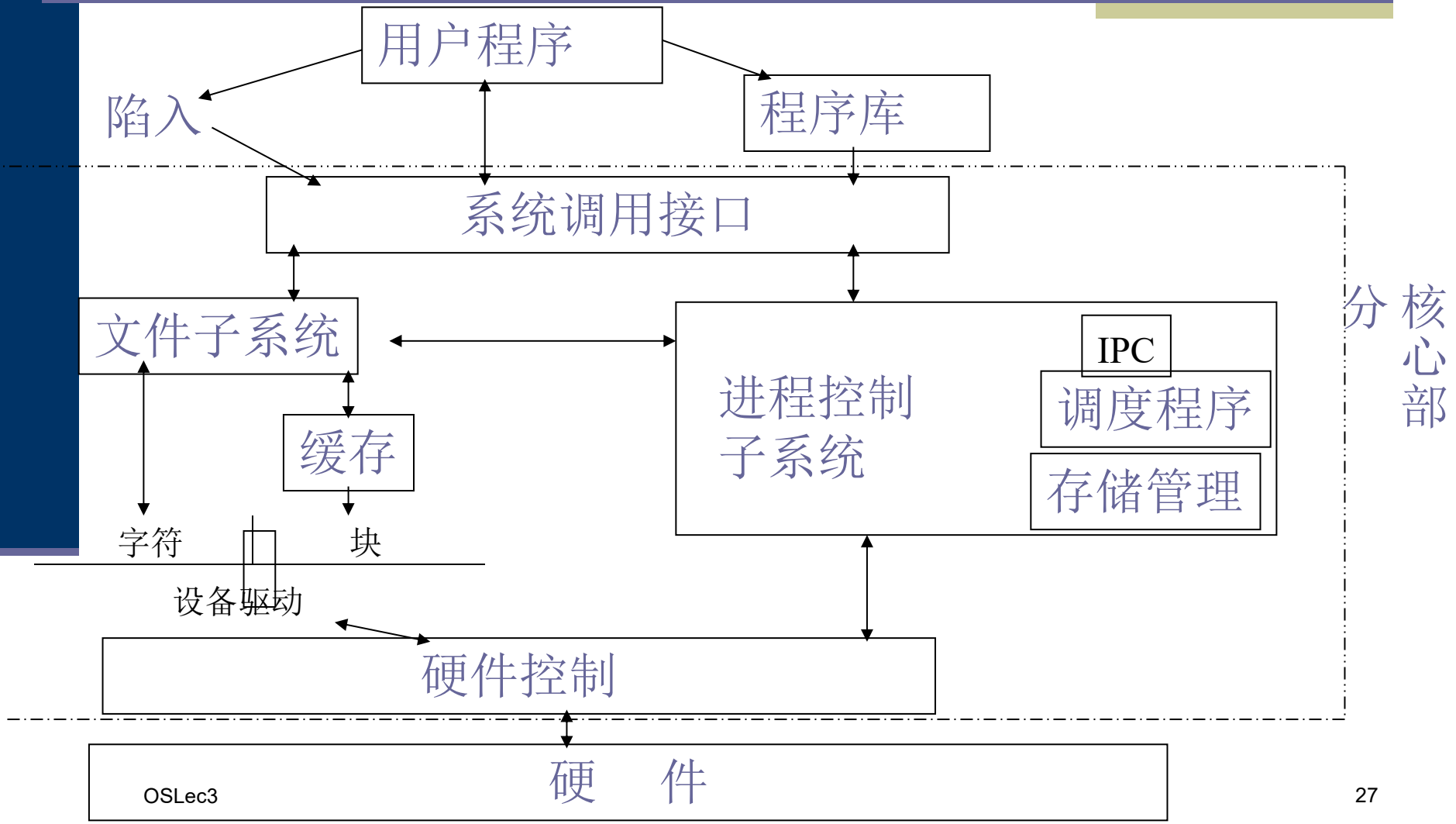
- 核心部分设计简单且功能全面
- 支持多用户多任务
- 文件系统可装卸
- 有良好的开放性及可移植性
- 具有强大的命令解释功能
- 具有完善的安全机制
- 具有网络特性

UNIX系统结构



UNIX简略结构

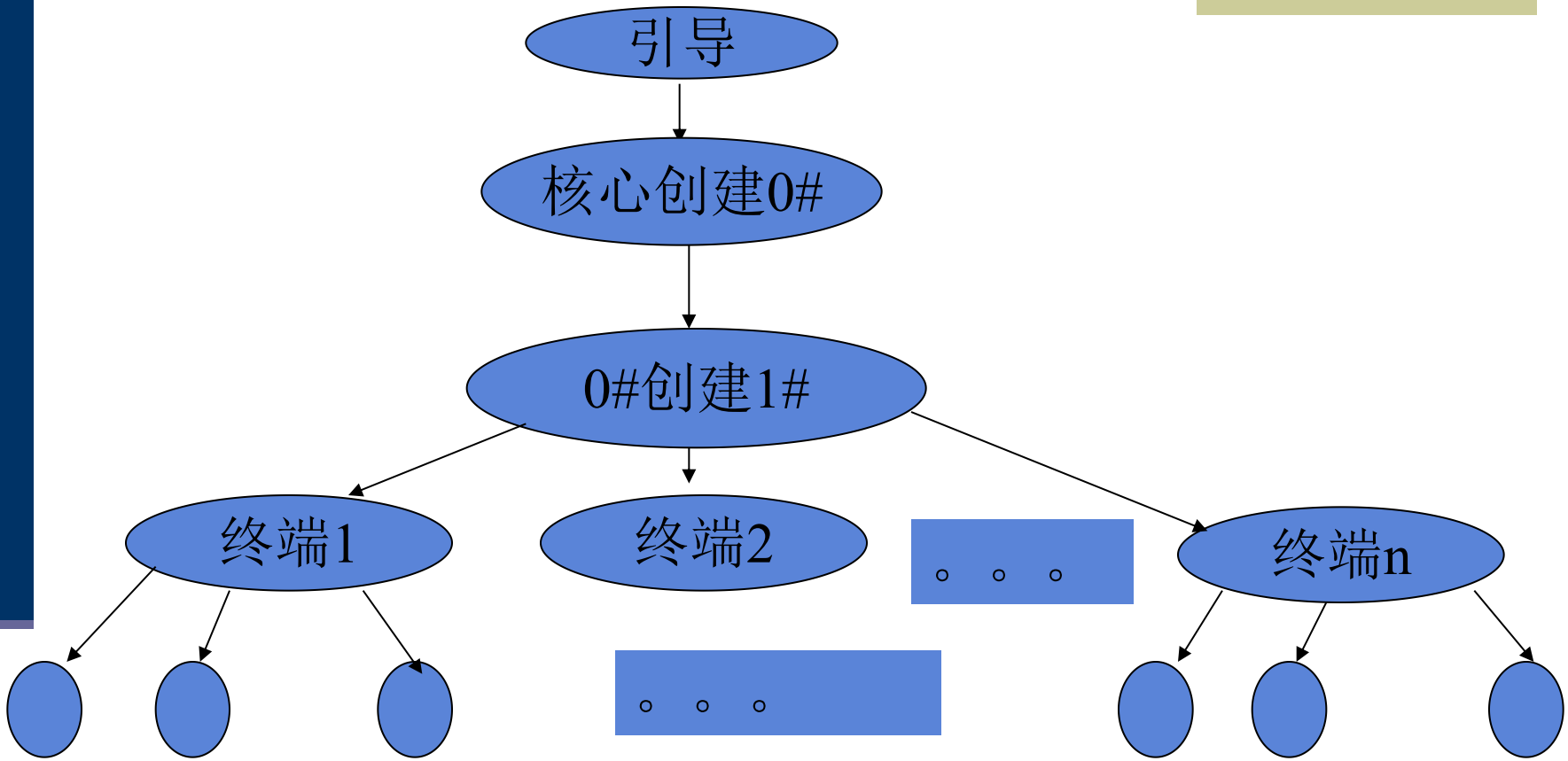
UNIX核心框图


用户程序
陷入
程序库
系统调用接口
文件子系统
缓存
字符
块
设备驱动
进程控制
子系统
IPC
调度程序
存储管理
硬件控制
硬　件
分 核心部

UNIX进程控制子系统

- UNIX中进程创建机制
- UNIX中的进程描述
- UNIX进程状态及其转换
- 进程映象

UNIX中进程创建机制



进程创建层次结构

UNIX中的两个特殊进程

- 0#进程
  - 由内核程序创建，初始化时创建1#进程
  - 负责进程调度与对换
- 1#进程
  - 是所有用户进程的祖先进程

UNIX中的进程描述

- UNIX把PCB分为四部分：
  - proc结构（进程基本控制块）
  - User结构（进程扩充控制块）
  - 系统区表：各区的页表位置
  - 本进程区表pprt(Per Process Region Table)，指向系统区表

proc结构（进程基本控制块）

- 进程状态
- 进程用户标识
- 进程标识
- 进程存储大小
- 进程调度参数
- 软中断信号项
- 执行时间及资源使用
- User结构起始址
- 进程区表指针
- 被挂起事件描述

user结构（进程扩充控制块）

- 指向proc的指针
- 系统调用参数
- 与用户标识有关项
- 与文件结构有关项
- 输入输出项
- 打开文件描述符
- 中断及软中断参数
- 错误信息
- 文件权限屏蔽项
- 交换数据项

UNIX进程状态及其转换

九种状态：

1）初始态

2）内存就绪态

3）外存就绪态

4）核心态下的执行态

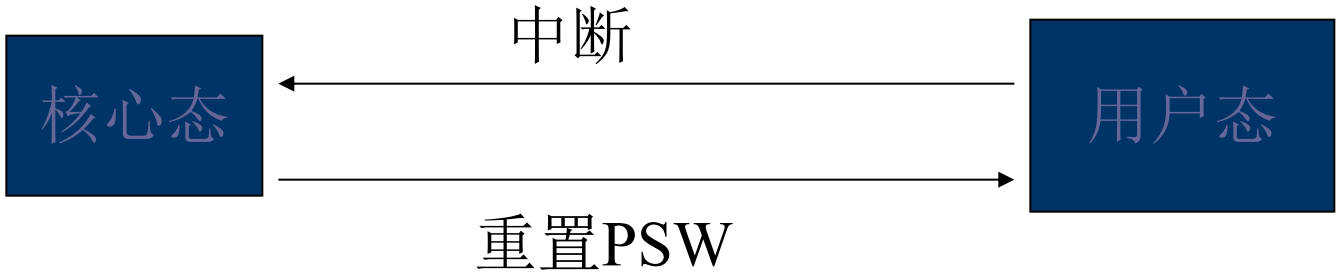
5）用户态下的执行态

6）内存中的睡眠

7）外存睡眠

8）被剥夺

9）僵死态



进程映象

UNIX进程映象由三部分组成：

- 用户级上下文：包括用户正文段、用户数据段和用户栈
- 寄存器级上下文：程序寄存器（PC）、处理机状态寄存器（PSW）、栈指针、通用寄存器
- 系统级上下文：
  - 静态部分（PCB和资源表格）
  - 动态部分：核心栈（核心过程的栈结构，不同进程在调用相同核心过程时有不同核心栈）

UNIX进程调度

进程调度程序主要责任：

- 对参与竞争CPU且已具备执行条件的进程进行分析和裁决
- 对选中的进程做CPU控制权转交
- 管理进程运行中各种状态的转换
- 完成进程在系统内外存之间的对换

UNIX中有两种调度时机：

- 进程自动放弃CPU
- 进程由核心态转为用户态时

调度算法

动态优先级多级反馈循环调度法

Round Robin With Multiple Feedback

- Review：多队列反馈调度算法
  - 将就绪队列分为N级，每个就绪队列分配给不同的时间片，队列级别越高，时间片越长，级别越小，时间片越短；
  - 系统从第一级调度，当第一级为空时，系统转向第二个队列，.....当运行进程用完一个时间片，放弃CPU时，进入下一级队列；
  - 等待进程被唤醒时，进入原来的就绪队列；
  - 当进程第一次就绪时，进入第一级队列

算法简要描述

- 给进程分配时间片
- 时间片结束时计算进程优先级
- 用优先数做比较选出高优先级进程
- 调度高优先级进程开始运行
- 被抢夺了cpu的进程反馈到相应的优先级队列中

计算公式

$P_{pri} = P_{cpu}/2 + PUSER + P_{nice} + NZERO$

其中：

- PUSER，NZERO—是优先数基值；
- P_{cpu}—是每个进程最近一次使用 CPU的时间；
- P_{nice}—是用户设置的进程优先数偏移值。

普通用户只可增大Pnice

UNIX进程调度的过程

- 分三步完成：
  - 第一步：检查是否做且系统允许做上下文切换,若条件满足，则保存：
    - 进程中各段内容（数据，正文，栈）
    - 有关寄存器中内容
    - 相关的栈指针

    否则返回有关信息，不作调度。
  - 第二步：0号进程选取就绪队列中一个优先级最高的进程，使之占有CPU；若没有满足条件的进程存在，0号进程循环，等待直到条件满足为止。
  - 第三步：被选中的进程变为当前进程。系统从它的核心栈或user结构中恢复该进程的有关寄存器内容和栈指针，新进程开始执行。

What you need to do?

- 1. 有五个待运行的作业，估计它们的运行时间分别是：9，6，3，5和X。采用哪种次序运行这些作业将得到最短的平均响应时间？（答案将依赖于X）
- 2. 有五个批处理作业A~E，他们几乎同时到达一个计算中心。估计它们的运行时间分别为10，6，2，4和8分钟，其优先级由外部设定，分别为3，5，2，1和4，其中5为最高优先级。对于下列每种调度算法，计算其平均进程周转时间。忽略进程切换开销。
  - 轮转法（时间片粒度为2分钟）
  - 优先级调度
  - 先来先服务（按照10，6，2，4，8次序）
  - 最短作业优先

- 复习课本3.4节的内容
- 课后作业：P91，第4、5题

See you next time!