操作系统
Operating System

汤臣薇、冯文韬

tangchenwei@scu.edu.cn, Wtfeng2021@scu.edu.cn

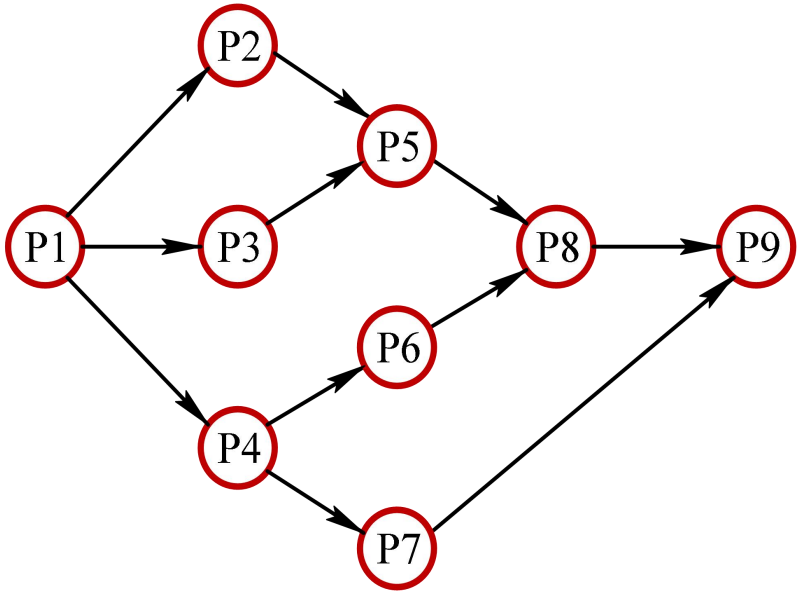
四川大学计算机学院（软件学院）

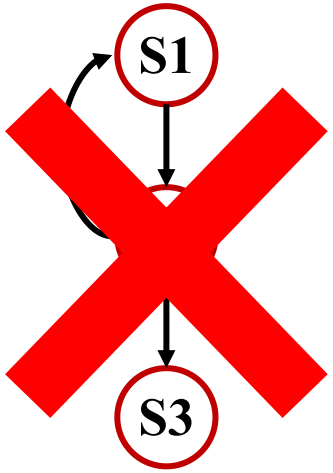
数据智能与计算艺术实验室

回顾——前趋图

◆ **前趋图（Precedence Graph）：描述程序执行先后顺序**

- 描述多个进程之间的关系
- 有向无循环图（DAG）
- 结点表示一个进程或一段程序
- 结点之间用一个有方向的线段相连
- 方向表示所连接的结点之间的前趋和后继关系
- 被指向的结点为后继结点，离开箭头的结点是前趋结点

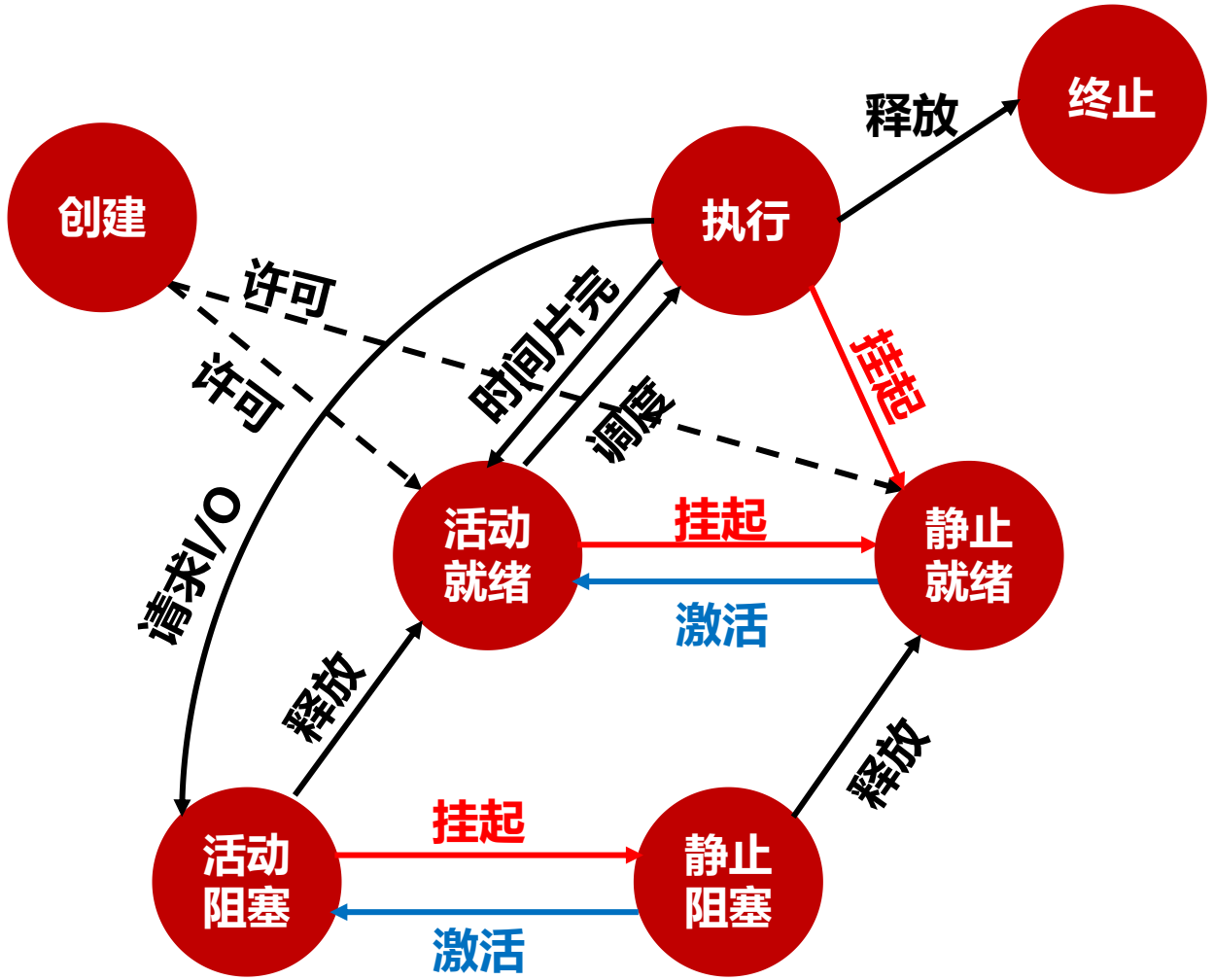




回顾——进程的特征

结构性	进程包含有描述进程信息的数据结构（包含进程控制块、程序块和代码块等）和运行在进程上的程序，OS用PCB描述和记录进程的动态变化过程。
动态性	最基本特征，是程序执行过程，有一定的生命期：由创建而产生、由调度而执行，因得不到资源而暂停，由撤消而死亡。而程序是静态的，它是存放在介质上一组有序指令的集合，无运动的含义。
并发性	进程的重要特征，也是OS的重要特征。指多个进程实体同存于内存中，能在一段时间内同时运行。而程序（没有建立PCB）是不能并发执行。
独立性	进程是一个能独立运行的基本单位，即是一个独立获得资源和独立调度的单位，而未建立PCB的程序不能作为独立单位参加运行、获取资源。
异步性	进程按各自独立的不可预知的速度向前推进（即按异步方式进行），正是这一特征导致程序执行的不可再现性，因此OS必须采用某种措施来限制各进程推进序列以保证各程序间正常协调运行。

回顾——进程的基本状态与转换



回顾——进程同步机制

- 进程
- 多道程序并发执行
- 资源利用率
- 系统吞吐量
- 系统更加复杂

进程同步的基本概念

多道程序系统
进程同步机制

- 进程运行管理
- 对系统资源的无序争夺
- 系统混乱
- 结果不确定性
- 结果不可再现性

硬件同步机制

信号量机制

管程机制

任务：对多个相关进程在执行次序上进行协调，使并发执行的诸进程之间按照一定的规则(或时序)共享系统资源，并能很好地相互合作，从而使程序的执行具有可再现性

回顾——进程同步机制

◆ 同步机制应遵循的规则

- **空闲让进：**
  - 当无进程在互斥区时，任何有权使用互斥区的进程可进入
- **忙则等待：**
  - 不允许两个以上的进程同时进入互斥区
- **有限等待：**
  - 任何进入互斥区的要求应在有限的时间内得到满足
- **让权等待：**
  - 处于等待状态的进程应放弃占用CPU，以使其他进程有机会得到CPU的使用权

◆ 硬件同步机制

- **关中断**
- **利用test-and-set指令实现互斥**
- **利用swap指令实现进程互斥**

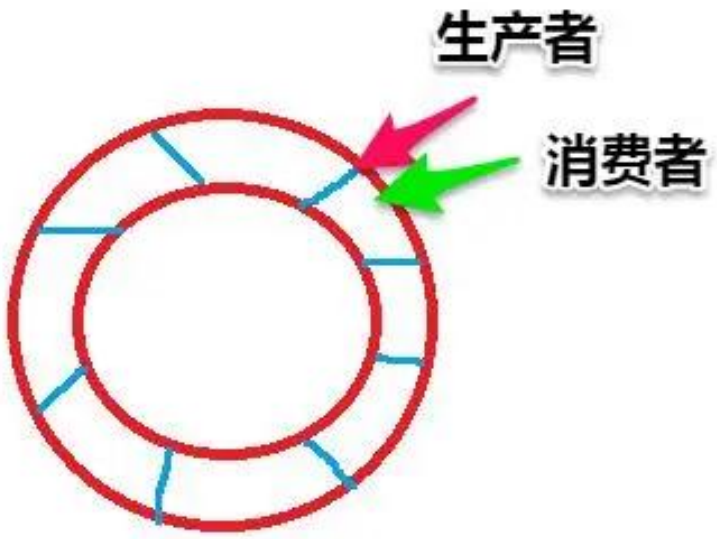
◆ 信号量机制（PV操作）

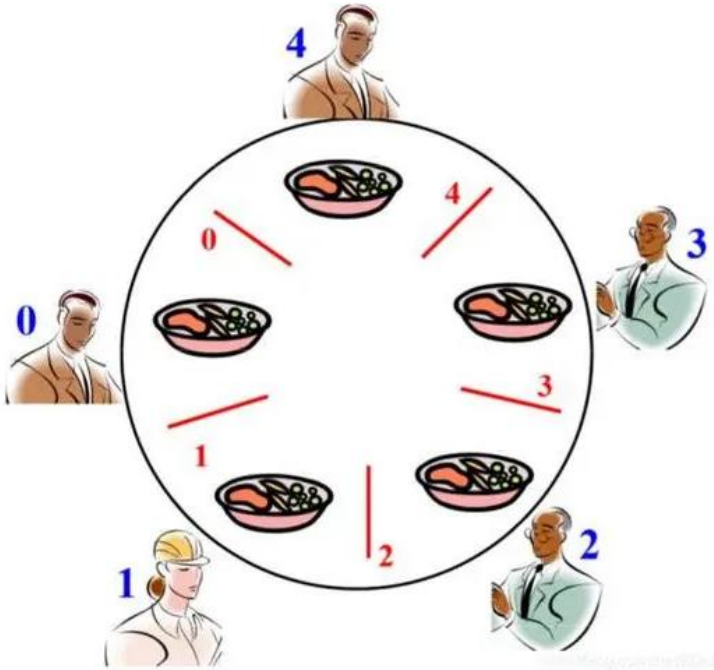
- **整型信号量**
- **记录型信号量（结构型信号量）**
- **AND型信号量**
- **信号量集**

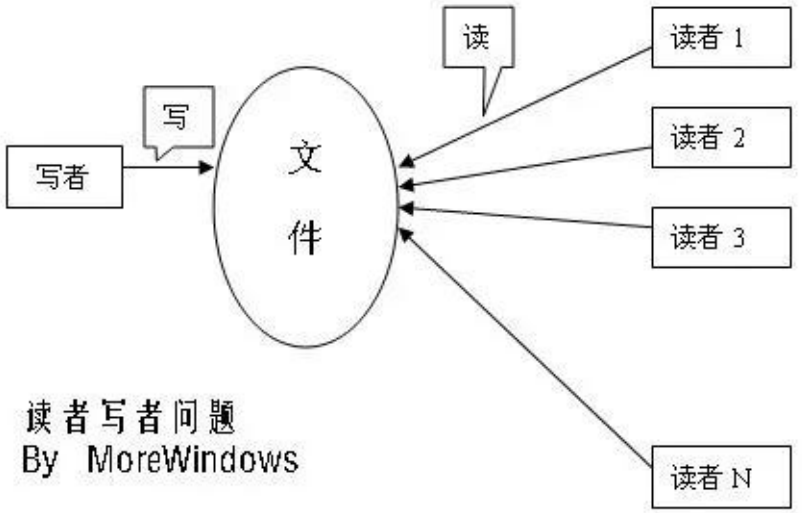
回顾——经典进程的同步问题

经典进程的同步问题

- 生产者和消费者问题
- 哲学家就餐问题
- 读者和写者问题







回顾——进程通信的类型

- 进程通信的类型
  - 共享存储器系统（shared-memory system）
  - 管道（piple）通信系统
  - 消息传递系统（message passing system）
    - 直接通信方式
    - 间接通信方式（邮箱）
  - 客户机-服务器系统（client-server system ）

进程的描述和控制

- 前趋图和程序执行
- 进程的描述
- 进程控制
- 进程同步 ★
- 经典进程的同步问题 ★
- 进程通信
- **线程的基本概念**
- 线程Thread的实现

2.7.1 线程的引入

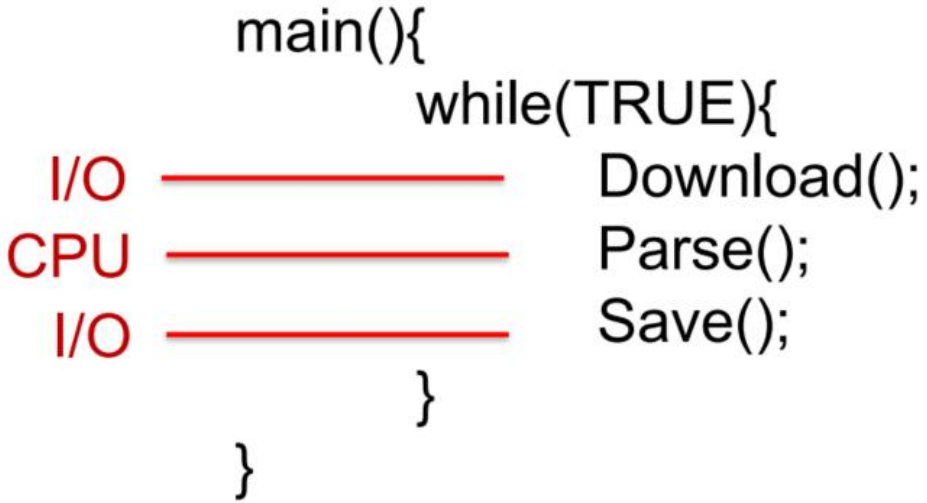
- **案例**
  - **编写一个网络爬虫。核心功能模块有三个：**
    - **下载Html文件**
    - **解析Html文件**
    - **存储结果信息**

2.7.1 线程的引入

◆ 案例

➢ 编写一个网络爬虫。核心功能模块有三个：

- 下载Html文件
- 解析Html文件
- 存储结果信息



问题：

各函数之间不是并发执行，影响资源的使用效率

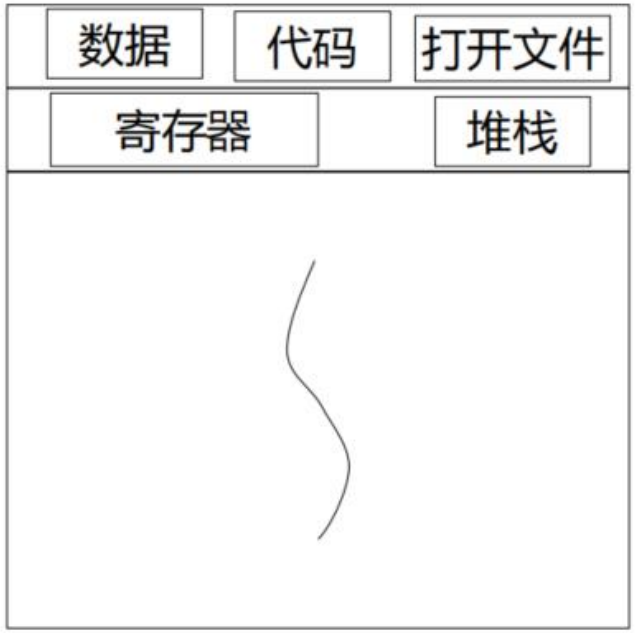
2.7.1 线程的引入

- **引入进程是为了解决什么问题?**
  - 解决单处理机环境下的程序并发执行问题
    - ✓ 进程是一个可**拥有资源**的独立单位
    - ✓ 进程是一个可**独立调度**和**分派**的基本单位
- **引入进程是否是最完美的解决方案?**
  - 进程太大?
  - 进程太小?
    - ✓ 思路：设法将进程的两个属性分开

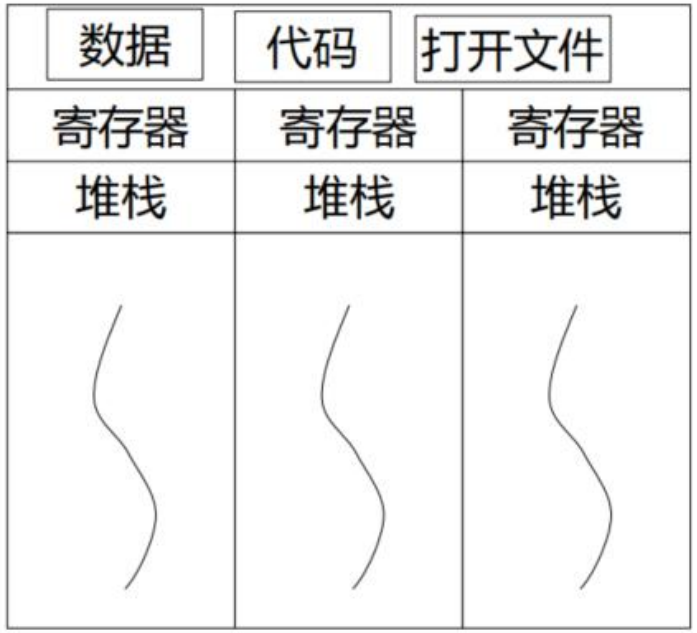
2.7.1 线程的引入

◆ 线程的基本概念

- ➢ 线程是进程的一部分，描述指令流执行状态
- ➢ 进程中指令执行的最小单元，是CPU调度的基本单位



单线程进程



多线程进程

2.7.1 线程的引入

◆ 线程的优点

- 一个进程中可以同时存在多个线程
- 各个线程间可以并发执行
- 各个线程之间可以共享地址空间和文件等资源
- 提高程序并发执行的程度，进一步改善系统的服务质量

◆ 线程的缺点

- 一个线程崩溃，会导致所属进程的所有其他线程崩溃

2.7.1 线程的引入

◆ 进程的两个基本属性

- **资源拥有的独立单位**：进程还是资源的拥有者
- **CPU调度单位**：线程继承了这个属性

> 引入进程的目的是使多个程序能并发执行，以提高资源利用率和系统吞吐量。引入线程则是为了减少程序在并发执行时所付出的时空开销，以使OS具有更好的并发性~

◆ 线程与进程的比较

- **调度的基本单位**：线程作为调度和分派的基本单位
- **并发性**：引入线程的OS并发度更高
- **拥有资源**：线程本身不拥有系统资源，仅有一点必不可少的资源
- **独立性**：线程之间独立性低，共享进程的内存地址空间和资源
- **系统开销**：线程的开销低
- **支持多处理机系统**：线程更适合多处理机

2.7.1 线程的引入

◆ 线程和进程的关系

- 一个线程只能属于一个进程，而一个进程可以有多个线程，至少有一个线程。
- 资源分配给进程，同一进程的所有线程共享该进程的所有资源。
- 处理机分给线程，即真正在处理机上运行的是线程。
- 线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。线程是指进程内的一个执行单元，也是进程内的可调度实体

2.7.1 线程的引入

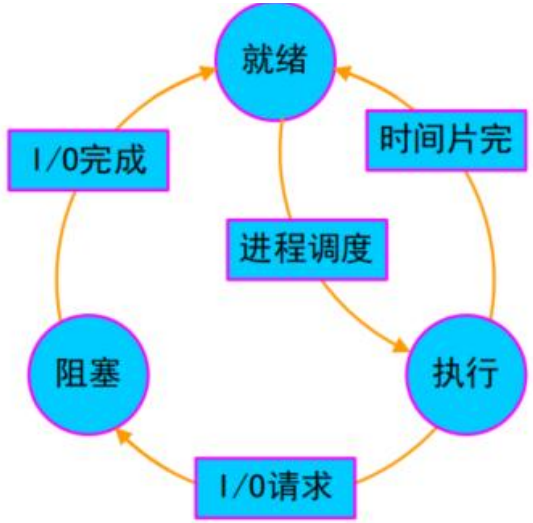
◆ 线程和进程的区别

- **调度：** 线程作为调度和分配的基本单位，进程作为拥有资源的基本单位
- **并发性：** 不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行
- **拥有资源：** 进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源
- **系统开销：** 在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销

2.7.1 线程的引入

◆ 线程的三个运行状态

- 执行状态
- 就绪状态
- 阻塞状态



◆ 线程控制块TCB

- 线程标识符、一组寄存器、线程运行状态、优先级、线程专有存储区、信号屏蔽、堆栈指针

◆ 多线程OS中的进程属性：

- 进程是一个可拥有资源的基本单位
- 多个线程可并发执行
- 进程已不是可执行的实体

进程的描述和控制

- 前趋图和程序执行
- 进程的描述
- 进程控制
- 进程同步 
- 经典进程的同步问题 
- 进程通信
- 线程的基本概念
- 线程的实现

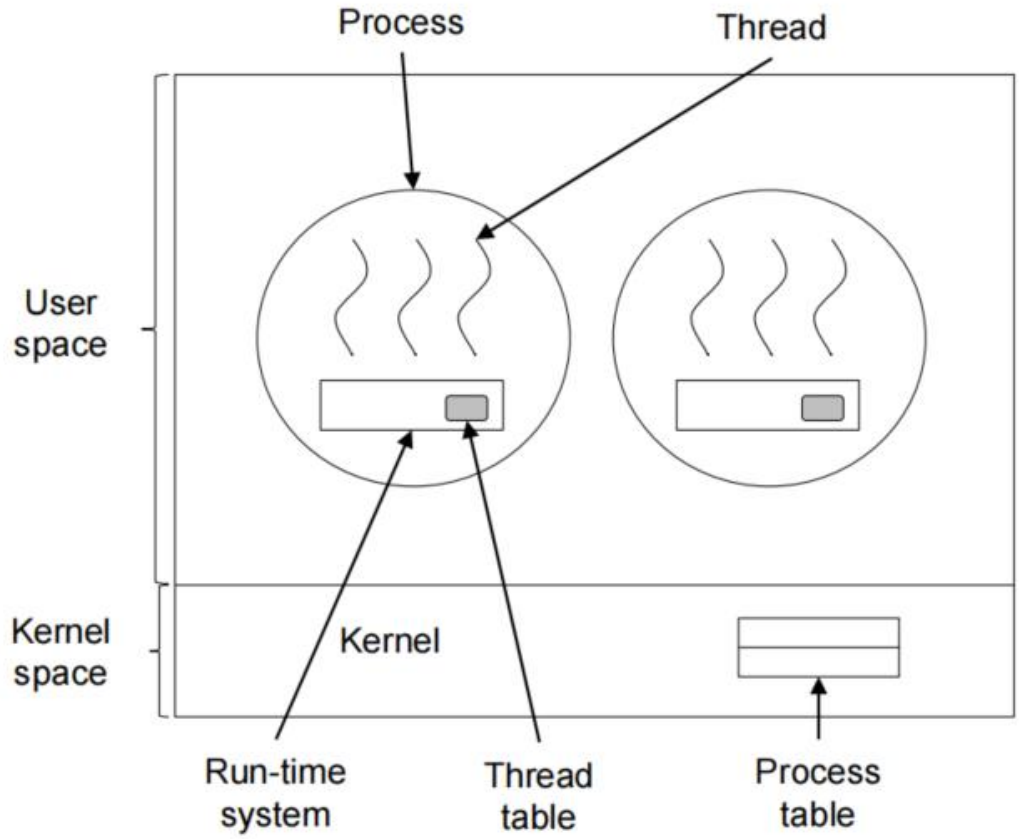
2.8.1 线程的实现方式

- 用户级线程ULT (User level Threads)
- 内核支持线程KST (Kernel Supported Threads)
- 组合方式 (Combined)

2.8.1 线程的实现方式

◆ 用户级线程ULT (User level Threads)

- **在用户空间建立线程库：**提供一组管理线程的过程
- **运行时系统：**完成线程的管理工作(操作、线程表)
- **内核管理的还是进程：**内核不知道线程的存在
- **线程切换不需要内核态特权**
- 例子：UNIX



2.8.1 线程的实现方式

◆ 用户级线程ULT(User level Threads)

- **Pthreads线程库：遵循POSIX(Portable Operating System Interface)标准，定义了创建和操作线程的一整套API**

Tread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread`s attribute structure
Pthread_attr_destroy	Remove a thread`s attribute structure

2.8.2 线程的实现

◆ 用户级线程ULT(User level Threads)

➢ 优点

- 线程切换不需转换到内核空间，节省了模式切换的开销
- 调度算法是应用程序特定的
- 用户级线程可运行在任何操作系统上（只要实现了线程库）

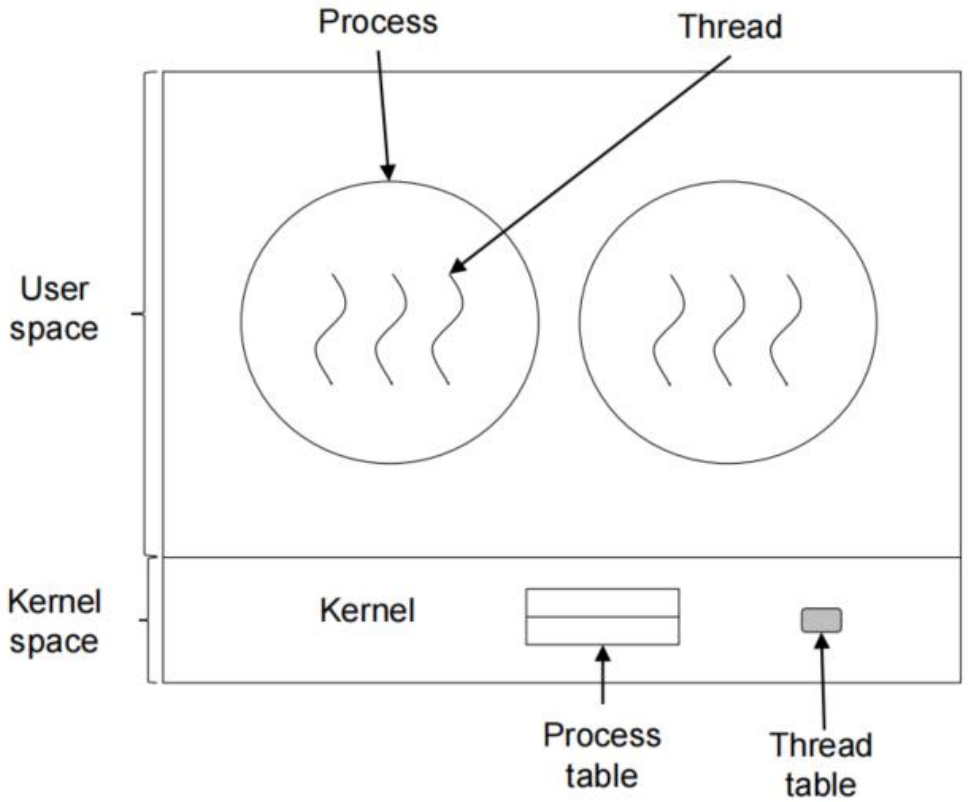
➢ 缺点

- 内核只将处理器分配给进程，同一进程中的两个线程不能同时运行在两个处理器上
- 大多数系统调用是阻塞的，因此，由于内核阻塞进程，故进程中的所有线程也被阻塞

2.8.2 线程的实现

◆ 内核支持线程 KST (Kernel Supported Threads)

- ➢ 内核管理所有线程管理，并向应用程序提供API
- ➢ 内核维护进程和线程的上下文
- ➢ 线程的切换需要内核支持
- ➢ 以线程为基础进行调度
- ➢ 例子：Windows



2.8.2 线程的实现

◆ 内核支持线程 KST(Kernel Supported Threads)

➢ 优点

- 在多处理器系统中，内核能够同时调度同一进程中的多个线程并行执行
- 如果进程中一个线程被阻塞，并不会同时阻塞同一进程中的其他线程
- 具有很小的数据结构和堆栈，线程的切换比较快
- 内核本身也可以采用多线程技术，提高执行速度和效率

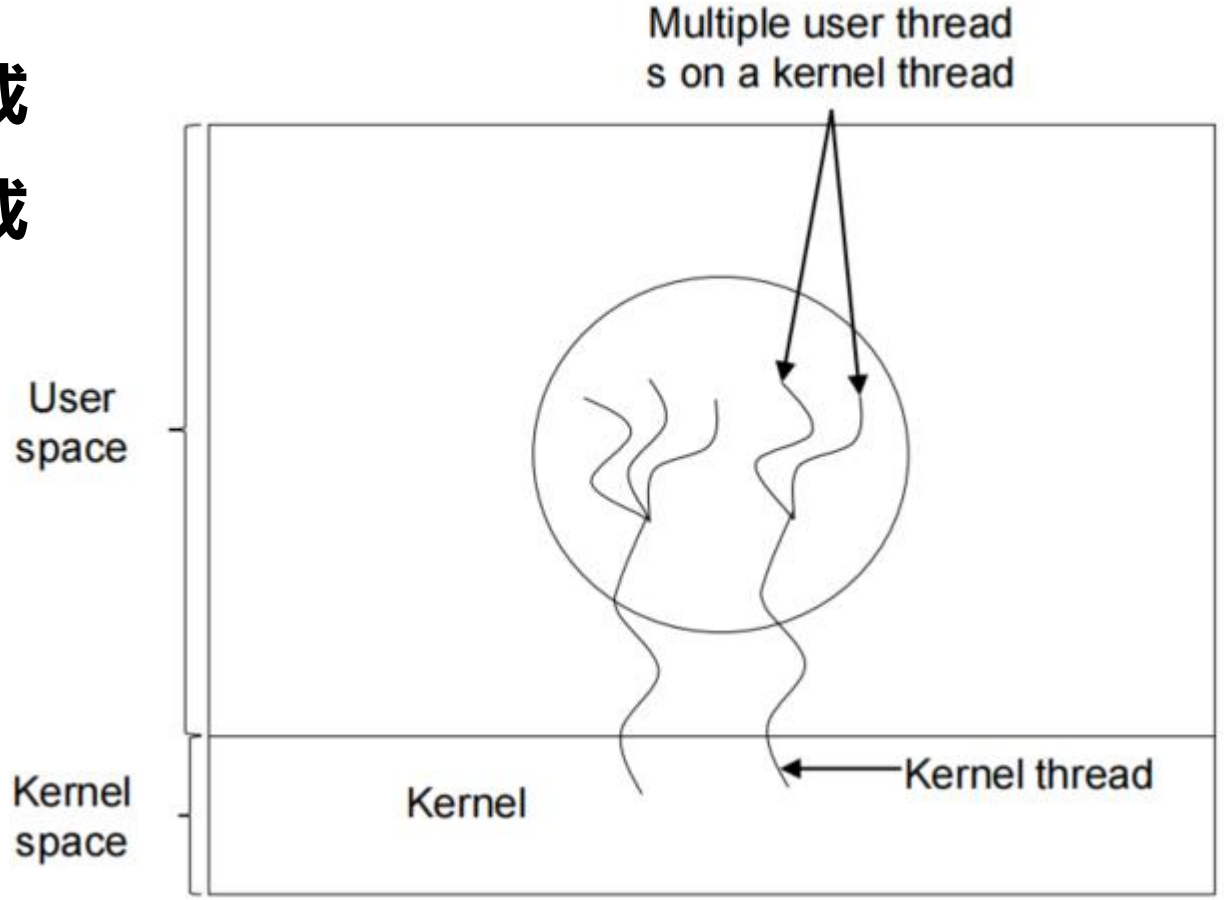
➢ 缺点

- 模式切换开销较大

2.8.2 线程的实现

◆ 组合方式

- ➢ 线程创建在用户空间完成
- ➢ 线程调度等在核心态完成
- ➢ 例子：Solaris



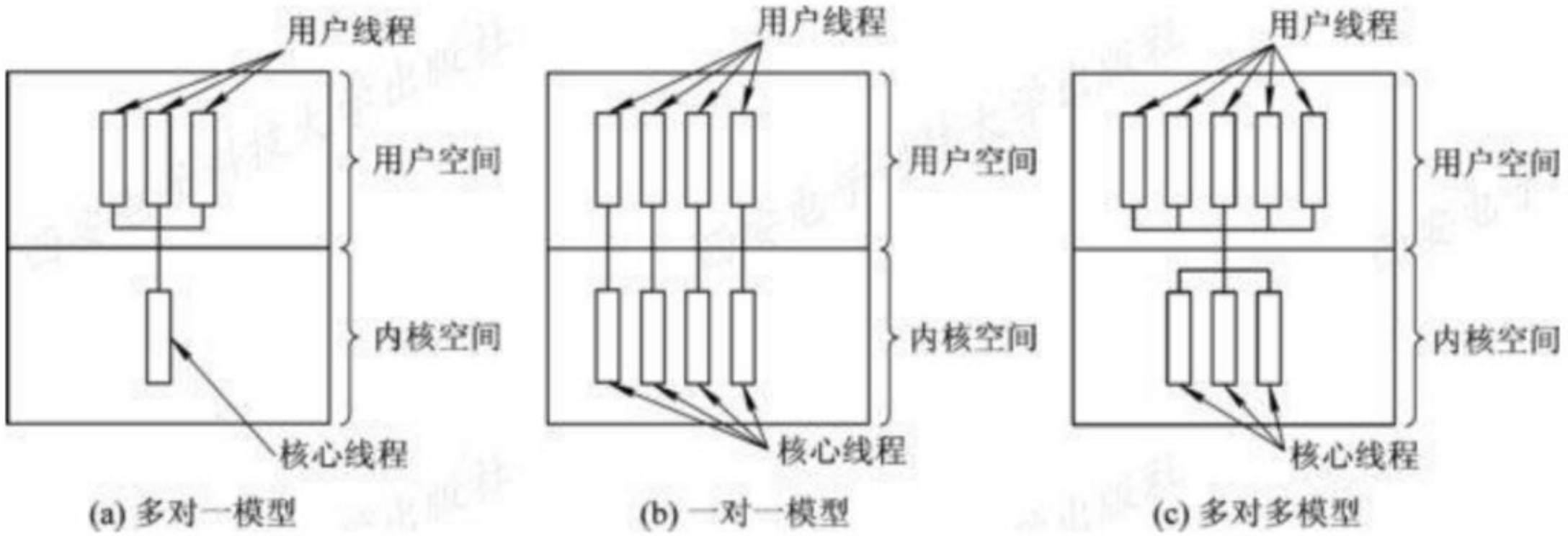
2.8.2 线程的实现

◆ 组合方式——三种模型

- **一对一：**为每一个用户线程都设置一个内核控制线程与之连接，当一个线程阻塞时，允许调度另一个线程运行。并行能力较强，但开销较大，因此需要限制整个系统的线程数（Windows 2000/NT等)
- **多对一：**将多个用户线程映射到一个内核控制线程，为了管理方便，这些用户线程一般属于一个进程，运行在该进程的用户空间，对这些线程的调度和管理也是在该进程的用户空间中完成。线程管理的开销小效率高，但当一个线程在访问内核时发生阻塞，整个进程都会被阻塞；在多处理机系统中，一个进程的多个线程无法实现并行。
- **多对多：**结合上述两种模型的优点，将多个用户线程映射到多个内核控制线程，内核控制线程的数目可以根据应用进程和系统的不同而变化，可以比用户线程少，也可以与之相同。

2.8.2 线程的实现

◆ 组合方式——三种模型



2.8.2 线程的实现

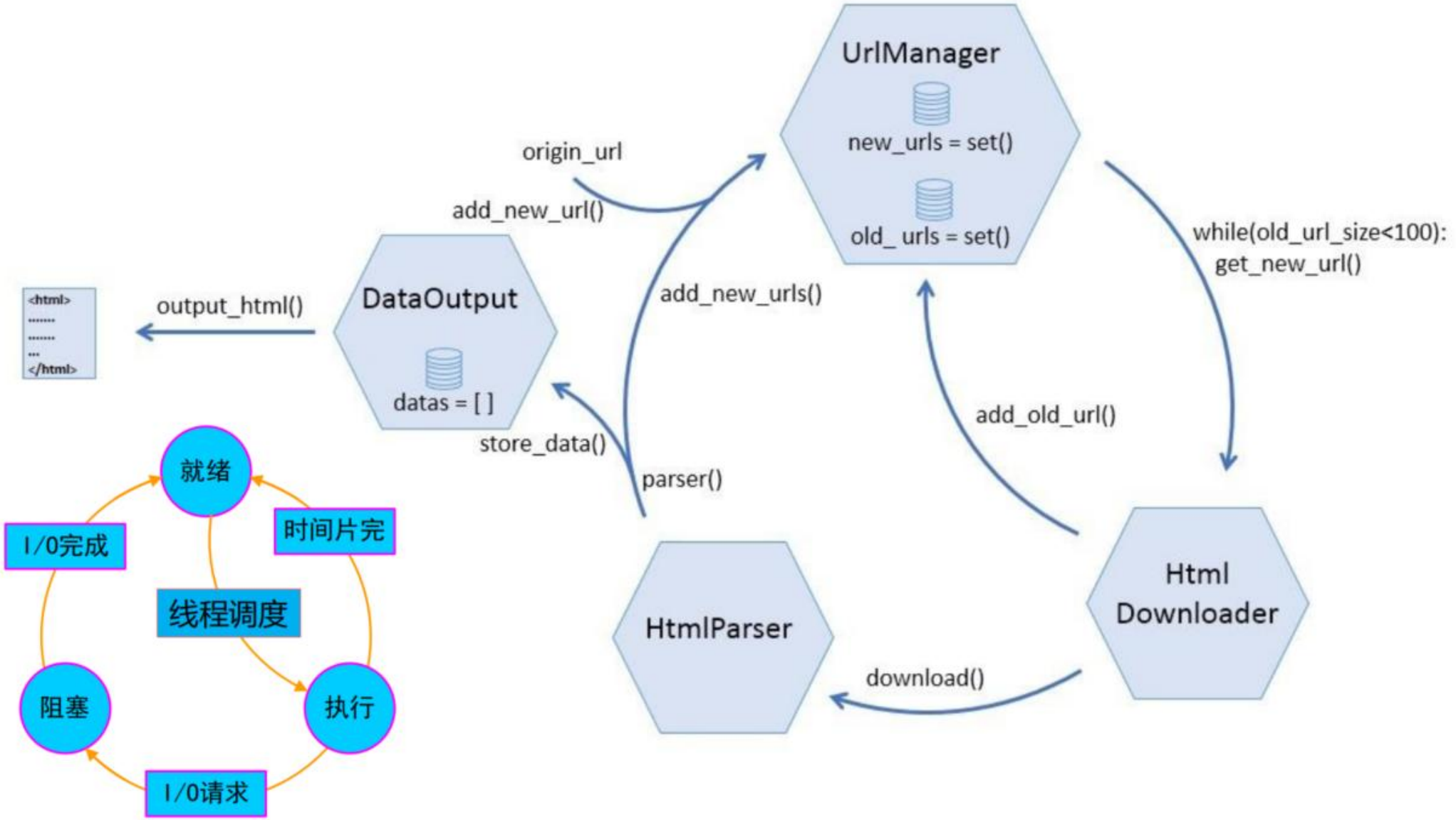
◆ 创建

- 应用程序在启动时，通常仅有一个线程在执行，人们把线程称为“初始化线程”，它的主要功能是用于创建新线程。在创建新线程时，需要利用一个线程创建函数(或系统调用)，并提供相应的参数。在线程的创建函数执行完后，将返回一个线程标识符供以后使用

◆ 终止

- 当一个线程完成了自己的任务(工作)后，或是线程在运行中出现异常情况而须被强行终止时，由终止线程通过调用相应的函数(或系统调用)对它执行终止操作

2.8.3 线程的实例

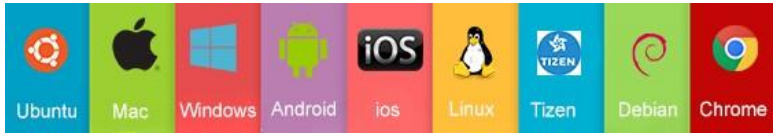

UrlManager
new_urls = set()
old_ urls = set()
origin_url
add_new_url()
add_new_urls()
while(old_url_size<100):
get_new_url()
DataOutput
datas = [ ]
output_html()
<html>
</html>
store_data()
parser()
add_old_url()
HtmlParser
Html
Downloader
download()
就绪
时间片完
I/O完成
线程调度
阻塞
执行
I/O请求

2.8.3 线程的实例

任务	任务类型	线程数
HtmlDownloader	I/O密集型操作	3
HtmlParser	CPU密集型操作	1
DataOutput	I/O密集型操作	1

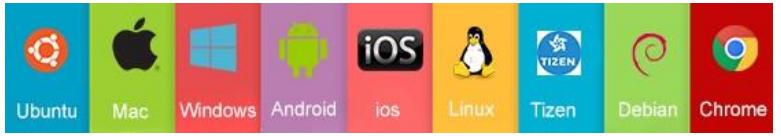
第三章

处理机调度与死锁


Ubuntu
Mac
Windows
Android
iOS
Linux
Tizen
Debian
Chrome

操作系统主要功能


处理机管理
存储器管理
设备管理
文件管理
用户接口
OS
功能



操作系统主要功能

- 在多道程序环境下，主存中有着多个进程，其数目往往多于处理机数目
- 要求系统能按某种算法，动态地把处理机分配给就绪队列中的一个进程，使之执行
- 分配处理机的任务是由处理机调度程序完成的

- 处理器管理：
  - **进程和线程的描述与控制**
  - **进程或线程的同步与互斥**
  - **进程之间及线程之间的通信**
  - **处理器调度**
  - **死锁的检测和预防**

处理机调度与死锁

处理机调度的层次和调度算法的目标

作业与作业调度 

进程调度

实时调度

死锁概述

预防死锁 

避免死锁

死锁的检测与解除

3.1.1 处理机调度的层次

◆ 处理机调度要解决的问题

- **What**：按什么原则分配
  - 调度算法
- **When**：何时分配
  - 调度的时机
- **How**：如何分配
  - 调度过程

3.1.1 处理机调度的层次

◆ 高级调度/长程调度/作业调度——作业

- ➢ 根据某种算法决定将外存上处于后备队列中的哪几个作业作业调入内存，为其创建进程、分配必要资源，放入就绪队列
- ✓ 多道批处理系统
- × 分时系统和实时系统

◆ 低级调度/短程调度/进程调度——进程/内核级线程

- ➢ 根据某种算法决定就绪队列中的哪个进程获得处理机，并分派程序将处理机分配给被选中的进程
- ✓ 多道批处理系统、分时系统、实时系统

3.1.1 处理机调度的层次

◆ 中级调度/中程调度/内存调度

- 提高内存**利用率和系统吞吐量**
- **就绪驻外存状态/挂起状态**：使那些暂时不能运行的进程不再占用宝贵的内存资源，而将它们调至外存上去等待
- 当这些进程重又具备运行条件且内存又稍有空闲时，由中级调度来决定把外存上的那些又具备运行条件的就绪进程**重新调入内存**，并修改其状态为**就绪状态**，挂在就绪队列上等待进程调度
- 中级调度实际上就是**存储器管理**中的**对换功能**

3.1.1 处理机调度的层次

◆ 高级调度/长程调度/作业调度

- 在一个(批)作业运行完毕退出系统，而需重新调入一个(批)作业进入内存时发生，周期较长，大约几分钟一次

◆低级调度/短程调度/进程调度/微观调度

- 运行频率最高，时间尺度毫秒级，分时系统（10～100 ms 进行一次）
- 频繁使用，要求在实现时做到高效

◆中级调度/中程调度/内存调度

- 运行频率基本上介于上述两种调度之间

3.1.2 处理机调度算法的目标

◆ 处理机调度算法的共同目标

- **资源利用率**：使处理机和其他资源都尽可能保持忙碌
- **公平性**：进程都能获得合理的CPU时间，不会发生进程饥饿现象
- **平衡性**：保持系统资源使用的平衡性
- **策略强制执行**：只要需要，就必须予以准确地执行

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

3.1.2 处理机调度算法的目标

◆ 批处理系统的目标

➢ 作业周转时间和平均周转时间短

- 作业在后备队列等待作业调度时间
- 进程在就绪队列等待进程调度时间
- 进程在CPU的执行时间
- 进程等待I/O操作完成的时间
- W=作业周转时间/系统服务的时间

$$T = \frac{1}{n}\left[\sum_{i=1}^{n} T_i\right] \quad W = \frac{1}{n}\left[\sum_{i=1}^{n} \frac{T_i}{T_s}\right]$$

➢ 系统吞吐量高

- 单位时间内系统所完成的作业数→短作业

➢ 处理机利用率高

- 计算量大的作业

要求存在一定矛盾

3.1.2 处理机调度算法的目标

◆ 分时系统的目标

- **响应时间快（进程调度算法的重要准则）**
  - 请求输入到处理机的时间
  - 处理机处理信息的时间
  - 响应信息回到终端显示器的时间
- **均衡性：系统响应时间的快慢与服务复杂性相适应**

◆ 实时系统的目标

- **截止时间的保证：**对于HRT必须确保截止时间要求
- **可预测性：**实时系统提供了请求的可预测性，缓冲可以可提高实时性

处理机调度与死锁

- 处理机调度的层次和调度算法的目标
- 作业与作业调度 
- 进程调度
- 实时调度
- 死锁概述
- 预防死锁 
- 避免死锁
- 死锁的检测与解除

处理机调度与死锁

处理机调度的层次和调度算法的目标

作业与作业调度

- 批处理系统中的作业
- 作业调度的主要任务
- 先来先服务和短作业优先
- 优先级调度和高响应比优先调度

3.2.1 批处理系统中的作业

◆ 作业（Job）——比程序更为广泛的概念

➢ 概念

- 作业由一组统一管理和操作的**进程集合**构成，是用户要求计算机系统完成的一项相对独立的工作。
- 作业可以是完成了编译、链接之后的**用户程序**，也可以是用各种命令构成的一个**脚本**
- 作业通过相应输入设备输入到磁盘存储器，保存在**后备队列（外存）**，由作业调度程序调入内存
- 输入作业流：若干作业进入系统后，被依次存放在外存上
- 处理作业流：在操作系统的控制下，逐个作业进行处理

3.2.1 批处理系统中的作业

◆ 作业（Job）——比程序更为广泛的概念

➢ 分类

- 根据需要处理工作的类型，分为**计算型和I/O型**作业
- 按照作业提交方式，分为**批处理作业和终端型作业**

➢ 一个系统能够接纳作业的个数称为系统的多道程序度

➢ 作业步（JOB STEP）

- 每个作业经过若干个相互独立、相互关联的顺序加工步骤得到结果，每个加工步骤就是作业步
- 典型作业步：编译、链接装配、运行

3.2.1 作业、进程、程序

- **作业：**用户提交给系统的一个任务，在用户向计算机提交作业后，系统将它放入**外存**中的作业等待队列中等待执行。
- **进程：**完成用户任务的执行实体，是向系统申请分配资源的基本单位。任一进程，只要它被创建，总有相应的部分存在于**内存**中。
- 一个作业通常包括几个进程，几个进程共同完成一个任务，且必须至少由一个进程组成。
- 用户提交作业以后，当作业被调度，系统会为作业创建进程，一个进程无法完成时，系统会为这个进程创建子进程。
- 作业与进程最主要的区别是：前者是由**用户**提交，后者是由**系统**自动生成；前者以用户任务为单位，后者是操作系统控制的单位。

3.2.1 作业、进程、程序

- **进程：系统进行资源分配和调用的基本单位（低级调度）**
- **作业：用户需要计算机完成的某项任务，要求计算机所做工作的集合（高级调度）**
- **联系**
  - **一个作业通常包括多个进程，多个进程共同完成一个任务，即作业**
  - **用户提交作业后，当作业被提交后，系统会为作业自动创建进程，一个进程无法完成后，系统会为它再创建子进程（进程树）**
- **区别**
  - **作业的概念用于批处理系统。**
  - **进程的概念用于几乎所有的多道程序系统中。**
  - **进程是一个程序在一个数据集上的一次执行，而作业是用户提交给系统的一个任务。**

3.2.1 作业、进程、程序

- 一个作业通常包括程序、数据和操作说明书3部分
- 每一个进程由PCB、程序和数据集合组成
- 程序是进程的一部分，是进程的实体
- 一个作业可划分为若干个进程来完成，而每一个进程有其实体：程序和数据集合

3.2.1 批处理系统中的作业

◆作业控制块 JCB(Job Control Block)

- 作业在系统中存在的**标志**，保存了系统对**作业**进行**管理和调度**所需的全部信息。
- 每当作业进入系统时，系统便为每个作业建立一个 JCB，根据作业类型将它插入相应的**后备队列**中。
- 作业调度程序依据一定的调度算法来调度它们，被调度到的作业将会装入**内存**。
- 通常应包含的内容有：**作业标识**、用户名称、用户帐户、**作业类型**(CPU 繁忙型、I/O 繁忙型、批量型、终端型)、**作业状态**、调度信息(**优先级**、作业已运行时间)、**资源需求**(预计运行时间、 要求内存大小、要求 I/O 设备的类型和数量等)、进入系统时间、开始处理时间、作业完成时间、作业退出时间、资源使用情况等。

3.2.1 批处理系统中的作业

◆ 作业状态

➢ 后备状态（收容）

- 作业已经过SPOOLing系统输入到磁盘输入井，等待调入内存运行。
- 为了管理和调度作业，为每个作业设置一个JCB， JCB记录了作业类型和资源要求等有关信息。
- JCB按作业类型组成一个或多个后备作业队列。

3.2.1 批处理系统中的作业

◆ 作业状态

➢ 运行状态

- 一个在后备作业队列的作业被作业调度程序选中后，分配必要的资源，建立一组相应的**进程**后，**调入内存**
- **进程各状态**（进程运行态、内存进程就绪态、内存阻塞态、外存进程就绪态、外存进程阻塞态等）都对应**作业运行状态**。

➢ 完成状态

- 当进程正常运行结束或因发生错误而终止时，作业进入完成状态。终止作业程序将负责善后处理。

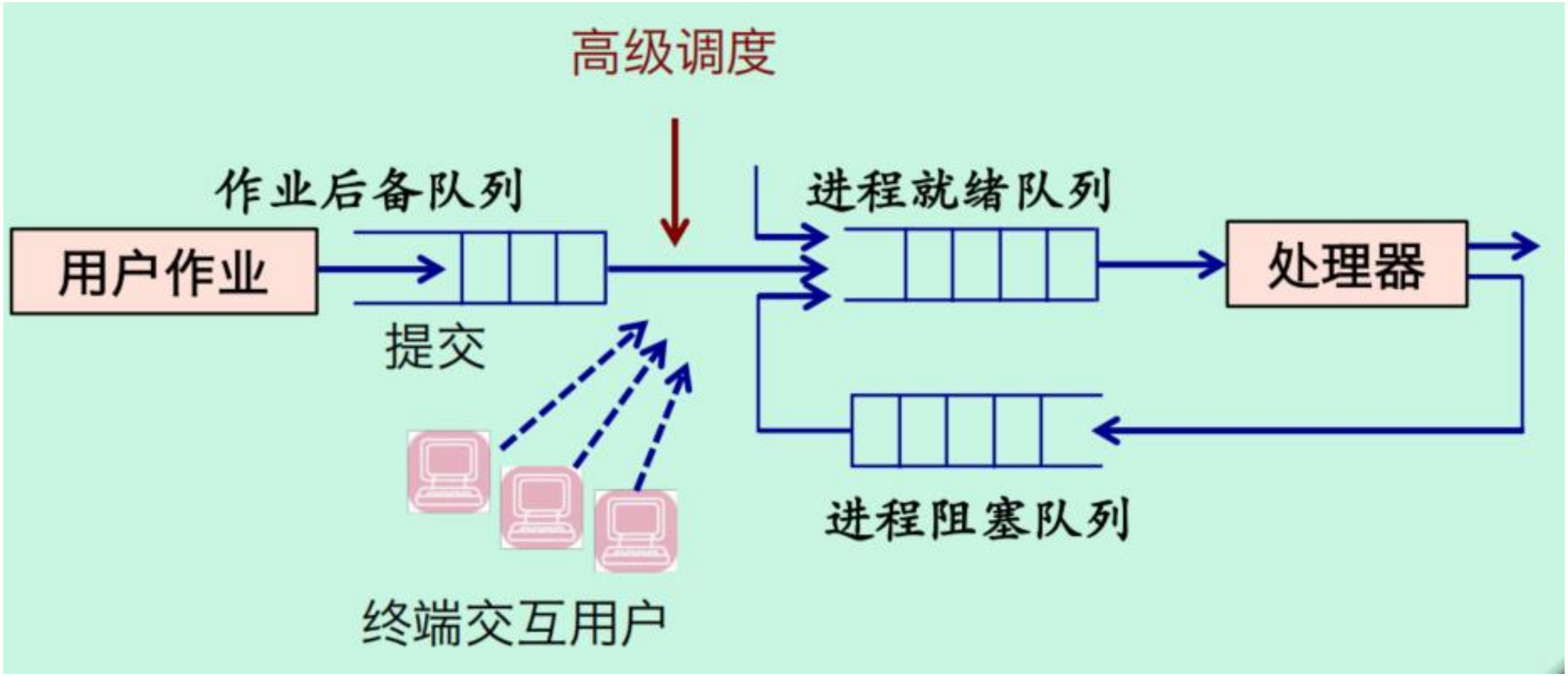
3.2.2 作业调度的主要任务

◆ 作业调度 = 接纳调度(Admission Scheduling)

- 作业调度的主要功能是根据作业控制块中的信息，审查系统能否满足用户作业的资源需求，以及按照一定的算法，从外存的后备队列中选取某些作业调入内存，并为它们创建进程、分配必要的资源。再将新创建进程插入就绪队列，准备执行。
- **对用户而言**，希望自己作业的周转时间尽可能的少
- **对系统来说**，希望作业的平均周转时间尽可能少，有利于提高 CPU 的利用率和系统的吞吐量。
- 每个系统在选择作业调度算法时，既应考虑用户的要求，又能确保系统具有较高的效率。

3.2.2 作业调度的主要任务

◆ **作业调度 = 接纳调度(Admission Scheduling)**



3.2.2 作业调度的主要任务

◆ 决定接纳多少个作业

- 作业调度每次要接纳多少个作业进入内存，取决于**多道程序度(Degree of Multiprogramming)**，即允许多少个作业同时在内存中运行
- 当内存中同时运行的作业数目太多时，可能会影响到系统的服务质量
- 在内存中同时运行作业的数量太少时，会导致系统的资源利用率和系统吞吐量太低
- 多道程序度的确定应根据系统的规模和运行速度等情况做适当的折衷

3.2.2 作业调度的主要任务

◆ 决定接纳哪些作业

- 应将哪些作业从外存调入内存，取决于所采用的调度算法
  - 先来先服务调度算法（最简单）
    - ✓ **First-Come First-Served, FCFS**
  - 短作业优先调度算法（较常用）
    - ✓ **Shortest Job/Process First, SJF/SPF**
  - 基于作业优先级调度算法（较常用）
    - ✓ **Priority Scheduling Algorithm, PSA**
  - 响应比高者优先调度算法（比较好）
    - ✓ **Highest Response Ratio Next/Frist, HRRN/HRRF**

3.2.3 先来先服务（FCFS）调度算法

◆ 先来先服务（First-Come First Served, FCFS）

- 既可用于作业调度，也可用于进程调度。
- **作业调度**：每次调度都是从**后备作业队列**中选择一个或多个最先进入该队列的作业，将它们**调入内存**，为它们分配资源、创建进程，然后放入**就绪队列**。
- **进程调度**： 每次调度是从**就绪队列**中选择一个最先进入该队列的进程，为之**分配处理机**，使之投入运行。该进程一直运行到完成或发生某事件而阻塞后才放弃处理机。
- 有利于**长作业(进程)**，而不利于短作业(进程)

3.2.3 先来先服务（FCFS）调度算法

◆ 先来先服务（First-Come First Served, FCFS）

- 有利于 **CPU 繁忙型的作业**，而不利于 **I/O 繁忙型的作业**。CPU 繁忙型作业是指该类作业需要大量的 CPU 时间进行计算，而很少请求 I/O。通常的**科学计算**便属于 CPU 繁忙型作业。I/O 繁忙型作业是指 CPU进行处理时需频繁地请求 I/O。目前的**大多数事务处理**都属于 I/O 繁忙型作业。

进程名	到达时间	服务时间	开始执行时间	完成时间	周转时间	带权周转时间
A	0	1				
B	1	100				
C	2	1				
D	3	100				

3.2.3 先来先服务（FCFS）调度算法

◆ 先来先服务（First-Come First Served, FCFS）

- 有利于 **CPU 繁忙型的作业**，而不利于 **I/O 繁忙型的作业**。CPU 繁忙型作业是指该类作业需要大量的 CPU 时间进行计算，而很少请求 I/O。通常的**科学计算**便属于 CPU 繁忙型作业。I/O 繁忙型作业是指 CPU进行处理时需频繁地请求 I/O。目前的**大多数事务处理**都属于 I/O 繁忙型作业。

进程名	到达时间	服务时间	开始执行时间	完成时间	周转时间	带权周转时间
A	0	1	0	1	1	1
B	1	100	1	101	100	1
C	2	1	101	102	100	100
D	3	100	102	202	199	1.99

3.2.3 短作业(进程)优先调度算法

◆ 短作业/进程优先 (Short Job/Process First, SJF/SPF)

- 对短作业或短进程优先调度的算法
- **短作业优先(SJF)调度算法**：从**后备队列**中选择一个或若干个估计运行时间**最短的作业**，将它们调入内存运行
- **短进程优先(SPF)调度算法**：从**就绪队列**中选出一个估计运行时间**最短的进程**，将处理机分配给它，使它立即执行并一直执行到完成，或发生某事件而被阻塞放弃处理机时再重新调度

3.2.3 短作业(进程)优先调度算法

◆ 短作业/进程优先 (SJF/SPF)缺点

- **必须预知作业的运行时间**：很难估计，一般会偏长估计
- 该**对长作业不利**，长作业的周转时间会明显增长，甚至出现饥饿现象
- 完全**未考虑作业紧迫程度**，不能保证紧迫性作业(进程)会被及时处理
- 人机**无法实现交互**
- 由于作业(进程)的长短只是根据用户所提供的估计执行时间而定的，而**用户**又可能会**有意或无意地缩短其作业的估计运行时间**，致使该算法不一定能真正做到短作业优先调度。

3.2.3 短作业(进程)优先调度算法

◆ **FCFS vs SJF/SPF**

作业情况 / 调度算法	进程名	A	B	C	D	E	平　均
	到达时间	0	1	2	3	4	
	服务时间	4	3	5	2	4	
FCFS (a)	完成时间						
	周转时间						
	带权周转时间						
SJF (b)	完成时间						
	周转时间						
	带权周转时间						

3.2.3 短作业(进程)优先调度算法

◆ **FCFS vs SJF/SPF**

作业/情况 调度/算法	进程名	A	B	C	D	E	平　均
	到达时间	0	1	2	3	4	
	服务时间	4	3	5	2	4	
FCFS (a)	完成时间	4	7	12	14	18	
	周转时间	4	6	10	11	14	9
	带权周转时间	1	2	2	5.5	3.5	2.8
SJF (b)	完成时间	4	9	18	6	13	
	周转时间	4	8	16	3	9	8
	带权周转时间	1	2.67	3.1	1.5	2.25	2.1

3.2.4 优先级调度算法

◆ 优先级调度算法 (Priority-Sheduling Algorithm, PSA)

- **FCFS调度算法：作业等待时间是作业的优先级**
- **SJF/SPF调度算法：**作业/进程的长短是作业/进程的优先级
- **PSA：**基于作业的紧迫程度，由外部赋予作业响应的优先级

3.2.4 高响应比优先调度算法

◆ 高响应比优先 (Highest Response Ratio Next, HRRN)

- 为每个作业引入动态优先权，使作业的优先级随着等待时间的增加而以速率a提高，则长作业在等待一定的时间后，必然有机会分配到处理机。该优先权的变化规律可描述为：

$$\text{优先权} = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}}$$

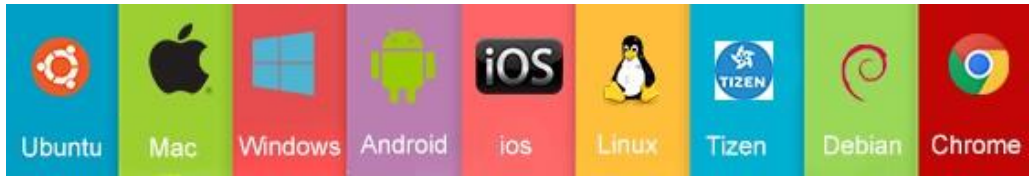
- 等待时间与服务时间之和就是系统对该作业的响应时间，故该优先权又相当于响应比 RP。据此，又可表示为：

$$R_P = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} = \frac{\text{响应时间}}{\text{要求服务时间}}$$

3.2.4 高响应比优先调度算法

◆ 高响应比优先 (Highest Response Ratio Next, HRRN)

- 如果作业的等待时间相同，则要求服务的时间愈短，其优先权愈高，因而该**算法有利于短作业**。
- 当要求服务的时间相同时，作业的优先权决定于其等待时间，等待时间愈长，其优先权愈高（**先来先服务**）
- 对于**长作业**，作业的优先级可以随等待时间的增加而提高，当其等待时间足够长时， 其优先级便可升到很高，从而也可获得处理机。
- 既照顾了短作业，又考虑了作业到达的先后次序，不会使长作业长期得不到服务，该算法实现了一种**较好的折衷**。
- 在利用该算法时，每进行调度之前，都须先做**响应比的计算**，这会增加**系统开销**。



– 如果4个作业A、B、C、D到达系统磁盘后备队列的顺序和需要执行的时间如下表所示。

作业	到达时间（ms）	需要执行时间（ms）
A	0	20
B	5	15
C	10	10
D	15	5

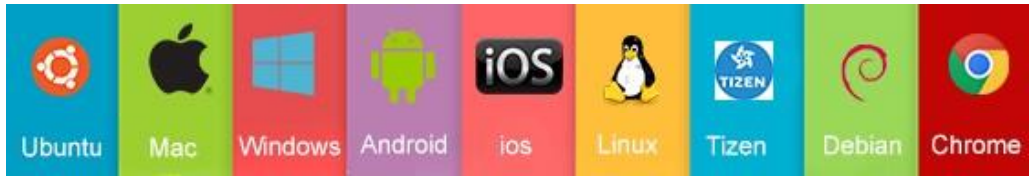
FCFS

SJF

HRRN

– 请给出作业执行的图示（横轴为时间，纵轴为作业）
– 请计算每个作业的周转时间和带权周转时间
– 请计算上述作业的平均周转时间和平均带权周转时间

《操作系统原理》2024课程



作业	到达时间	需要处理时间	FCFS 周转时间	SJF 周转时间	HRRN 周转时间
A	0	20	20	20	20
B	5	15	30	45	30
C	10	10	35	25	40
D	15	5	35	10	25
平均周转时间			30.00	25.00	28.75
平均带权周转时间			3.38	1.97	3.00

思考：对上述比较结果进行分析和评价。

处理机调度与死锁

- 处理机调度的层次和调度算法的目标
- 作业与作业调度 
- 进程调度
- 实时调度
- 死锁概述
- 预防死锁 
- 避免死锁
- 死锁的检测与解除

处理机调度与死锁

处理机调度的层次和调度算法的目标

作业与作业调度 

进程调度

- 任务机制和方式
- 轮转调度算法
- 优先级调度算法
- 多队列调度算法
- 多级反馈队列
- 基于公平原则的调度算法

3.3.1 进程调度的任务、机制和方式

◆ 进程调度 = 低级调度 = 短程调度（操作最频繁的调度）

➢ 进程调度的任务

- **保存处理机的现场信息：**如程序计数器、多个通用寄存器内容等
- **按某种算法选取进程：**按某种算法从就绪队列中选取一个进程改为运行状态，并准备把处理机分配给它。
- **把处理器分配给进程：**由分派程序(Dispatcher)把处理器分配给进程，恢复处理机现场，把选中进程的PCB内有关处理机现场的信息装入处理器相应寄存器中，把处理器的控制权交给该进程，让它从取出的断点处继续运行

3.3.1 进程调度的任务、机制和方式

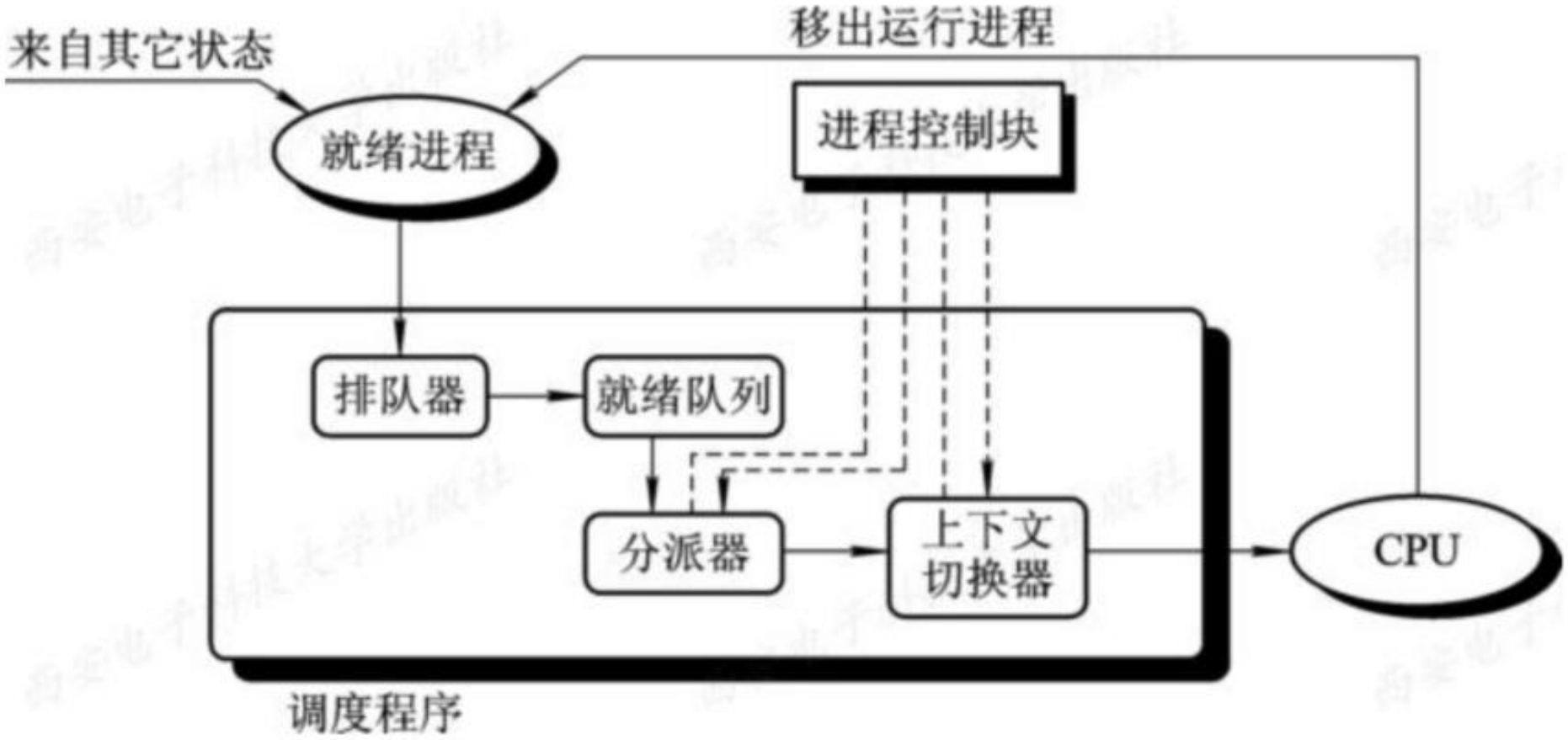
◆ 进程调度 = 低级调度 = 短程调度（操作最频繁的调度）

➢ 进程调度的机制

- **排队器**：事先将系统中所有的就绪进程按照一定的方式排成一个或多个队列，以便调度程序能最快地找到它
- **分派器(分派程序)**：把由进程调度程序所选定的进程，从就绪队列中取出进行上下文切换，将处理机分配给它
- **上下文切换器：**两对上下文切换操作，1）保存当前进程的上下文，而装入分派程序的上下文，以便分派程序运行；2）将移出分派程序，而把新选进程的CPU 现场信息装入到处理机的各个相应寄存器中。

3.3.1 进程调度的任务、机制和方式

◆ **进程调度 = 低级调度 = 短程调度（操作最频繁的调度）**



3.3.1 进程调度的任务、机制和方式

◆ 进程调度方式

➢ 非抢占方式 (Nonpreemptive Mode)

- 一旦把处理机分配给某进程后，让它一直运行下去，直至该进程完成，或发生某事件而被阻塞时，才再把处理机分配给其他进程。
- 引起进程调度的**因素**：
  - ✓ 正在执行的进程执行完毕，或因发生某事件而不能再继续执行；
  - ✓ 执行中的进程因提出 I/O 请求而暂停执行；
  - ✓ 在进程通信或同步过程中执行了某种原语操作，如P操作(wait 操作)、Block 原语、Wakeup 原语等。
- **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——立即执行

3.3.1 进程调度的任务、机制和方式

◆ 进程调度方式

➢ 抢占方式 (Preemptive Mode)

- 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。
- **优点：**防止长进程长时间占用处理机，为大多数进程提供更公平的服务，能满足对响应时间有着较严格要求的实时任务的需求。
- **缺点：**比非抢占方式调度所需付出的开销较大。

原则

- **优先权原则：**重要和紧急的作业赋予较高的优先权。
- **短作业(进程)优先原则：**新到达的作业(进程)比正在执行的作业(进程)明显的短；即短作业(进程)可抢占当前较长作业(进程)的处理机
- **时间片原则：**各进程按时间片轮流运行，当一个时间片用完后，便停止该进程的执行而重新进行调度

3.3.2 进程调度的主要算法

- 轮转调度算法
  - ✓ **Time Round-Robin, TRR/RR**
- 优先级调度算法
  - ✓ **Priority Scheduling Algorithm, PSA**
- 多队列调度算法
  - ✓ **Multi-level Queue**
- 多级反馈队列调度算法
  - ✓ **Multi-level Feed Queue**
- 基于公平原则的调度算法

3.3.2 轮转调度算法 TRR/RR

◆ 基本思想

- 分时思想；
- 首先将处理器的处理时间划分为大小相等的时间片；
- 就绪队列按照FCFS形成；
- 调度程序每次从就绪队列中选择队首的进程，为之分配处理器的一个时间片并让进程运行；
- 当进程运行的时间片到时，强迫进程放弃处理器，到就绪队列中再次排队，并将处理器的下一个时间片分配给就绪队列中队首的进程；
- 所有就绪队列的进程按照这样的形式轮转使用处理器时间片

3.3.2轮转调度算法 TRR/RR

◆ 进程切换时机

➢ 在何时进行进程的切换，分为两种情况

- 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，将它从就绪队列中删除，再调度就绪队列中队首的进程运行，并启动一个新的时间片。
- 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的末尾。

➢ 时间片大小的确定

- 时间片的大小对**系统性能**有很大的影响，
- **时间片过小**：频繁发生中断、进程切换，增加系统的开销；
- **时间片过大**：使得每个进程都能在一个时间片内完成，时间片轮转算法便退化为 FCFS 算法，无法满足交互式用户的需求。

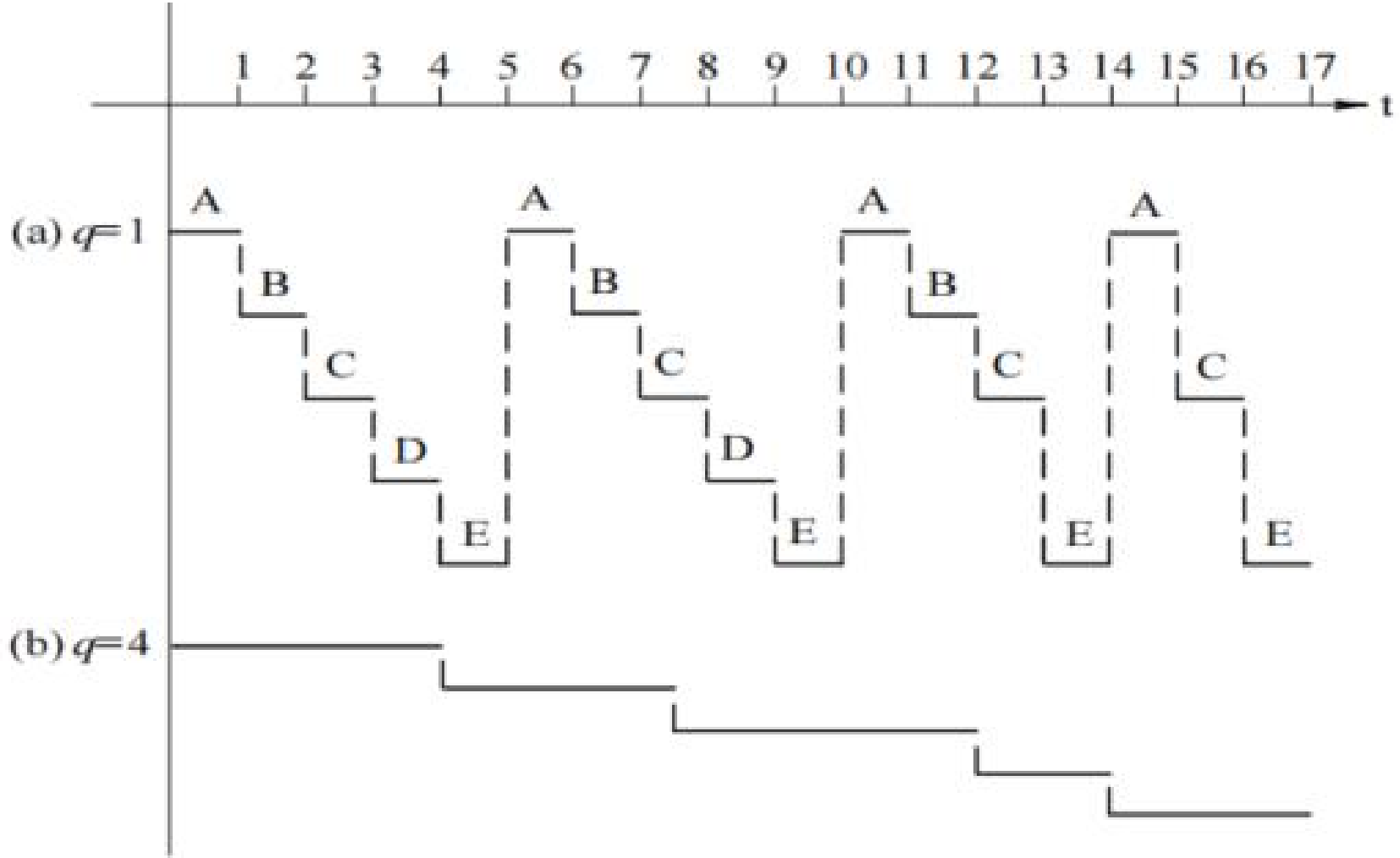
3.3.2轮转调度算法 TRR/RR

◆q=1和q=4时进程的周转时间

作业 情况 / 时间片	进程名	A	B	C	D	E	平均
	到达时间	0	1	2	3	4	
	服务时间	4	3	4	2	4	
RR q=1	完成时间						
	周转时间						
	带权周转时间						
RR q=4	完成时间						
	周转时间						
	带权周转时间						

3.3.2轮转调度算法 TRR/RR

◆q=1和q=4时进程的周转时间



3.3.2轮转调度算法 TRR/RR

◆q=1和q=4时进程的周转时间

作业 情况 / 时间片	进程名	A	B	C	D	E	平均
	到达时间	0	1	2	3	4	
	服务时间	4	3	4	2	4	
RR q=1	完成时间	15	12	16	9	17	
	周转时间	15	11	14	6	13	11.8
	带权周转时间	3.75	3.67	3.5	3	3.33	3.46
RR q=4	完成时间	4	7	11	13	17	
	周转时间	4	6	9	10	13	8.4
	带权周转时间	1	2	2.25	5	3.33	2.5

3.3.2 轮转调度算法 TRR/RR

- 进程A、B、C、D需要处理的时间分别为20ms、10ms、15ms、5ms，在0时间达到。达到的先后顺序为A→B→C→D。

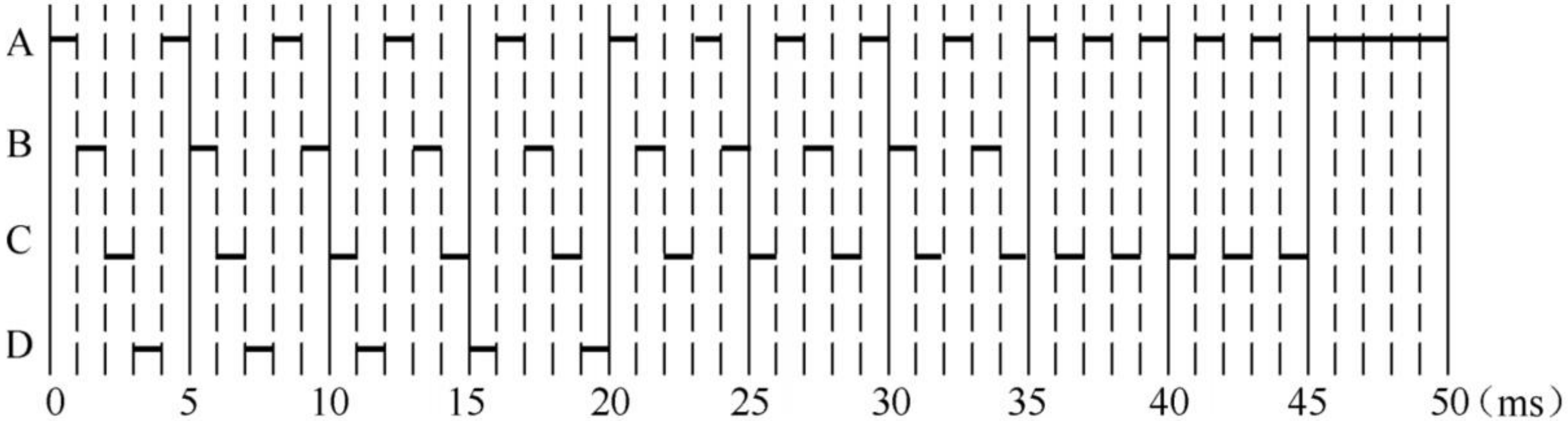
- 请给出不同时间片下的调度图示；
- 请计算不同时间片下的平均周转时间。

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

  - 1）时间片为1
  - 2）时间片为5
  - 3）时间片为10

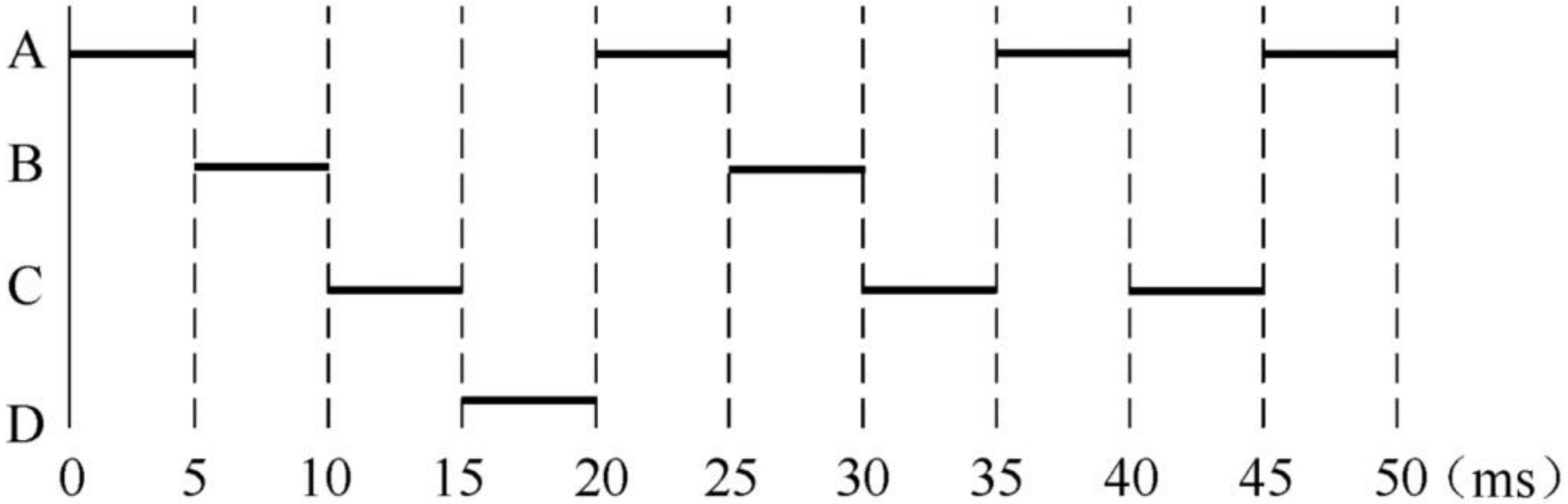
◆ 例题

– 进程A、B、C、D需要处理的时间分别为20ms、10ms、15ms、5ms，在0时间达到。达到的先后顺序为A→B→C→D。



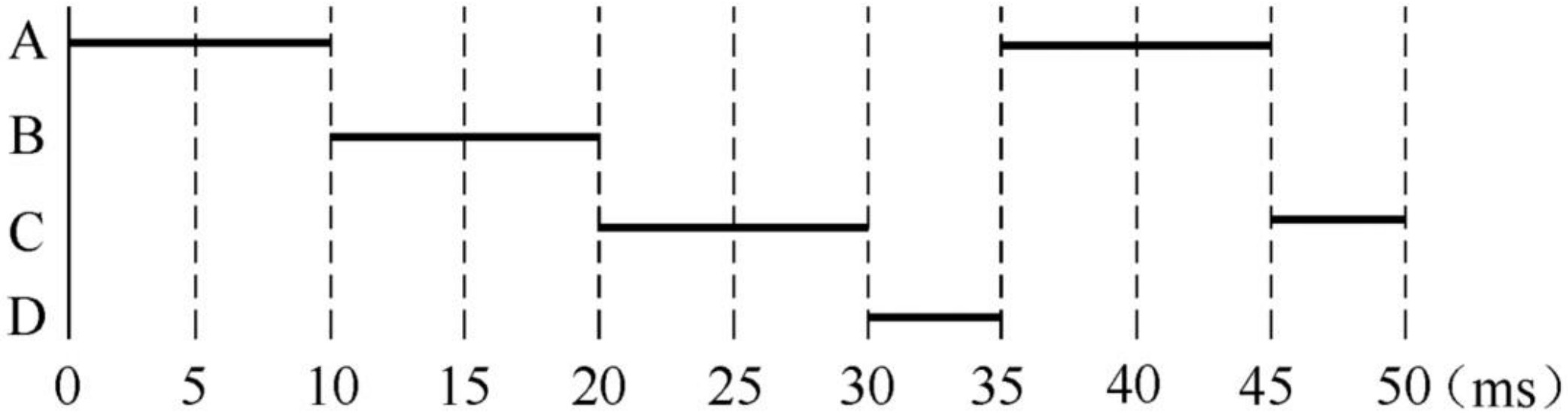
◆ 例题

– 进程A、B、C、D需要处理的时间分别为20ms、10ms、15ms、5ms，在0时间达到。达到的先后顺序为A→B→C→D。



◆ 例题

– 进程A、B、C、D需要处理的时间分别为20ms、10ms、15ms、5ms，在0时间达到。达到的先后顺序为A→B→C→D。



3.3.3 优先级调度算法 PSA

◆ 优先级调度算法的类型

- ➢ 处理器分配给就绪队列中优先级最高的进程
- ➢ 非抢占式优先级调度算法：执行完成或放弃
- ➢ 抢占式优先级调度算法：优先级高者剥夺低者

◆ 优先级类型

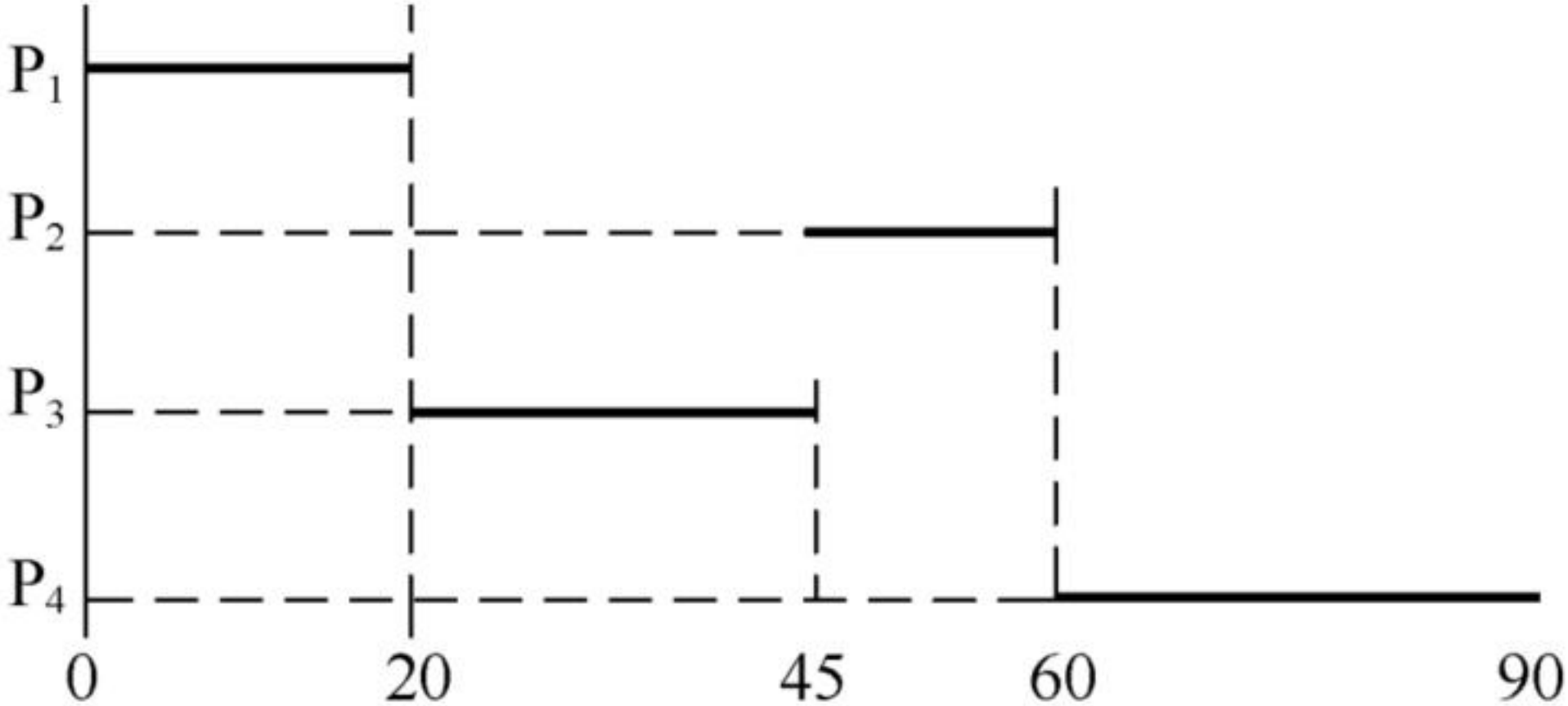
- ➢ 静态优先级（保持不变）
- ➢ 确定依据
  - 进程类型
  - 进程对资源的需求
  - 用户要求分派器
- ➢ 动态优先级（随时间改变）
  - 等待时间的影响因素
  - 避免长作业垄断处理机

3.3.3 优先级调度算法 PSA

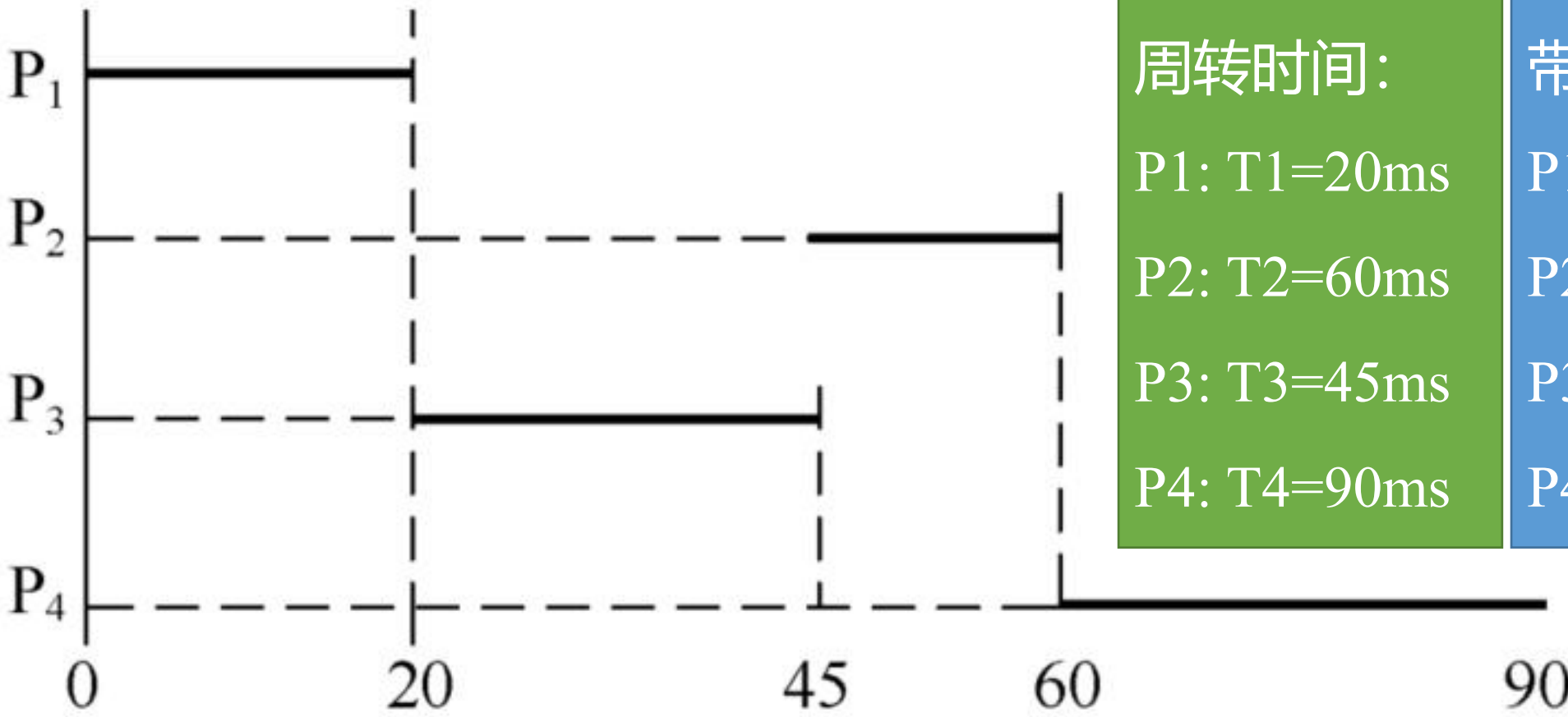
- 同时达到的进程P1、P2、P3、P4，它们的优先数分别为80、70、75、65，数字越大表示进程优先级越高，
  处理的时间分别是20ms、15ms、25ms、30ms
  。

- 请给出采用优先级算法的调度图示；
- 请计算平均周转时间和平均带权周转时间。

– 同时达到的进程P1、P2、P3、P4，它们的优先数分别为80、70、75、65，数字越大表示进程优先级越高，需要处理的时间分别是20ms、15ms、25ms、30ms



– 同时达到的进程P1、P2、P3、P4，它们的优先数分别为80、70、75、65，数字越大表示进程优先级越高，需要处理的时间分别是20ms、15ms、25ms、30ms



周转时间：

P1: T1=20ms

P2: T2=60ms

P3: T3=45ms

P4: T4=90ms

带权周转时间：

P1: W1=20/20

P2: W2=60/15

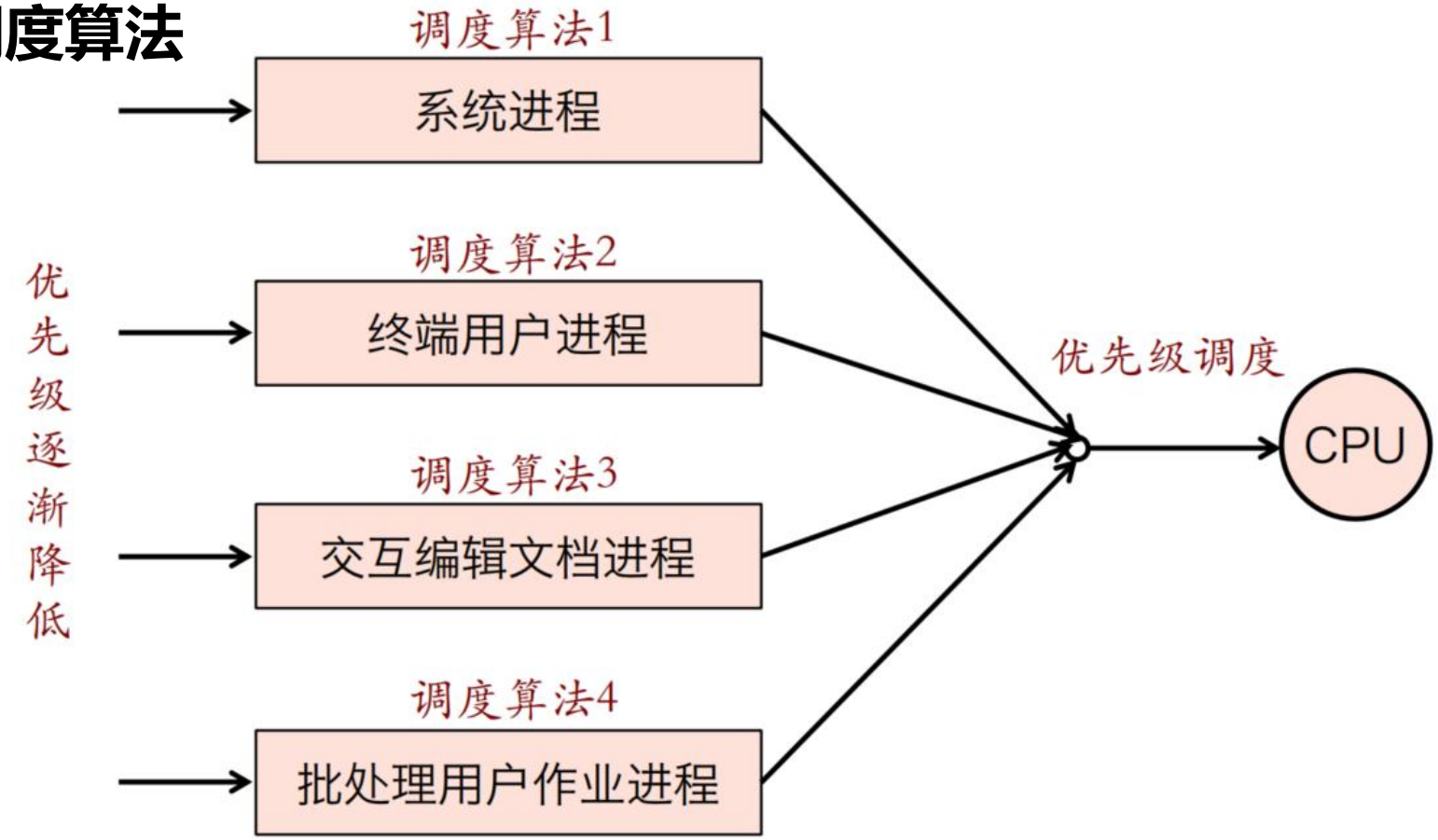
P3: W3=45/25

P4: W4=90/30

3.3.4 多队列调度算法

◆ 基本思想

➢ 处理器将就绪队列拆分为若干个，将不同类型或性质的进程固定分配在不同的就绪队列，采用不同调度算法

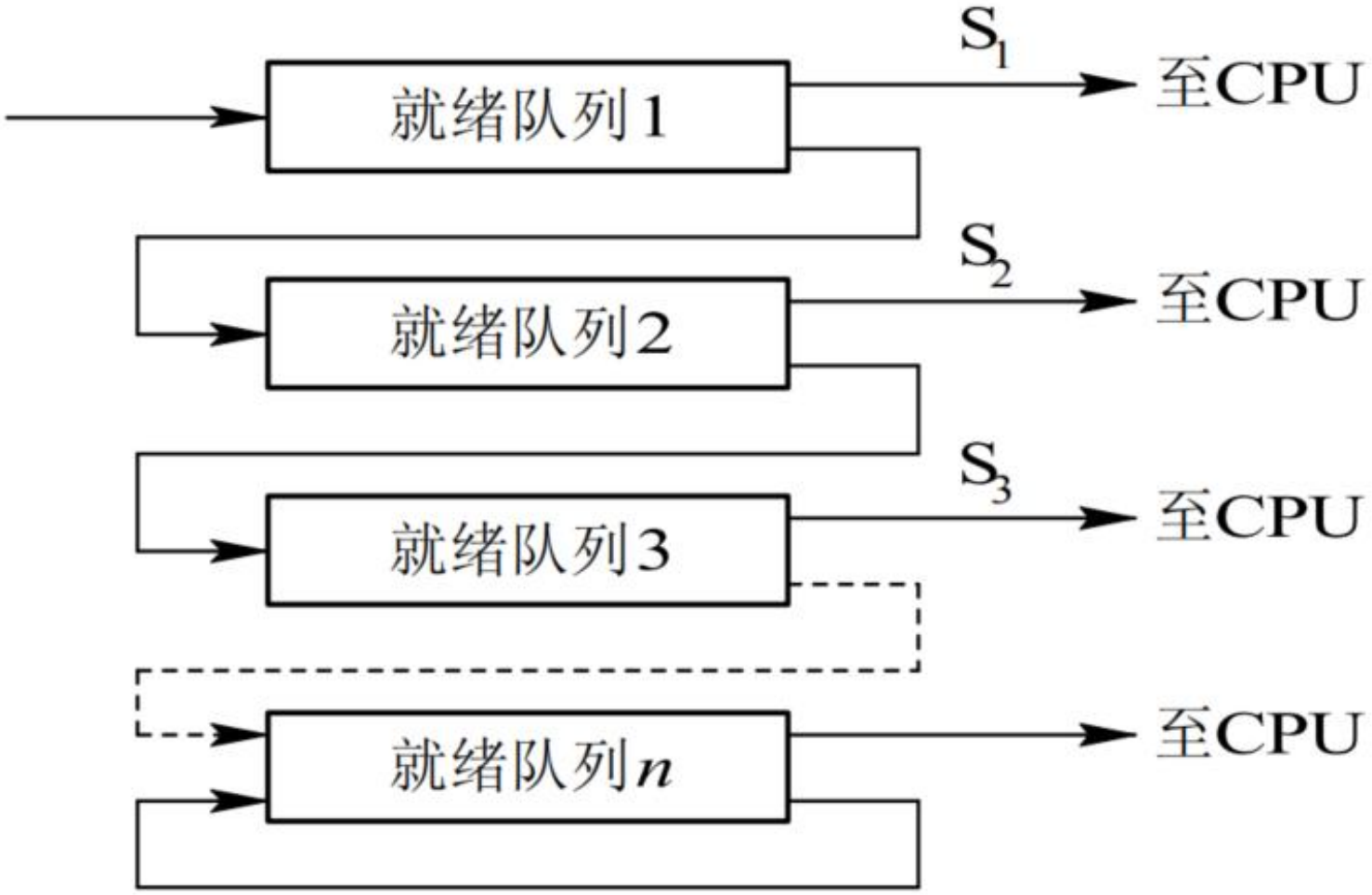


3.3.5 多级反馈队列调度算法

◆ 调度机制

- 不必事先知道各种进程所需的执行时间，可以满足各种类型进程的需要，目前被公认的一种较好的进程调度算法。
- 设置**多个就绪队列**，并为各个队列赋予不同的优先级，第一个队列的优先级最高，**优先权愈高的队列中，时间片愈小**
- 每个队列都**采用FCFS算法**
  - 当新进程进入内存后，首先将它放入第一队列的末尾，按FCFS原则等待调度。
  - 当轮到该进程执行时，如它能在该时间片内完成，便可撤离系统。否则，即它在一个时间片结束时尚未完成，调度程序将其转入第二队列的末尾等待调度，依此类推。
  - 当进程最后被降到第n队列后，在第n队列中便采取按RR方式运行。

3.3.5 多级反馈队列调度算法



（时间片：$S_1 < S_2 < S_3$）

3.3.5 多级反馈队列调度算法

◆ 调度机制

- 按队列优先级调度
  - 首先调度最高优先级队列中的诸进程运行，仅当第一队列空闲时才调度第二队列中的进程运行;
  - 优先级具有抢占式特点。

◆ 性能分析

- 如果规定第一个队列的时间片略大于多数人机交互所需之处理时间时，便能较好地满足各种类型用户的需要。
  - 终端型用户：可获得较好的交互性;
  - 短批处理作业用户：通常在前几个队列中可执行完毕;
  - 长批处理作业用户：依次在前面多个队列总运行，然后按照轮转方式运行，不必担心长期得不到处理。

3.3.6 基于公平原则的调度算法

◆ 保证调度算法

- 向用户所做出的保证并不是优先运行，而是明确的性能保证，该算法可以做到调度的公平性。
- 系统应具备以下功能:
  - 跟踪计算每个进程自创建以来已经执行的处理时间;
  - 计算每个进程应获得的处理机时间，即自创建以来的时间除以n;
  - 计算进程获得处理机时间的比率，即进程实际执行的处理时间和应获得的处理机时间之比;
  - 比较各进程获得处理机时间的比率，如进程A的比率最低，为0.5,而进程B的比率为0.8，进程C的比率为1.2等;调度程序应选择比率最小的进程将处理机分配给它，并让该进程一直运行， 直到超过最接近它的进程比率为止。

3.3.6 基于公平原则的调度算法

◆ 公平分享调度算法

- 调度的公平性主要针对用户而言，使所有用户能获得相同的处理机时间，或所要求的时间比例;
- 调度以进程为单位，需考虑每个用户所拥有的进程数目
  - 用户1启动A/B/C/D共计4个进程
  - 用户2启动E共计1个进程
  - 保证用户具有相同的处理机时间，则调度系列:

    A E B E C E D E A E B E C E D E ...
  - 保证用户1获得的处理机时间是用户2的两倍，调度系列:

    A B E C D E A B E C D E ...

处理机调度与死锁

- 处理机调度的层次和调度算法的目标
- 作业与作业调度 
- 进程调度
- 实时调度
- 死锁概述
- 预防死锁 
- 避免死锁
- 死锁的检测与解除

处理机调度与死锁

- 实现实时调度的基本条件
- 实时调度算法的分类

实时调度

- 最早截至时间优先算法
- 最低松弛度优先算法
- 优先级倒置

3.4.1 实现实时调度的基本条件

- **类型（均联系一个截至时间）**
  - HRT——硬实时
  - SRT——软实时
- **基本条件**
  - 提供必要信息
  - 系统处理能力强
  - 采用抢占式调度机制
  - 具有快速切换机制

3.4.1 实现实时调度的基本条件

◆ 系统向调度程序提供的必要信息

- **就绪时间**：该任务成为就绪状态的起始时间，在周期任务的情况下，它就是事先预知的一串时间序列；而在非周期任务的情况下，它也可能是预知的
- **开始截止时间**和**完成截止时间**
- **处理时间**：任务从开始执行直至完成所需的时间
- **资源要求**：任务执行时所需的一组资源
- **优先级**：如任务的开始截止时间错过会**引起故障**，则赋予**“绝对”优先级**；如果开始截止时间推迟对任务继续运行**无重大影响**，则赋予**“相对”优先级**

3.4.1 实现实时调度的基本条件

◆ 系统处理能力强

➢ 假定系统中有m个周期性的硬实时任务，它们的处理时间可表示为 Ci，周期时间表示为 Pi，则在单处理机情况下，可调度的限制条件为：

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

➢ 提高系统的处理能力

- 增强其处理能力，显著减少对每一个任务的处理时间
- 采用多处理机系统

3.4.1 实现实时调度的基本条件

◆采用抢占式调度机制

- 含有硬实时任务的实时系统中，广泛采用抢占机制。
- 小型实时系统，如果能预知任务开始截止时间，则采用非抢占调度机制，以简化调度程序和对任务调度花费的系统开销

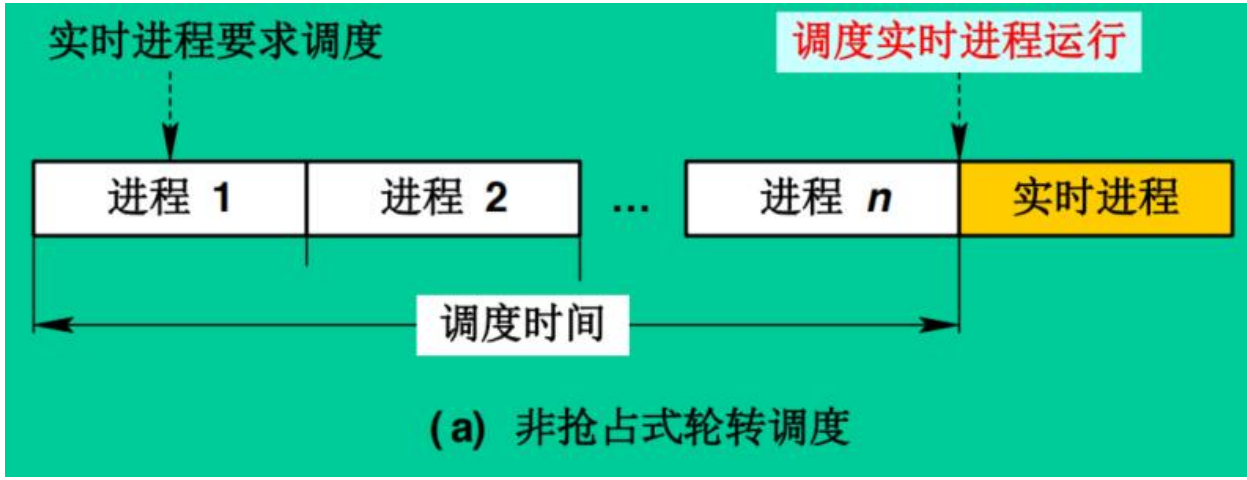
◆具有快速切换机制

- 对外部中断的快速响应能力，为使在紧迫的外部事件请求中断时系统能及时响应
- 快速的任务分派能力。在完成任务调度后，便应进行任务切换。为了提高分派程序进行任务切换时的速度，应使系统中的每个运行功能单位适当地小，以减少任务切换的时间开销

3.4.2 实时调度算法的分类

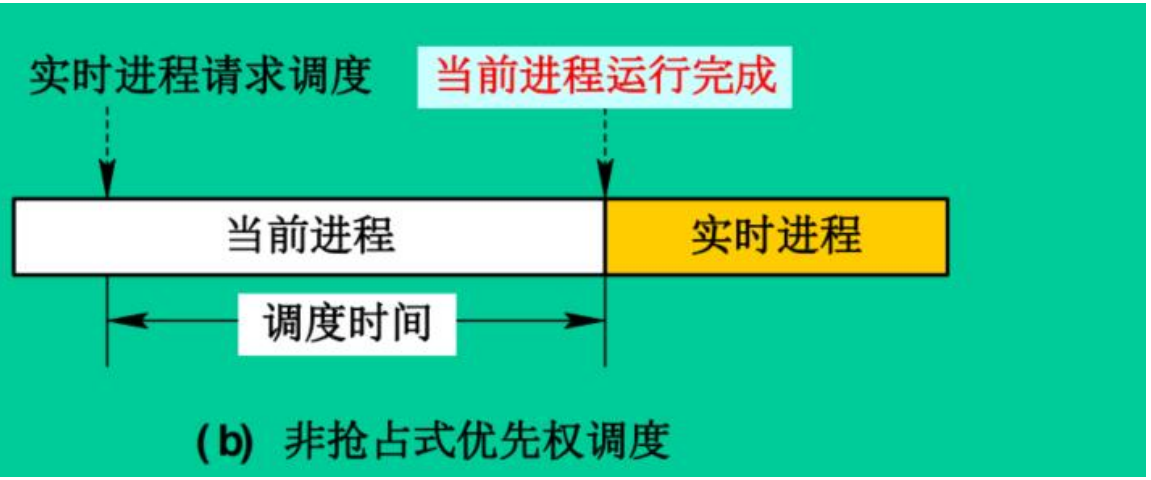
◆ 非抢占式调度算法

➢ 非抢占式轮转调度算法



(a) 非抢占式轮转调度

➢ 非抢占式优先调度算法

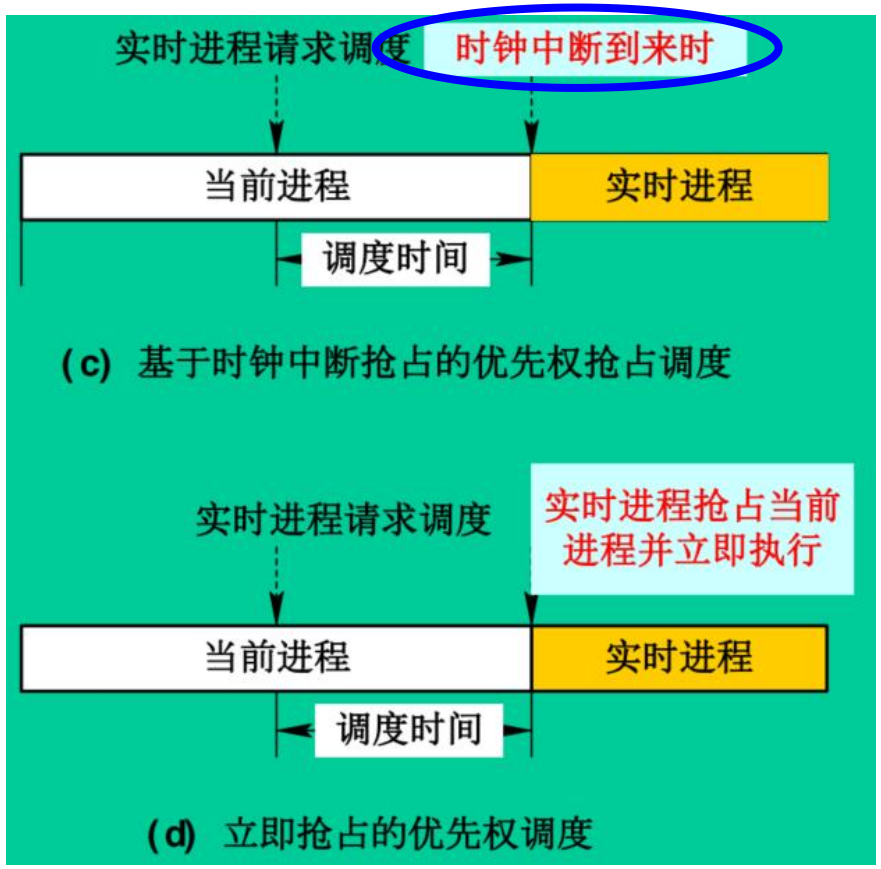


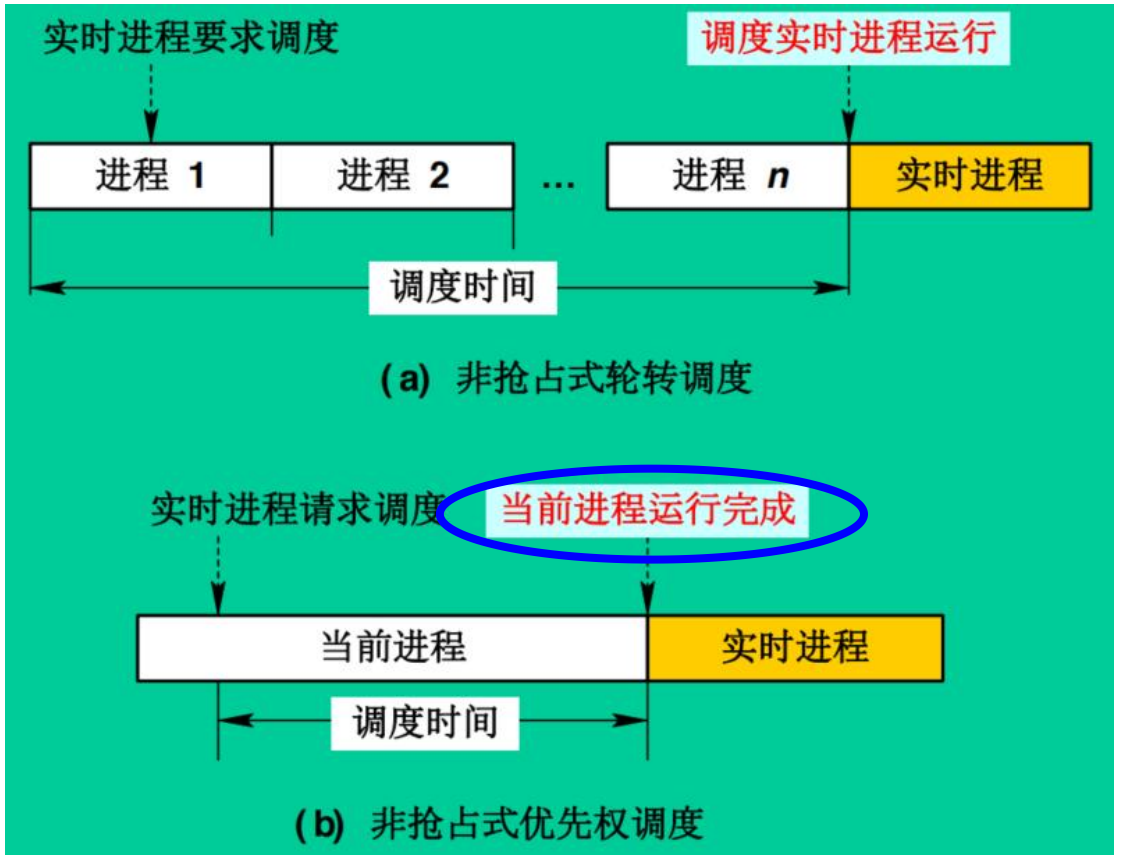
(b) 非抢占式优先权调度

3.4.2 实时调度算法的分类

◆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





3.4.3最早截至时间优先

◆最早截止时间优先 (Earliest Deadline First, EDF)

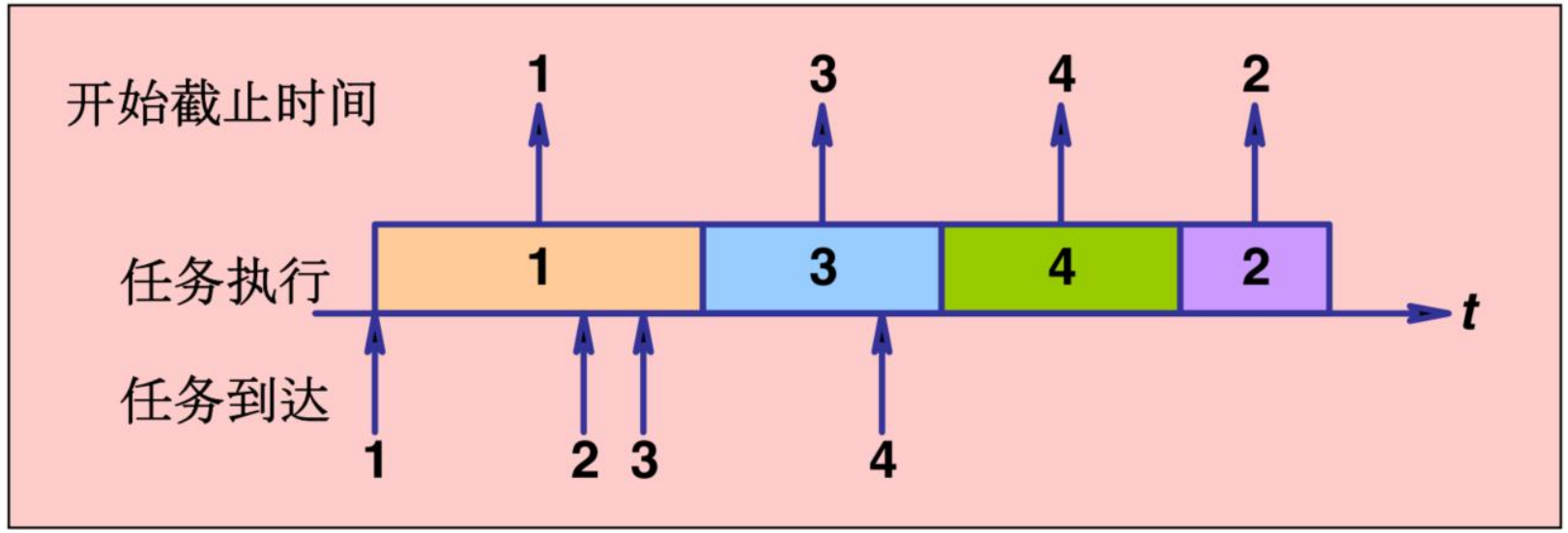
- 根据任务的**开始截止时间**来确定任务的优先级
- 截止时间愈早，其优先级愈高
- 可用于抢占式调度，也可用于非抢占式调度方式
  - 非抢占式调度方式用于非周期实时任务
  - 抢占式调度方式用于周期实时任务

3.4.3最早截至时间优先

◆ 非抢占式调度方式用于非周期实时任务

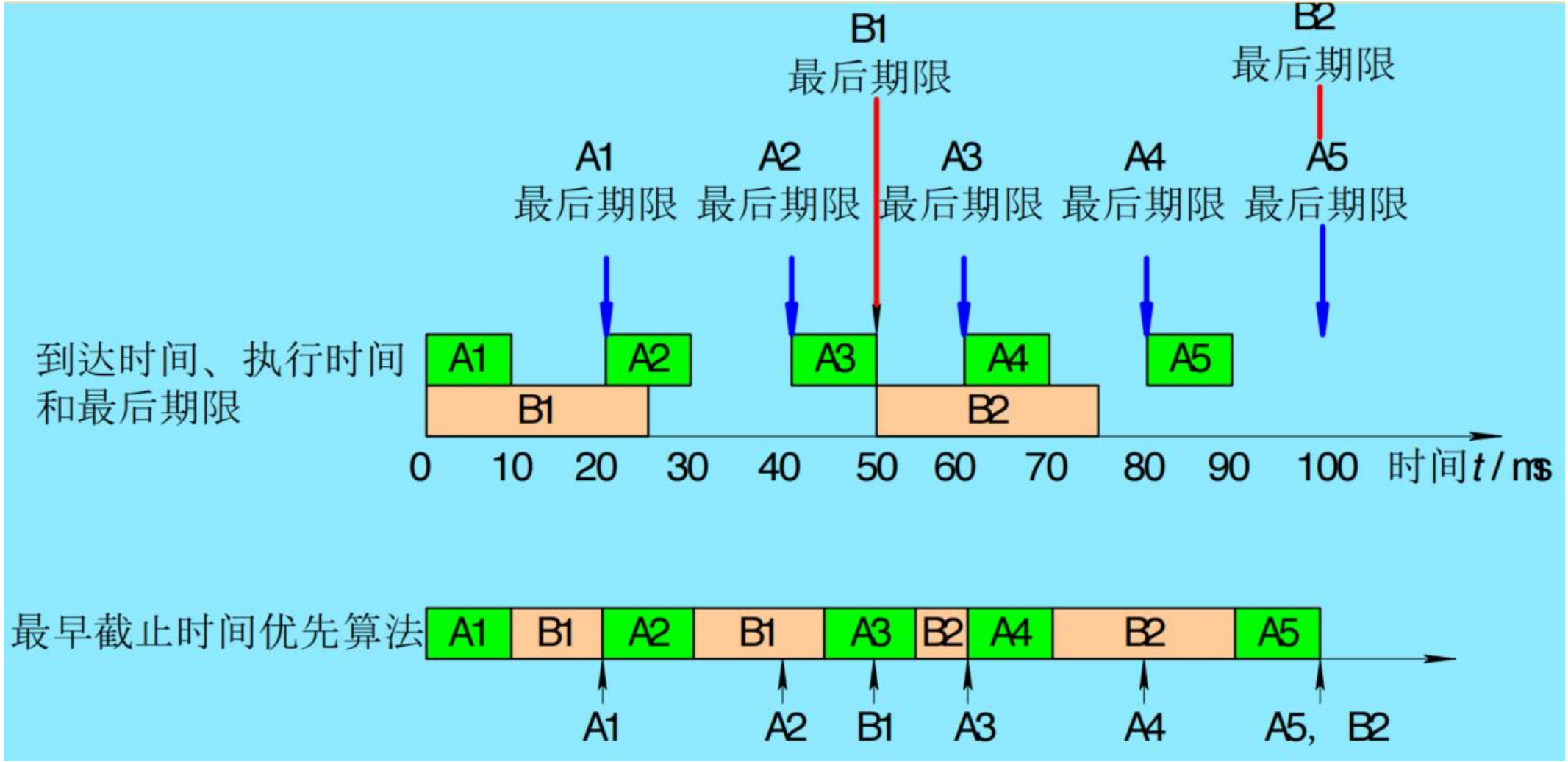
例1

四个非周期任务，它们先后到达。1）系统先调度任务1执行，在其执行期间，任务2、3又先后到达。2）由于任务3的开始截止时间早于任务2，故系统在任务1后将调度任务3执行。3）在此期间又到达作业4，其开始截止时间仍是早于任务2的，故在任务3执行完后，系统又调度任务4执行，最后才调度任务2执行。



3.4.3最早截至时间优先

- **抢占式调度方式用于周期实时任务**



3.4.4 最低松弛度优先

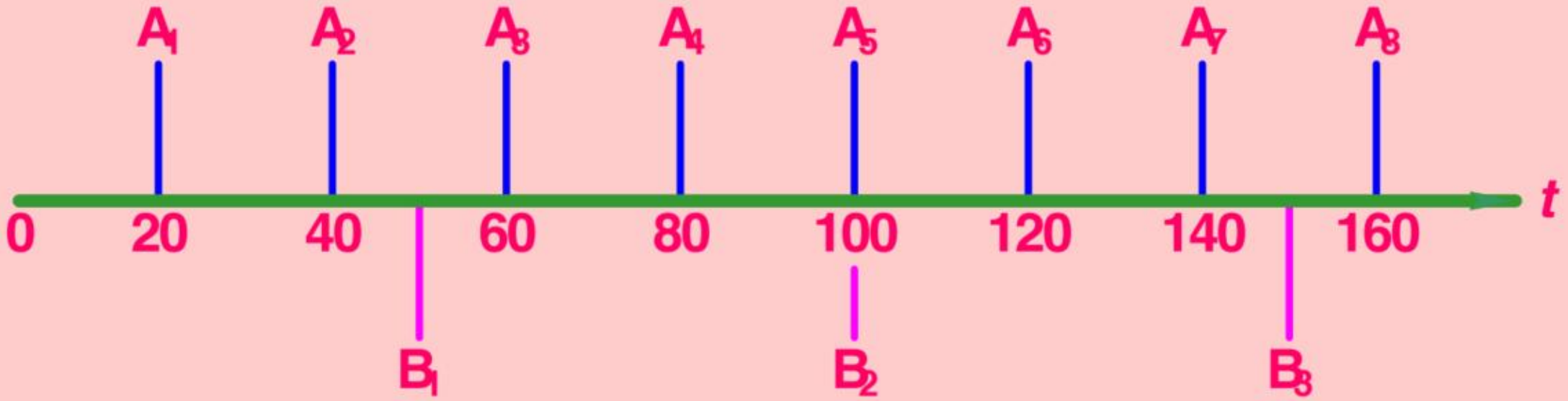
◆最低松弛度优先(Least Laxity First, LLF)算法

- 根据任务紧急(或松弛)的程度，来确定任务的优先级。
- 任务的紧急程度愈高，优先级就愈高，以使之优先执行
- 主要用于可抢占的调度方式中
- 系统中有一个按松弛度排序的实时任务就绪队列，松弛度最低的排在最前面，调度程序总是选择队首任务执行。
- 松弛程度 = 必须完成时间 - 其本身的运行时间 - 当前时间
- 一个任务在400ms时必须完成，它本身需要运行150ms，则其松弛程度为400-150=250ms

3.4.4 最低松弛度优先

◆最低松弛度优先(Least Laxity First, LLF)算法

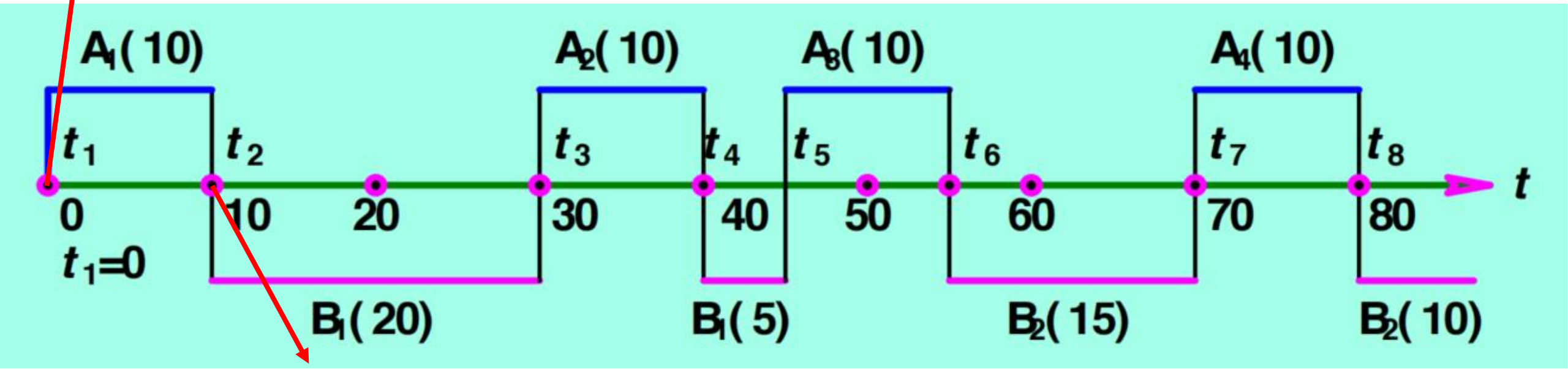
- 在一个实时系统中，两个周期性实时任务A和B，A要求每 20ms执行一次，执行时间为10ms；B只要求每 50ms执行一次，执行时间为25ms。由此可得知任务A和B每次必须完成的时间分别为：A1、A2、A3、…和 B1、B2、B3….



3.4.4 最低松弛度优先

◆最低松弛度优先(Least Laxity First, LLF)算法

- A1的松弛度为 20 – 10 = 10 ms；
- B1的松弛度为50 – 25 = 25 ms；



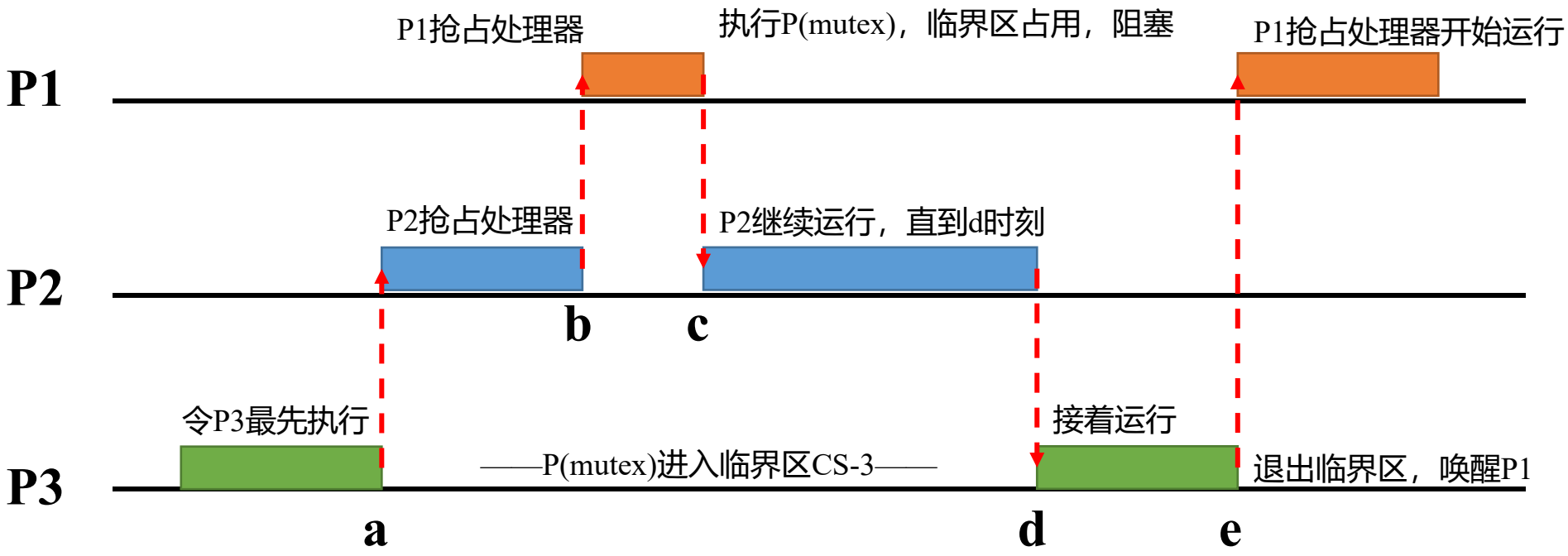
- A2的松弛度为40 – 10 – 10 = 20 ms ；
- B1的松弛度为50 – 25 – 10 = 15 ms；

3.4.5 优先级倒置Priority Inversion Problem

- **高优先级进程/线程被低优先级进程/线程延迟或阻塞**

优先级：P1 > P2 > P3，　P1和P3通过共享的临界资源进行交互

P1: …P(mutex); CS-1; V(mutex);…　　P2: …Program2;…　　P3: …P(mutex); CS-3; V(mutex);…

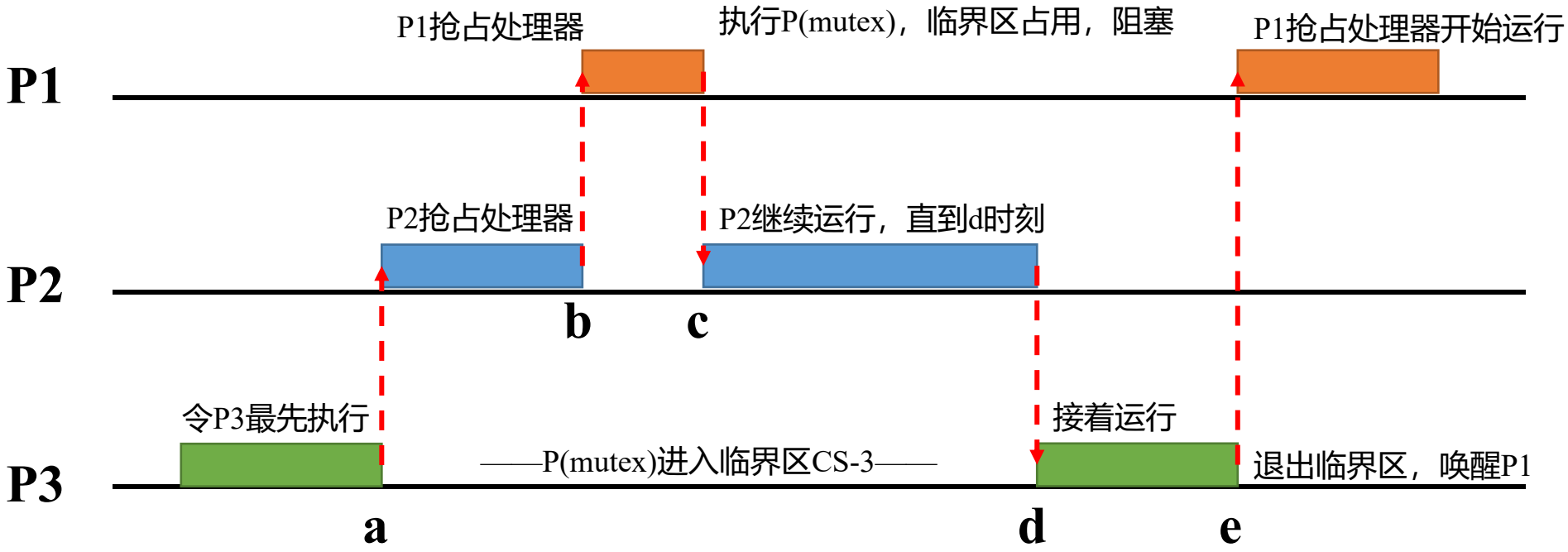


3.4.5 优先级倒置Priority Inversion Problem

- **高优先级进程/线程被低优先级进程/线程延迟或阻塞**

优先级：P1 > P2 > P3，　P1和P3通过共享的临界资源进行交互

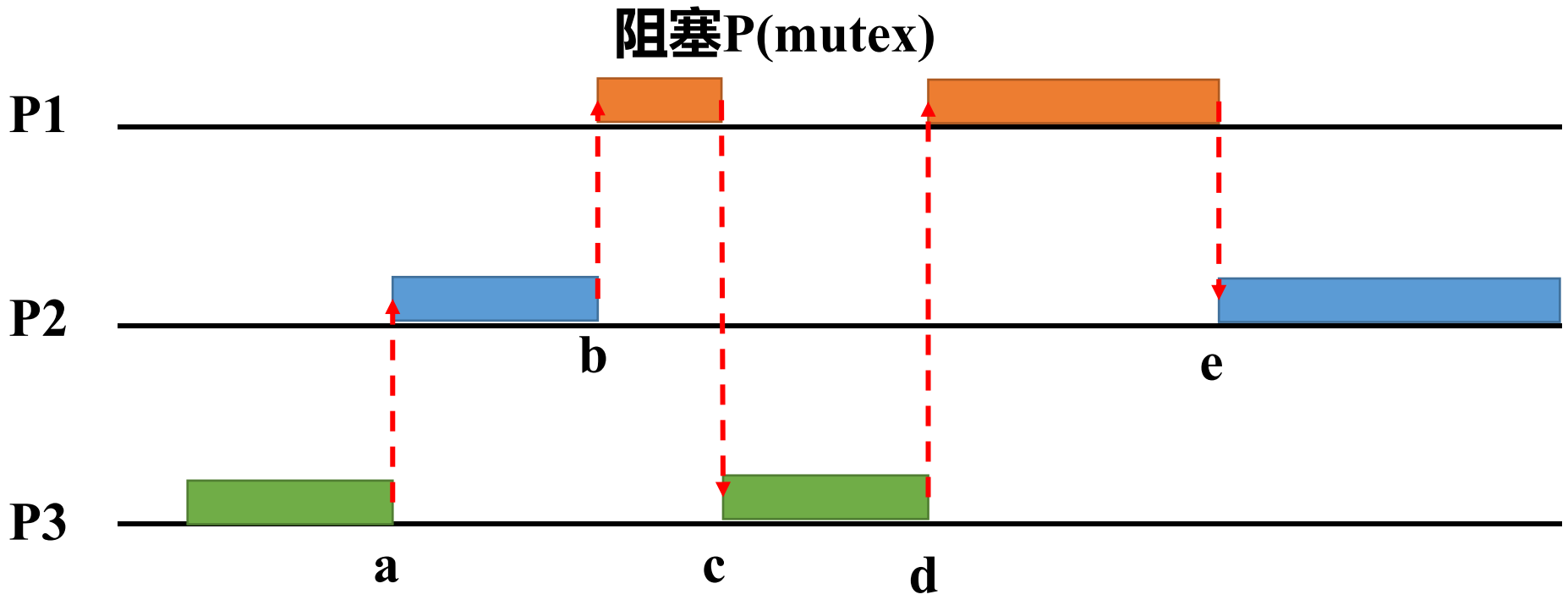
P1: …P(mutex); CS-1; V(mutex);…　P2: …Program2;…　P3: …P(mutex); CS-3; V(mutex);…



被延长的时间不可预知或无法限定，不应出现在实时系统中

3.4.5 优先级倒置Priority Inversion Problem

- 简单方案：低优先级进程进入临界区后不允许抢占处理机
- 实用方法：动态优先级继承，即P3继承P1优先级



处理机调度与死锁

- 处理机调度的层次和调度算法的目标
- 作业与作业调度 
- 进程调度
- 实时调度
- 死锁概述
- 预防死锁 
- 避免死锁
- 死锁的检测与解除


THEORY
PRACTICAL