第2章 处理器管理

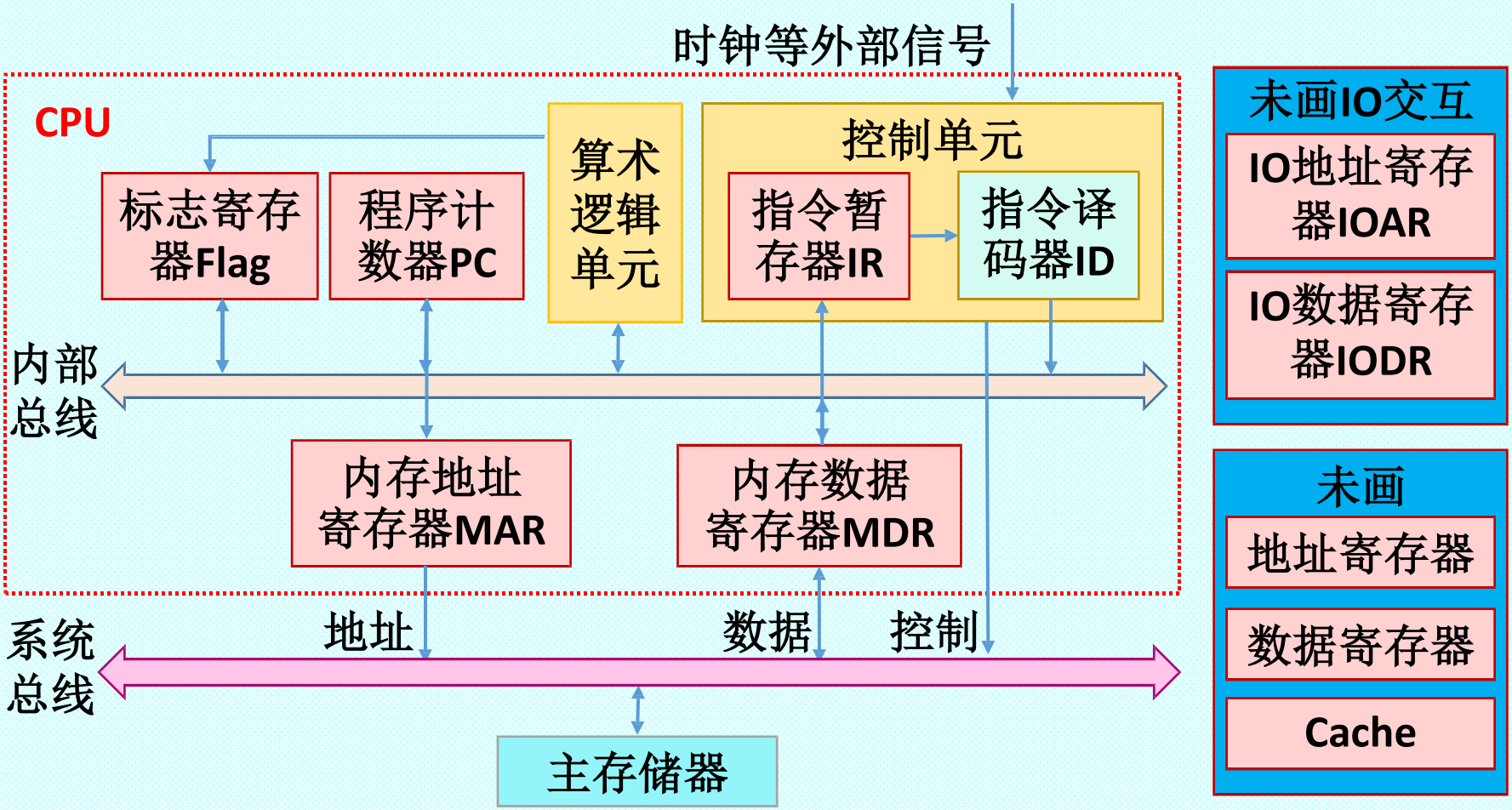
知识要点

- 掌握
  - 处理器：程序状态字，特权指令，处理器模式
  - 中断技术
  - 进程及其实现
  - 线程及其实现
  - 处理器调度
- 了解
  - Linux进程
  - Linux调度算法

2.1 处理器

- 处理器与寄存器
- 指令与处理器模式

2.1 处理器



处理器部件示意图

2.1.1 处理器与寄存器

- 计算机系统的处理器包括一组寄存器，其个数根据机型的不同而不同，它们构成了一级存储，比主存容量小，但访问速度快。
- 这组寄存器所存储的信息与程序的执行有很大关系，构成了处理器现场。

1. 用户可见寄存器

- 可以使程序减少访问主存储器的次数，提高指令执行的效率
- 所有程序可使用，包括应用程序和系统程序
  - 数据寄存器：又称通用寄存器，用于存放操作数；
  - 地址寄存器：用于存放地址，分为索引、栈指针、段地址等寄存器

2. 控制与状态寄存器

- 用于控制处理器的操作；主要被具有特权的操作系统程序使用，以控制程序的执行
- 程序计数器PC：存储将取指令的地址
- 指令寄存器IR：存储最近使用的指令
- 条件码CC：CPU为指令操作结果设置的位，标志正/负/零/溢出等结果
- 标志位：中断位、中断允许位、中断屏蔽位、处理器模 式位、内存保护位、…，等

2.1.2 指令与处理器模式

- 每台计算机机器指令的集合称指令系统，它反映了一台机器的功能和处理能力，可以分为以下五类：
  - 数据处理类指令：用于执行算术和逻辑运算。
  - I/O类指令：用于启动外围设备，让主存和设备交换数据。
  - 寄存器数据交换类指令：用于在处理器的寄存器和存储器之间交换数据。
  - 转移类指令：用于改变执行指令序列。
  - 处理器控制指令：修改处理器状态，改变处理器工作方式。

1.特权指令与非特权指令

- 在单道程序系统中，用户程序可以直接使用CPU指令启动I/O设备，进行I/O操作。
- 问题是：在多道程序系统中，这种模式可不可行?

1. 特权指令与非特权指令

- 从资源管理和控制程序执行的角度出发，把指令系统中的指令分作两部分：特权指令和非特权指令。
- 特权指令是指只能提供给操作系统的核心程序使用的指令，如启动I/O设备、设置时钟、控制中断屏蔽位、清主存、建立存储键，加载PSW等。

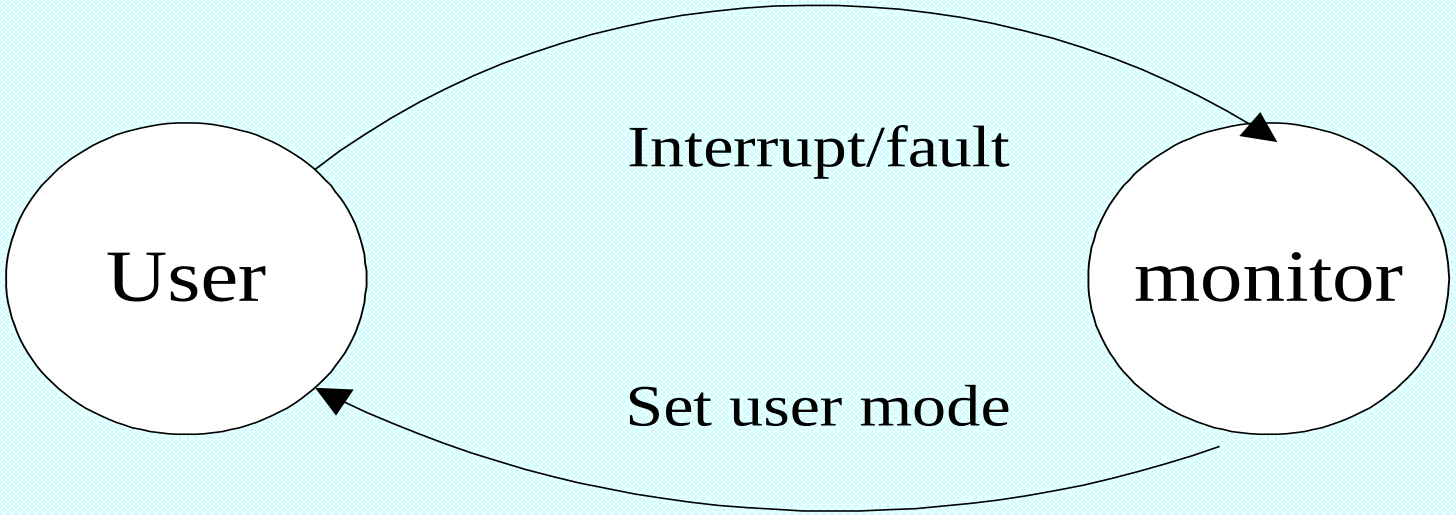
2. 内核态和用户态

- 处理器怎么知道当前是操作系统还是一般用户程序在运行呢?
- 处理器状态标志：管理状态（核心状态、特态或管态）和用户状态（目标状态、常态或目态)。
- 处理器处于管理状态时，程序可以执行全部指令，使用所有资源；处于用户状态时只能执行非特权指令。

3. 处理器状态及其转换

- 下列情况导致处理器从用户态向内核态转换
  - 程序请求操作系统服务，执行系统调用；
  - 程序运行时，产生中断或异常事件，运行程序被中断，转向中断处理或异常处理程序工作。
- 两类情况都通过中断机制发生，中断和异常是用户态到内核态转换仅有的途径。
- 从内核态转向用户态，计算机提供一条特权指令称作加载程序状态字(Intel x86为iret指令)，用来实现从系统(核心态)返回到用户态，控制权交给应用进程。

3. 处理器状态及其转换



管态和目态的切换

4. 用户栈和核心栈

- 用户栈：用户进程空间中开辟的区域，用于保存应用程序的子程序（函数）间相互调用的参数、返回值、返回点以及子程序的局部变量。
- 核心栈：操作系统空间的一块区域，用于保存中断现场和操作系统程序（函数）间相互调用的参数、返回值、返回点以及程序局部变量。
- 栈指针：为硬件栈指针，用户栈和核心栈共用一个栈指针。

5. 程序状态字

- 计算机如何知道当前处于何种工作状态？这时能否执行特权指令？通常操作系统都引入程序状态字PSW（Program Status Word）来区别不同的处理器工作状态。
- PSW用来控制指令执行顺序并保留和指示与程序有关的系统状态，主要作用是实现程序状态的保护和恢复。
- 每个程序都有一个与其执行相关的PSW，每个处理器都设置一个PSW寄存器。程序占有处理器执行，它的PSW将占有PSW寄存器。

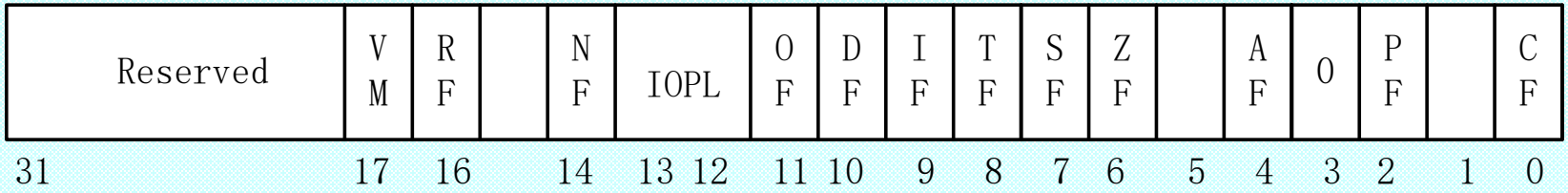
5. 程序状态字PSW

- PSW既是操作系统的概念，用于记录当前程序运行的动态信息：
  - 程序基本状态：程序计数器，指令寄存器，条件码
  - 中断字：保存程序执行时当前发生的中断事件。
  - 中断屏蔽位：指明程序执行中发生中断事件时，是否响应出现的中断事件
- PSW也是计算机系统的寄存器
  - 通常设置一组控制与状态寄存器
  - 也可以专设一个PSW寄存器

Intel x86程序状态字

- Intel x86中，PSW由标志寄存器EFLAGS和指令指针寄存器EIP组成，均为32位。
- EFLAGS的低16位称FLAGS，标志可划分为三组：状态标志、控制标志、系统标志。

微处理器Intel 80386的程序状态字



其中： IF：中断允许标记

IOPL：I/O特权级，用来表示I/O操作所处特权级。

00：0级　　01：1级

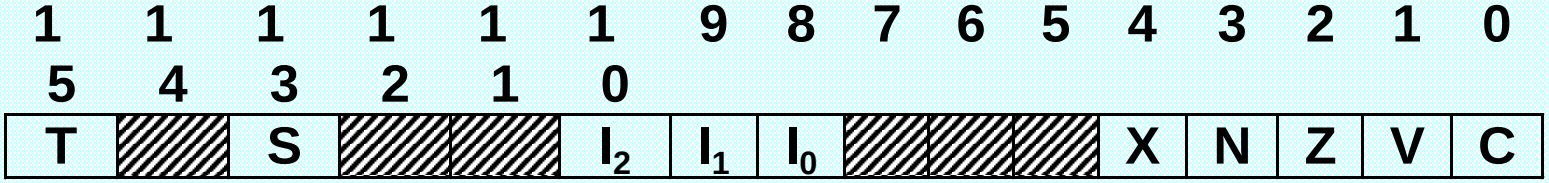
10：2级　　11：3级

0~2级：管态　　3级：目态

Pentium的处理器状态

- Pentium的处理器状态有四种，支持4个保护级别，0级权限最高，3级权限最低。一种典型的应用是把4个保护级别依次设定为：
  - 0级为操作系统内核级。处理I/O、存储管理、和其他关键操作。
  - 1级为系统调用处理程序级。用户程序可以通过调用这里的过程执行系统调用，但是只有一些特定的和受保护的过程可以被调用。
  - 2级为共享库过程级。它可以被很多正在运行的程序共享，用户程序可以调用这些过程，读取它们的数据，但是不能修改它们。
  - 3级为用户程序级。它受到的保护最少。
- 各个操作系统在实现过程中可以根据具体策略有选择地使用硬件提供的保护级别，如运行在Pentium上的Windows操作系统只使用了0级和3级。

微处理器M68000的程序状态字



条件位：

C： 进位标志位　　**V：** 溢出标志位

Z： 结果为零标志位　　**N：** 结果为负标志位

I0 – I2：三位中断屏蔽位

S：CPU状态标志位，为1处于管态，为0处于目态

T：陷阱（**Trap**）中断指示位为**1**，

在下一条指令执行后引起自陷中断

2.2中断技术

- 中断概念
- 中断源分类
- 中断和异常的响应及服务
- 中断事件处理原则
- 中断优先级和多重中断
- Linux中断处理（自学）

2.2.1 中断概念

- 系统处理以下情况时，需要打断处理器正常工作：
  - 请求系统服务，
  - 实现并行工作，
  - 处理突发事件，
  - 满足实时要求，
- 为此，提出了中断概念。

中断定义

- 中断是指程序执行过程中，遇到急需处理的事件时，暂时中止CPU上现行程序的运行，转去执行相应的事件处理程序，待处理完成后再返回原程序被中断处或调度其他程序执行的过程。
- 操作系统是“中断驱动”的；即中断是激活操作系统的唯一方式
- 中断有广义和狭义之分，上述中断是指广义的中断

中断系统的概念

- 中断系统是实现中断功能的部件，包括中断装置和中断处理程序。
- 中断装置：指发现中断，响应中断的硬件。
  - 发现中断源，提出中断请求。
  - 保护现场
  - 启动处理中断事件的程序 。
- 中断处理程序：由软件来完成。
  - 主要任务是处理中断事件和恢复正常操作 。

2.2.2中断源分类

- 外中断(中断或异步中断)
  - 指来自处理器之外的中断信号，包括时钟中断、键盘中断和设备中断等；
  - 又分可屏蔽中断和不可屏蔽中断，每个不同中断具有不同的中断优先级，表示事件的紧急程度，在处理高一级中断时，往往会屏蔽部分或全部低级中断。
- 内中断(异常或同步中断)
  - 指来自处理器内部，通常由于程序执行中，发现与当前指令关联的、不正常的、或是错误的事件。

中断和异常的区别(1)

- (1)中断：由与现行指令无关的中断信号触发的(异步的)，且中断的发生与CPU处在用户模式或内核模式无关，在两条机器指令之间才可响应中断，一般来说，中断处理程序提供的服务不是为当前进程所需的；
- 异常：由处理器正在执行现行指令而引起的，一条指令执行期间允许响应异常，异常处理程序提供的服务是为当前进程所用的。异常包括很多方面（有出错(fault)，也有陷入(trap)等）。

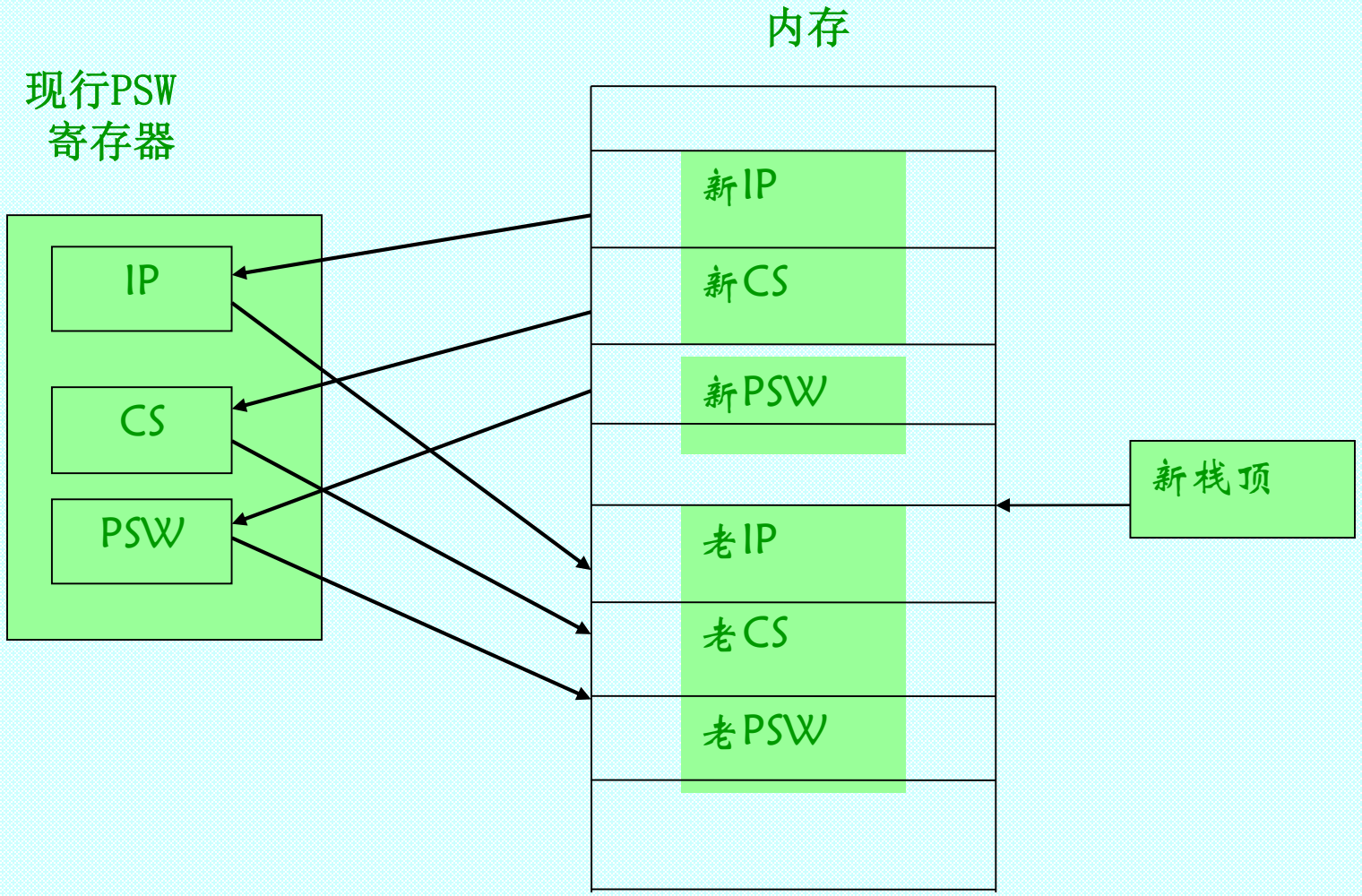
中断和异常的区别(2)

- （2）要求“中断”被快速处理，以便及时响应其它中断信号，所以，中断处理程序处理过程中是不能阻塞的。“异常”处于被打断的当前进程上下文中，所提供的服务是当前进程所需要的，所以，异常处理程序处理过程中是可以阻塞的。
- （3）中断允许发生嵌套，但异常大多为一重；异常处理过程中可能会产生中断，但中断处理过程中决不会被异常打断。

2.2.3中断和异常的响应及服务

- 发现中断源；
- 保护现场；
- 转向处理中断/异常事件的处理程序；
- 恢复现场。

IBM PC机中断响应过程



2.2.4 时钟中断(1)

- 时钟：操作系统进行调度工作的重要工具：
  - 让分时进程作时间片轮转
  - 让实时进程定时发出或接收控制信号
  - 系统定时唤醒或阻塞一个进程
  - 对用户进程进行记账。
- 时钟可分成绝对时钟和间隔时钟两种。

2.2.4时钟中断(2)

- Linux系统运行不同的间隔定时器，类型有三种：
  - real间隔定时器：按实际经过时间计时，不管进程处在何种模式下运行，包括进程被挂起时，计时总在进行，定时到达时发送给进程一个SIGALRM信号。
  - virtual间隔定时器：进程在用户态下执行时才计时，定时到达时发送给进程一个SIGVTALRM信号。
  - profile间隔定时器：进程执行在用户态或核心态时都计时，当定时到达时发送给进程一个SIGROF信号。

2.2.5 中断优先级和多重中断

- 1. 中断优先级
- 2. 中断屏蔽
- 3. 多重中断事件的处理

1. 中断优先级

- 计算机执行的每一瞬间，可能有几个中断事件同时发生，中断装置如何来响应同时发生的中断呢?
- 以不发生中断丢失为前提，把紧迫程度相当的中断源归在同一级，紧迫程度差别大的中断源归在不同级，
- 级别高的有优先获得响应的权力，中断装置所预设的中断响应顺序称为中断优先级。

2. 中断屏蔽

- 在CPU上运行的程序，有时由于种种原因，不希望其在执行过程中被别的事件所中断。
- 可编程中断控制器，可通过指令设置屏蔽码。中断屏蔽是指禁止CPU响应中断或禁止中断产生。
- 前者指硬件产生中断请求后，CPU暂时不予响应的状态。后者指可引起中断的事件发生时，硬件不允许提出中断请求也不通知处理器，故不可能导致中断。

中断屏蔽的作用

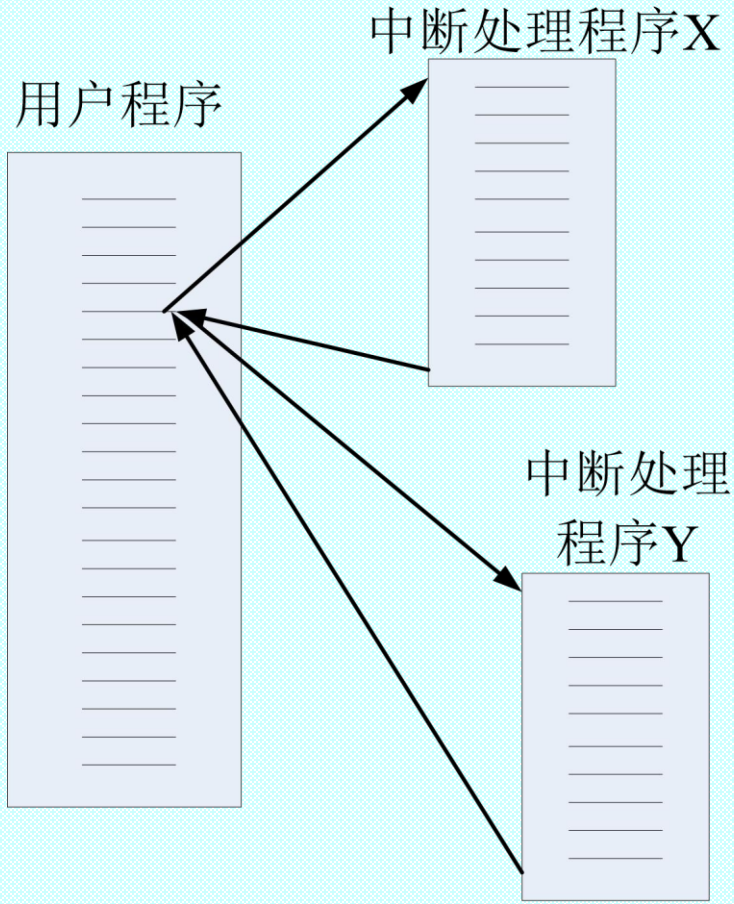
- 延迟或禁止某些中断的响应。系统程序执行过程中，不希望产生干扰事件，以免共享数据结构受到破坏。程序运行过程中产生某些事件认为是正常的，不必加以处理。
- 协调中断响应与中断处理的关系。确保高优先级中断可以打断低优先级中断，反之却不能。
- 防止同级中断相互干扰。在处理某优先级中断事件时，必须屏蔽该级中断，以免造成混乱。

3.多重中断事件的处理

- 中断正在进行处理期间，CPU又响应新的中断事件，于是暂时停止正在运行的中断处理程序，转去执行新的中断处理程序，就叫多重中断（又称中断嵌套）。
- 对同一优先级的不同中断：采用顺序处理方法。
- 对不同优先级的中断，主要采用以下处理方法：
  - (1) 串行处理：根据需要设置中断屏蔽
  - (2) 嵌套处理：发生某些优先级更高的中断，嵌套一般不超过3层。

3.多重中断事件的处理

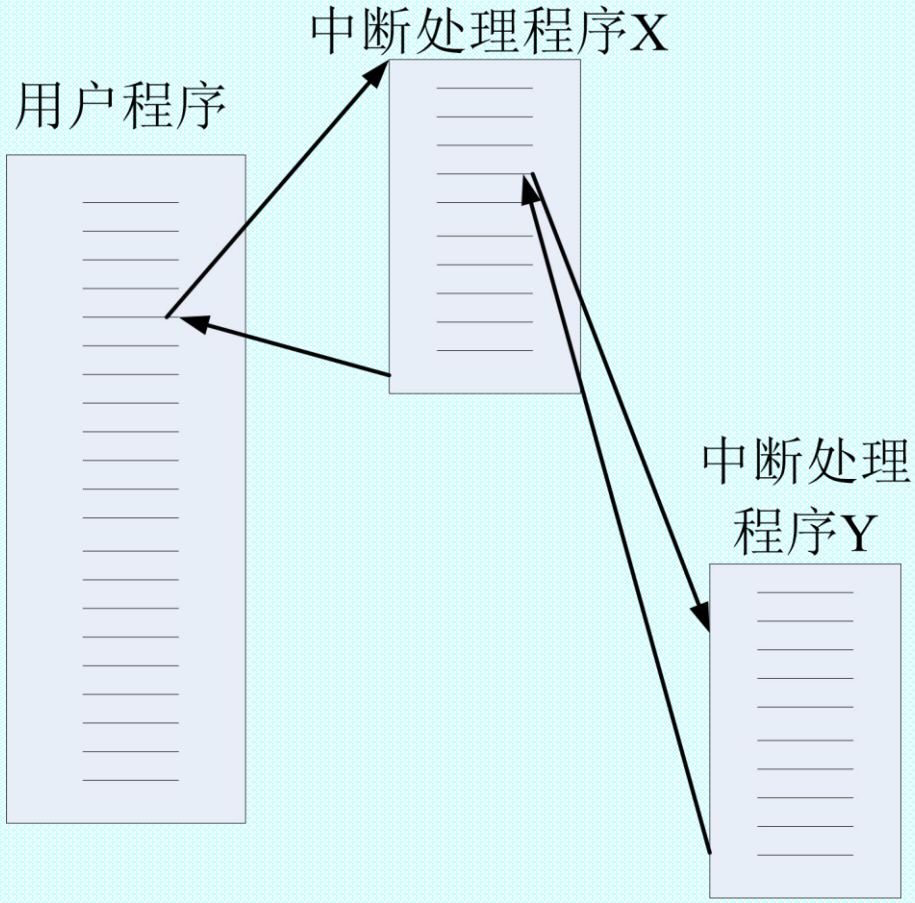
- X、Y两个中断同时发生
- 先响应X
- 因Y被屏蔽，继续处理X
- 再响应并处理Y



(a) 顺序中断处理

3.多重中断事件的处理

- X、Y两个中断同时发生
- 根据中断优先级，先响应X
- 因未屏蔽Y，再响应并处理Y
- Y处理完成后，再处理X



(b) 嵌套中断处理

4.中断向量(1)

- Intel x86机器支持256种中断信号，从0到255编号组成“中断向量”。
- 中断信号源分2类：中断和异常。中断分为屏蔽中断和非屏蔽中断；异常分为故障（fault）、陷阱（trap）、终止（abort）和编程异常（programmed exception）。

Linux中断向量(2)

- 非屏蔽中断和异常的向量是固定的，而屏蔽中断的向量可通过对中断控制器编程加以改变。
- 256个中断向量分配如下：
  - 1)0～31的向量对应于异常和非屏蔽中断；
  - 2)32～47的向量对应于屏蔽中断，被外部设备使用；
  - 3)48～255的向量分配给软中断。

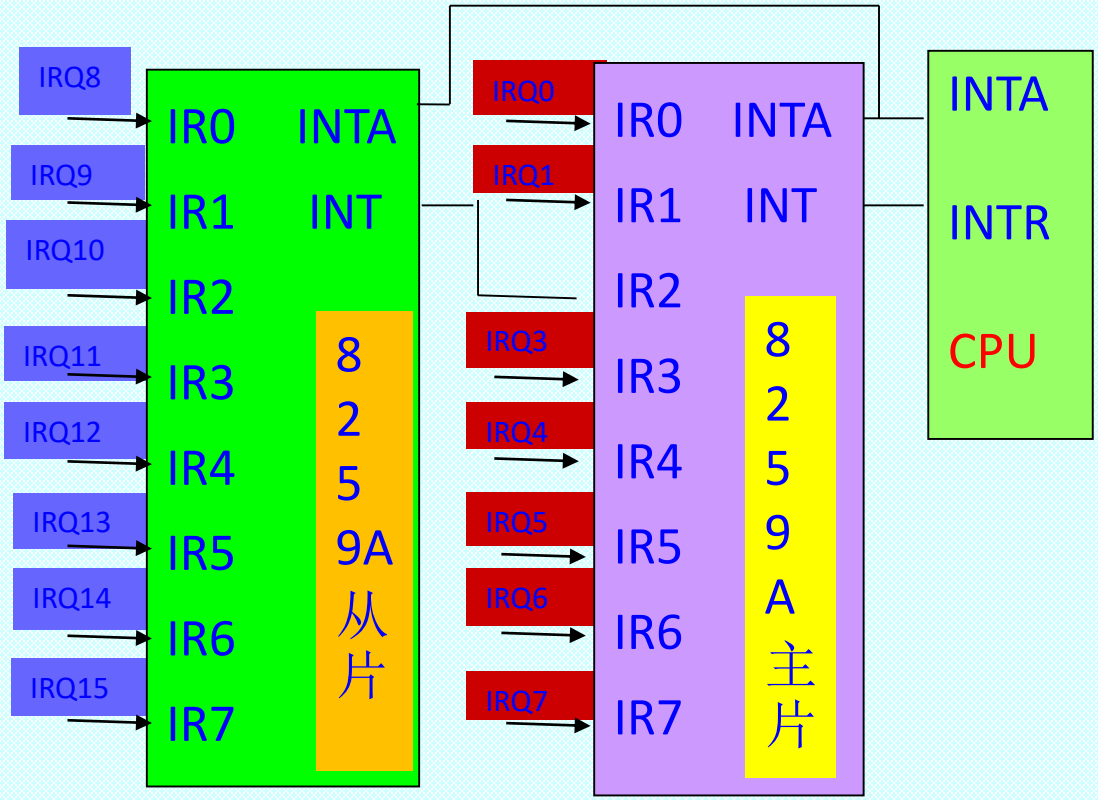
Intel x86处理器 中断向量表

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

中断请求

- 每个能发送中断信号的硬件设备控制器都有一根输出线，它与中断控制器8259A的输入引脚相连，若一个硬件设备欲向CPU发送中断信号，必须申请一条可用的“中断请求线”，或者说必须申请一个IRQ号，这就是“中断请求”IRQ（Interrupt Requirement）。
- IRQ号与中断向量号的对应关系
  - 由BIOS初始化
  - 一般IRQ0-IRQ7对应中断向量号0x8-0xF。

8259A中断控制器


IRQ8
IRQ9
IRQ10
IRQ11
IRQ12
IRQ13
IRQ14
IRQ15
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
INTA
INT
8259A从片
IRQ0
IRQ1
IRQ3
IRQ4
IRQ5
IRQ6
IRQ7
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
INTA
INT
8259A主片
INTA
INTR
CPU

5. 中断处理过程

- 简单的中断处理 - 典型的处理过程:
  - （1）设备给处理器发一个中断信号
  - （2）处理器处理完当前指令后响应中断，延迟非常短（要求处理器没有关闭中断）
  - （3）处理器处理完当前指令后检测到中断，判断出中断来源并向发送中断的设备发送了确认中断信号，确认信号使得该设备将中断信号恢复到一般状态
  - （4）处理器开始为软件处理中断做准备：保存中断点的程序执行上下文环境，这通常包括程 序状态字PSW，程序计数器PC中的下一条指令位置，一些寄存器的值，它们通常保存在系统控制栈中，处理器状态被切换到管态。

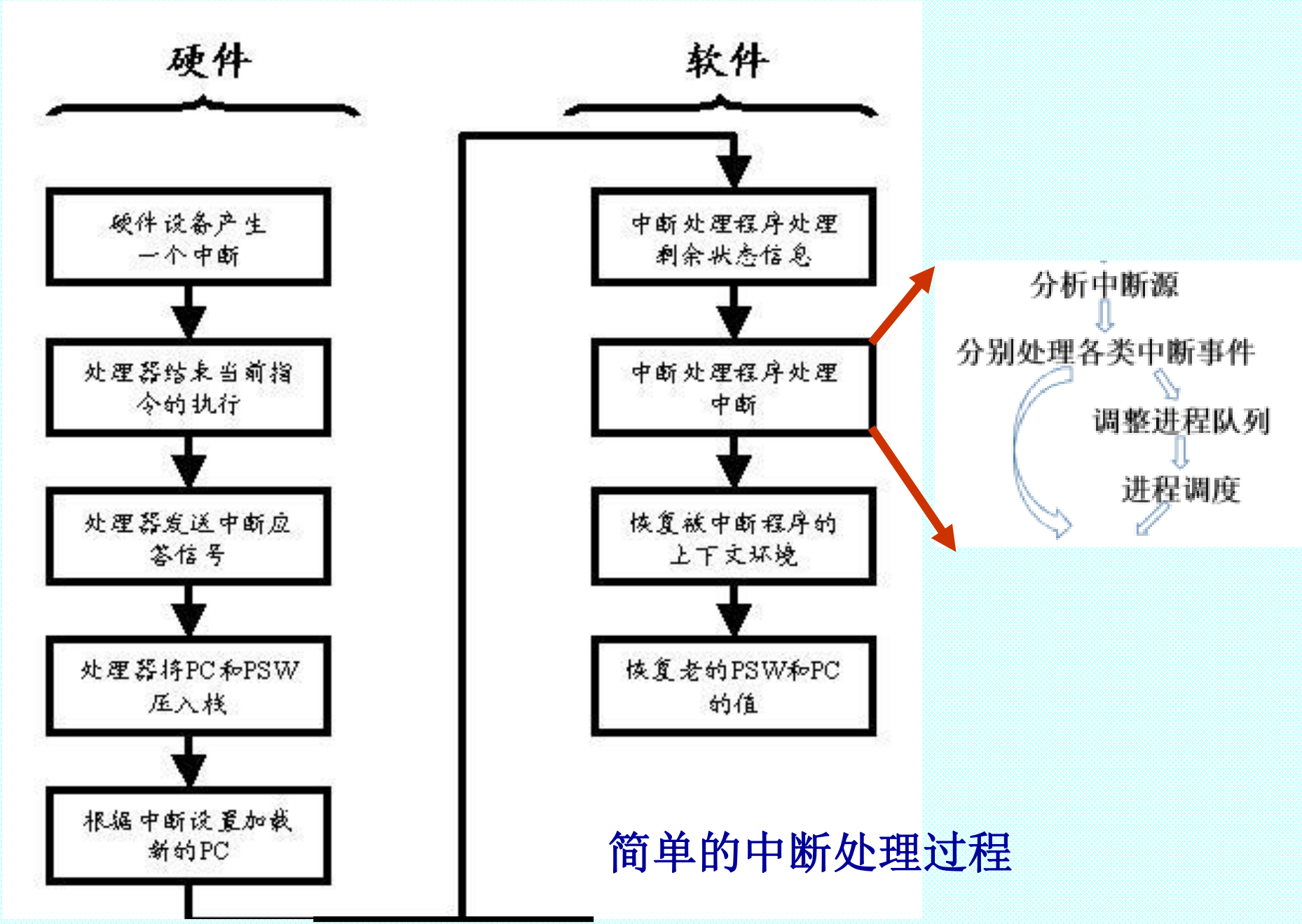
4. 中断处理过程

(5) 处理器根据中断源查询中断向量表，获得与该中断相联系的处理程序入口地址，并将PC置成该地址，处理器开始一个新的指令周期，控制转移到中断处理程序

(6) 中断处理程序开始工作，包括检查I/O相关的状态信息，操纵I/O设备或者在设备和主存之间传送数据等等

(7) 中断处理结束时，处理器检测到中断返回指令，被中断程序的上下文环境从系统堆栈中被恢复处理器状态恢复成原来的状态。

(8) PSW和PC被恢复成中断前的值，处理器开始一个新的指令周期，中断处理结束。


硬件
软件
硬件设备产生一个中断
处理器结束当前指令的执行
处理器发送中断应答信号
处理器将PC和PSW压入栈
根据中断设置加载新的PC
中断处理程序处理剩余状态信息
中断处理程序处理中断
恢复被中断程序的上下文环境
恢复老的PSW和PC的值
分析中断源
分别处理各类中断事件
调整进程队列
进程调度

简单的中断处理过程

6. 中断处理程序特点

- 以异步方式运行，可能会打断关键代码的执行，甚至打断其他中断处理程序的执行；
- 在屏蔽中断状态下运行，最坏的情况会禁止所有中断；
- 要对硬件进行操作，有很高的时限要求；它在中断上下文中运行，故不能被阻塞 。

2.3 进程及其实现

- 进程定义和属性
- 进程状态和转换
- 进程描述和组成
- 进程上下文切换与处理器状态转换
- 进程控制和管理

2.3.1 进程的定义和性质

- 进程定义：
  - 进程是可并发执行的程序在某个数据集合上的一次计算活动，也是操作系统进行资源分配和保护的基本单位。
- 进程是一个既能用来共享资源，又能描述程序并发执行过程的系统基本单位。
- 进程是一种支持程序执行的系统机制。

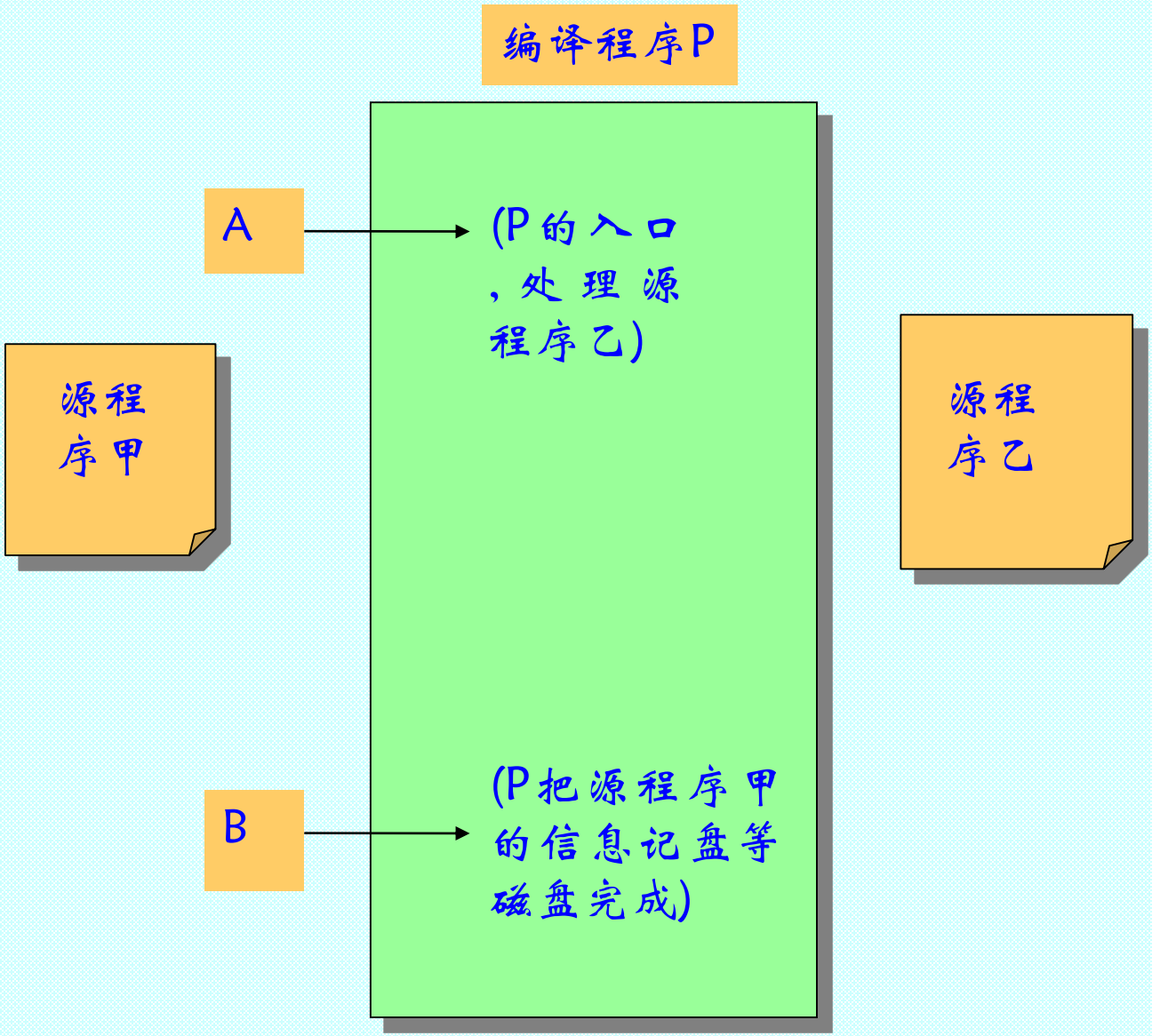
进程概念的引入(1)

- 原因1：刻画程序的并发性。
  - 程序是并发执行的，即不是连续而是走走停停的。程序的并发执行引起资源共享和竞争问题，执行的程序不再处在封闭环境中。
  - “程序”自身只是计算任务的指令和数据的描述，是静态概念无法刻画程序的并发特性，系统需要寻找一个能描述程序动态执行过程的概念，这就是进程。

进程概念的引入2)

- 原因2：解决资源的共享性。
  - “可再用” 程序
    - 指在调用它的程序退出之前不允许其他程序调用的程序。
    - 被调用过程中可以有自身修改。
  - “可再入” 程序
    - 指能够被多个程序同时调用的程序。
    - 纯代码，在执行过程中不被修改。
  - 程序的可再入性使程序与程序的执行不再一一对应。程序概念不能描述这种共享。

“可再入”程序举例



进程的属性

- 动态性：进程是程序在数据集合上的一次执行过程，是动态概念；而程序是一组有序指令序列，是静态概念。
  - 进程有一个生命过程：创建、运行、等待等。
  - 进程具有动态的地址空间（数量和内容）。
- 共享性：同一程序同时运行于不同数据集合上时，构成不同的进程。
- 独立性：是系统中资源分配和保护的基本单位，也是系统调度的独立单位(单线程进程)。每个进程的地址空间相互独立。
- 制约性：并发进程之间存在着制约性，在进行的关键点上需要相互等待或互通消息。
- 并发性：各进行按各自独立的，不可预知的速度并发推进。并发和异步特性会导致程序执行的不可再现性。

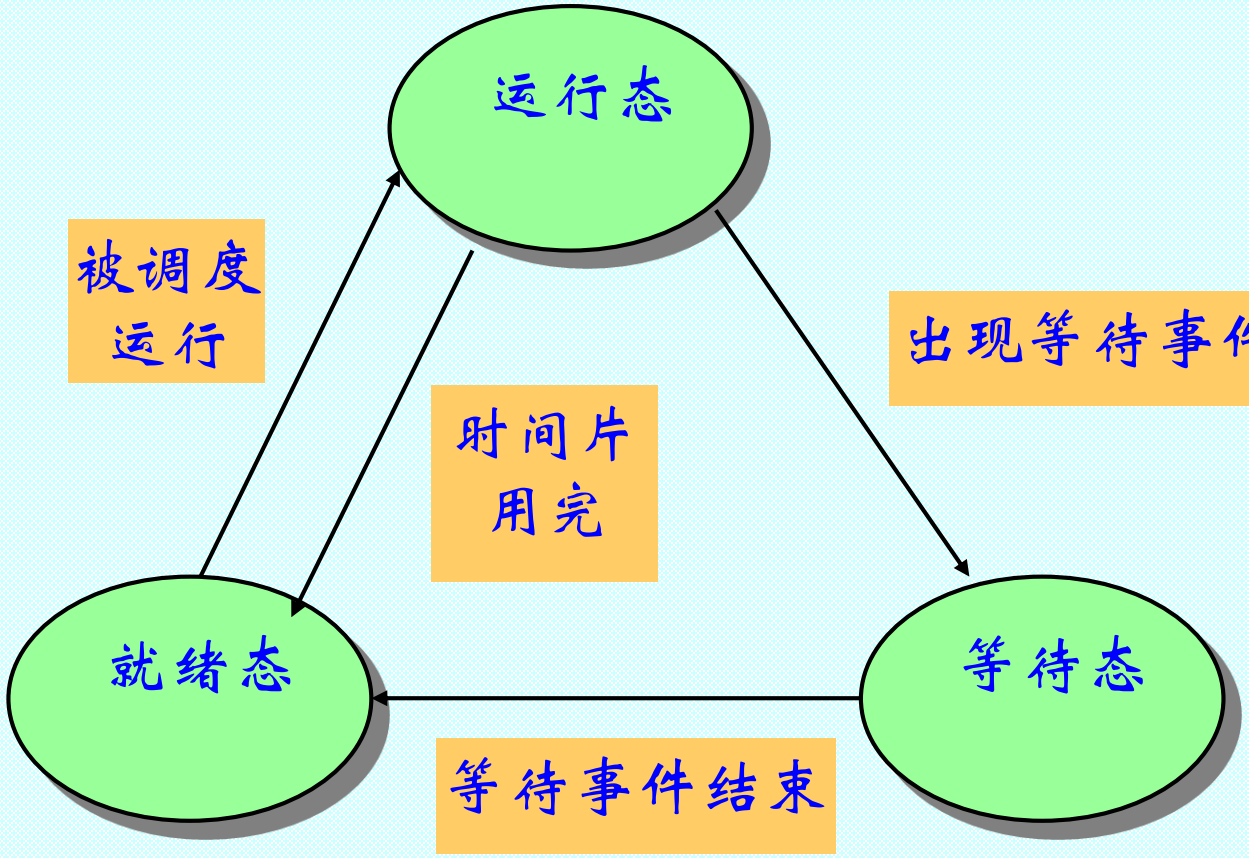
2.3.2 进程状态和转换

进程三态模型及其状态转换

- 运行状态（Running）：进程占有CPU，并在CPU上运行。处于此状态的进程的数目小于等于CPU的数目。
  - 在没有其他进程可以执行时（如所有进程都在阻塞状态），通常会自动执行系统的idle进程（相当于空操作）。
- 就绪状态（Ready）：进程已获得除处理机外的所需资源，等待分配处理机资源；只要分配CPU就可执行。
  - 可以按多个优先级来划分队列，如：时间片用完—>低优，I/O完成—>中优，页面调入完成—>高优
- 等待状态（Blocked）：指进程因等待某种事件的发生而暂时不能运行的状态（即使CPU空闲，该进程也不可运行）。等待的事件可以为：I/O操作或进程同步等。

2.3.2 进程状态和转换

进程三态模型及其状态转换



进程转换

- 就绪 --> 运行
  - 调度：调度程序选择一个新的进程运行
  - 该转换可以由其他转换引起
- 运行 --> 就绪
  - 运行进程用完了时间片
  - 运行进程被中断，因为一高优先级进程处于就绪状态
  - 该转换可以引起其他转换发生

进程转换（续）

- 运行 --> 等待
  - 当一进程必须等待某事件发生，
    - OS尚未完成服务
    - 对一资源的访问尚不能进行
    - 初始化I/O 且必须等待结果
    - 等待某一进程提供输入 (IPC)
  - 可以引起其他转换发生
- 等待 --> 就绪
  - 当所等待的事件发生时

因果变迁

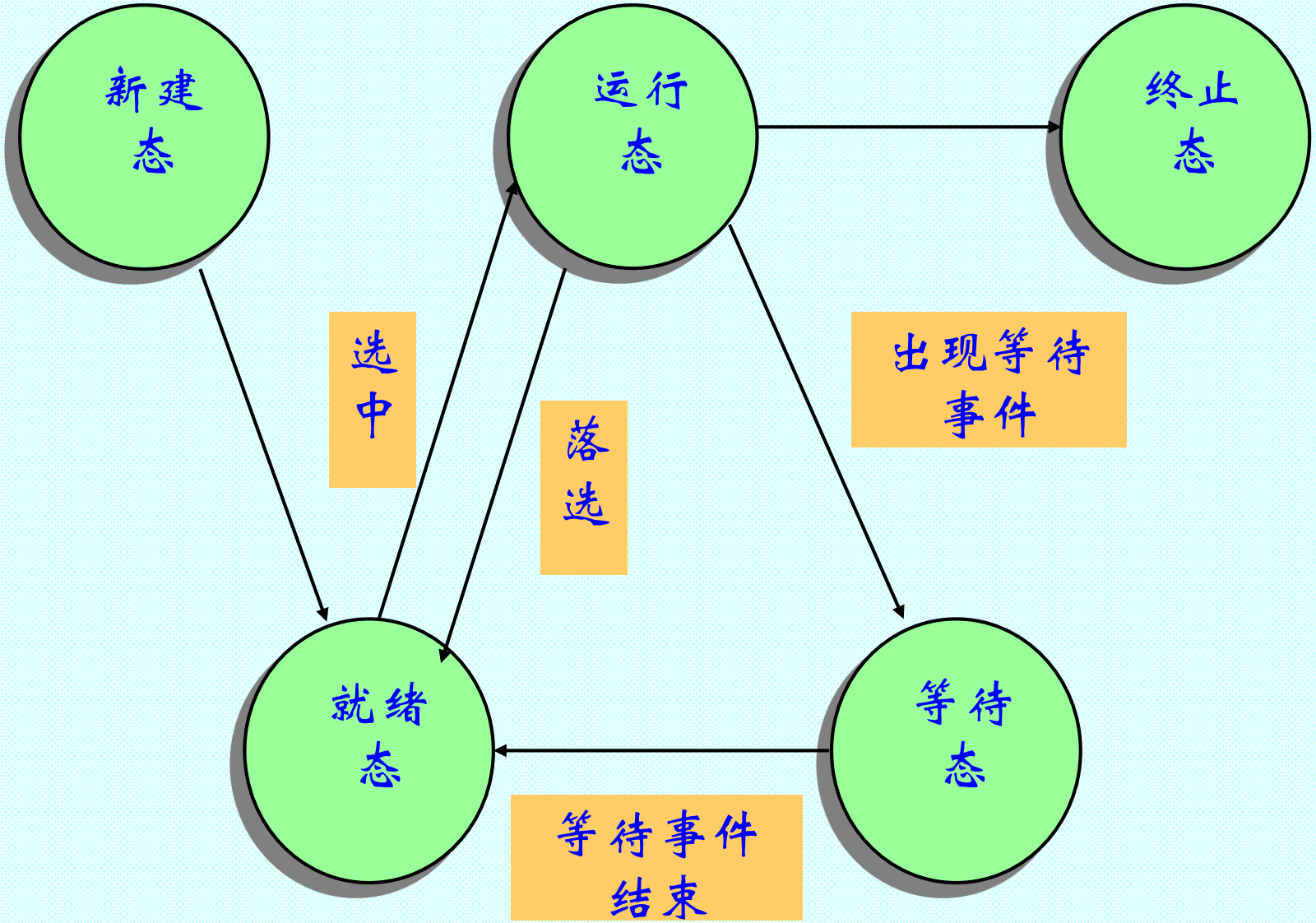
- 如果一个状态变迁是由于另一个状态变迁引起的，则这两个变迁为因果变迁。
- 思考下列说法是否对，为什么？

(1) 一个进程从运行状态变为就绪状态态，一定会引起另一个进程从就绪状态态变为运行状态。

(2) 一个进程从运行状态变为阻塞状态态，一定会引起另一进程从运行状态变为就绪状态。

(3) 一个进程从阻塞状态变为就绪状态，一定会引起另一个进程从就绪状态变为运行状态。

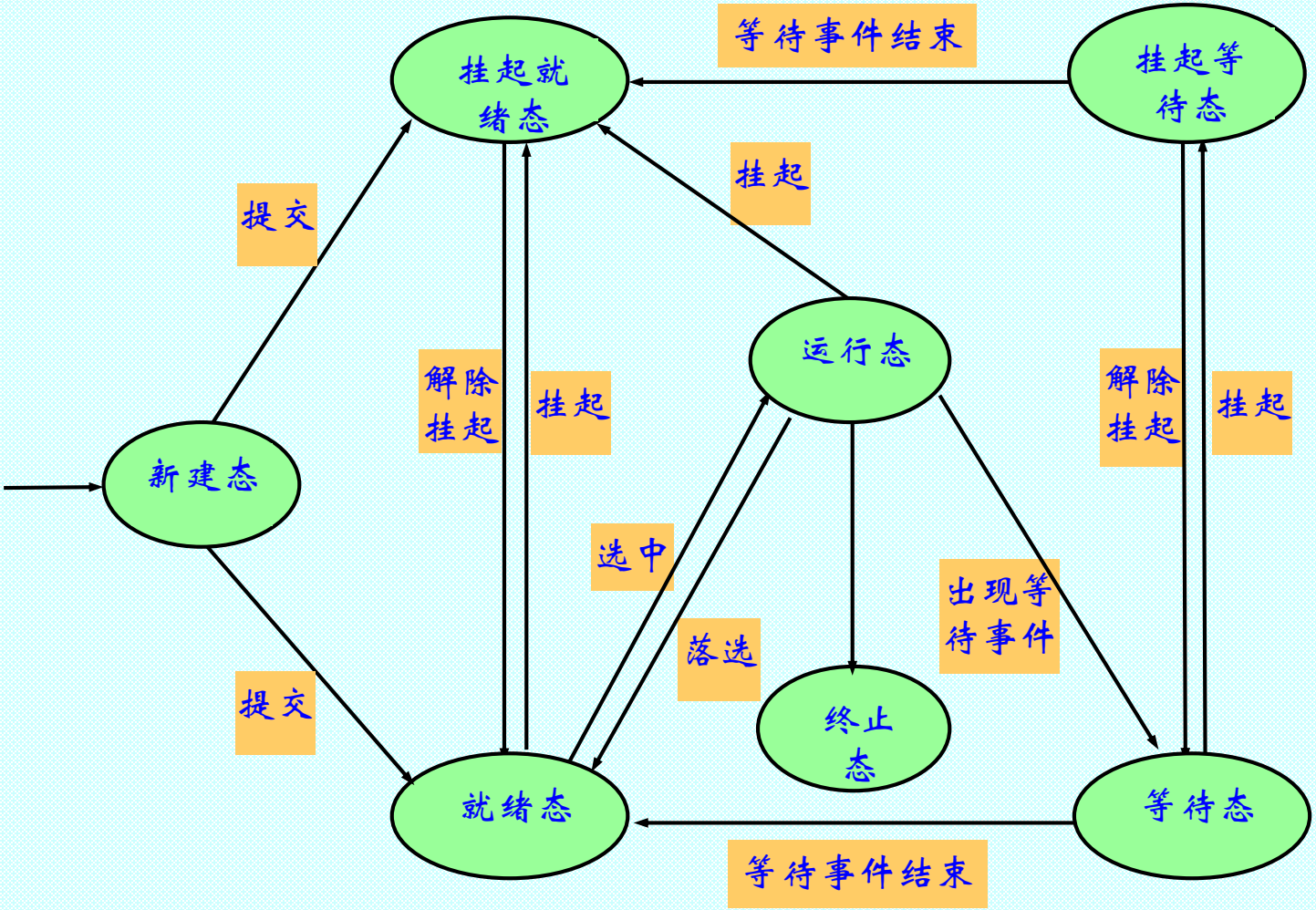
进程五态模型及其转换



进程的挂起

- 进程为什么要有“挂起”状态?
  - 为了让某些进程暂时不参与低级调度，释放它占有的资源，以平滑系统负荷的目的而需引入挂起态;
- 进程挂起的原因 ？
  - 引起进程挂起的原因多种多样。

具有挂起功能的进程状态及其转换



挂起进程具有如下特征

- 该进程不能立即被执行。
- 挂起进程可能会等待事件，但所等待事件是独立于挂起条件的，事件结束并不能导致进程具备执行条件。
- 进程进入挂起状态是由于操作系统、父进程或进程本身阻止它的运行。
- 结束进程挂起状态的命令只能通过操作系统或父进程发出。

2.3.3 进程描述和组成(1)

- 进程映象：进程某时刻的内容和状态集合。包括：
  - 进程控制块：存储进程标志信息、现场信息和控制信息。
  - 进程程序块：被进程执行的程序。
  - 进程核心栈：用来保存中断/异常现场，保存函数调用的参数、局部变量和返回地址。
  - 进程数据块：进程的私有地址空间，存放各种私有数据，包括用户栈。

进程描述和组成(2)

- 操作系统中把进程物理实体和支持进程运行的环境合称为进程上下文。
- 当系统调度新进程占有处理器时，新老进程随之发生上下文切换。进程的运行被认为是在上下文中执行。

进程描述和组成(3)
进程上下文组成

- 用户级上下文：由程序块、数据块、共享内存区、用户栈组成，占用进程的虚存空间。
- 系统级上下文：有进程控制块、内存管理信息、核心栈等操作系统管理进程所需要的信息组成。
- 寄存器上下文：由处理器状态寄存器、指令计数器、栈指针、通用寄存器等组成。

进程描述和组成(4)
Linux进程上下文组成

- Linux系统用户级上下文包括：
  - text、data、shared memory和user stack等。
- Linux系统寄存器上下文包括：
  - general register、pragram counter、EFLAGS、ESP等。
- Linux系统系统级上下文包括：
  - task_struct、mm_struct、vm_area_struct、pgd、pmd、pte和kernel stack等。

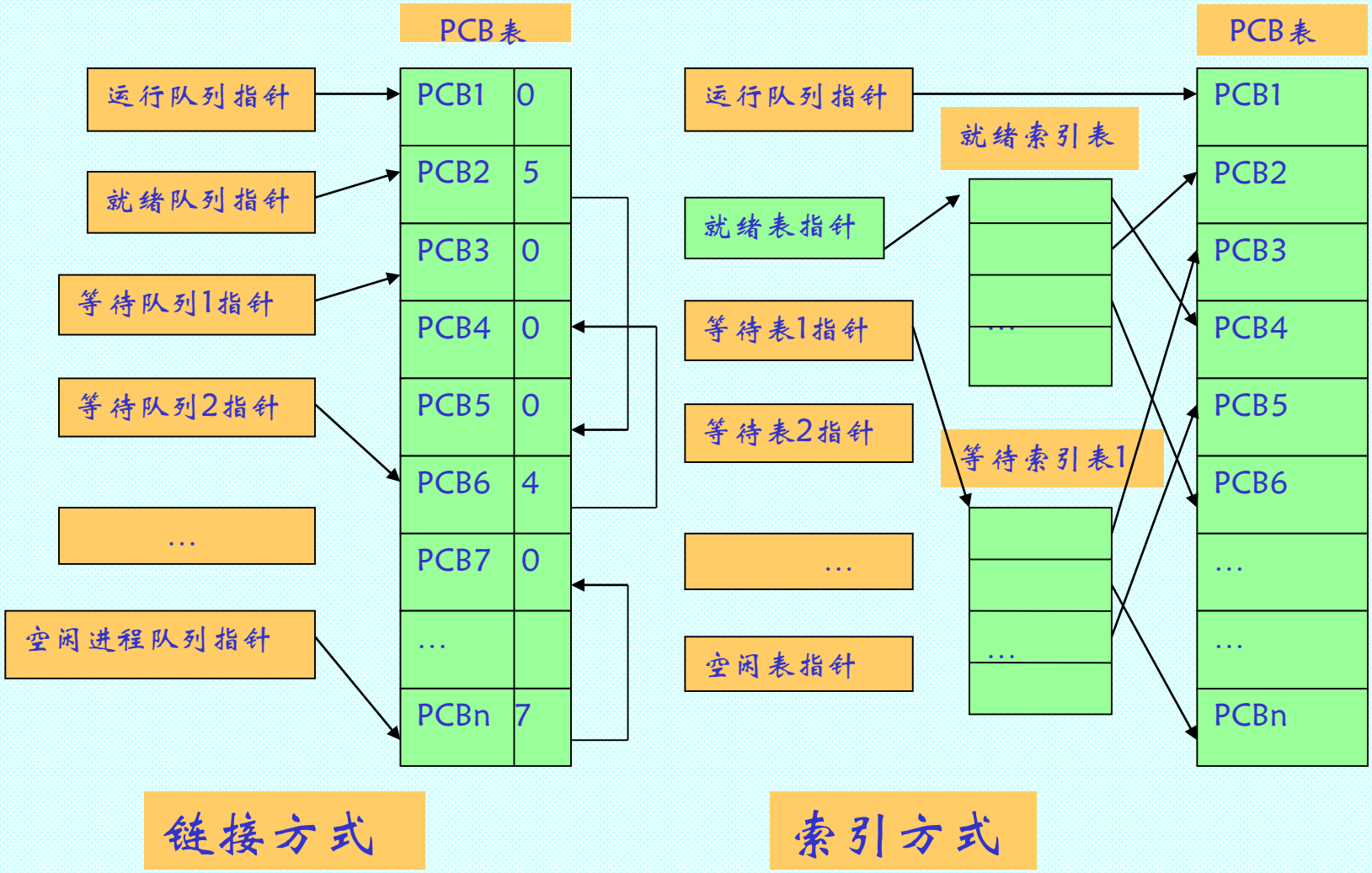
2.进程控制块

- 进程控制块PCB，是操作系统用于记录和刻划进程状态及有关信息的数据结构。也是操作系统掌握进程的唯一资料结构，它包括进程执行时的情况，以及进程让出处理器后所处的状态、断点等信息。
- 进程控制块包含三类信息
  - 标识信息
  - 现场信息
  - 控制信息

3 进程队列及其管理(1)

- 处于同一状态的所有PCB链接在一起的数据结构称为进程队列。
- 同一状态进程的PCB既可按先来先到的原则排成队列；也可按优先数或其它原则排队。
- 通用队列组织方式：
  - 线性方式
  - 链接方式
  - 索引方式。
- 进程入队和出队

进程队列及其管理(2)


PCB表
运行队列指针
就绪队列指针
等待队列1指针
等待队列2指针
…
空闲进程队列指针
PCB1 0
PCB2 5
PCB3 0
PCB4 0
PCB5 0
PCB6 4
PCB7 0
…
PCBn 7
链接方式
PCB表
运行队列指针
就绪索引表
就绪表指针
等待表1指针
等待表2指针
…
空闲表指针
等待索引表1
…
PCB1
PCB2
PCB3
PCB4
PCB5
PCB6
…
…
PCBn
索引方式

2.3.4 进程上下文切换与处理器状态转换

- 进程切换：让处于运行态的进程中断运行，让出处理器，这时要做一次进程上下文切换、即保存老进程的上下文而装入被保护了的新进程的上下文，以便新进程运行。

进程上下文切换的步骤

- 保存被中断进程的处理器现场信息
- 修改被中断进程的进程控制块有关信息，如进程状态等
- 把被中断进程的PSW加入有关队列
- 选择下一个占有处理器运行的进程
- 修改被选中进程的PSW的有关信息
- 根据被选中进程设置操作系统用到的地址转换和存储保护信息
- 根据被选中进程恢复处理器现场

进程调度和切换时机问题

- 请求调度的事件发生后，就会运行低级调度程序，低级调度程序选中新的就绪进程后，就会进行上下文切换。实际上，由于种种原因，调度和切换并不一定能一气呵成。
- 通常的做法是，由内核置上请求调度标志，延迟到上述工作完成后再进行调度和进程上下文切换，
- Linux进程调度标志位need-resched。Ｖ２.６版中，被移至thread_info 结构体中，用标志TIF_NEED_RESCHED 表示 。

Linux调度时机(1)

- 主动调度：指调用schedule( )函数来释放CPU,引起新一轮调度，通常发生在当前进程状态被改变,如:执行了read（）、write( )、exit（ ）等系统调用，导致进程终止、进程阻塞等。

Linux调度时机(2)

- 被动调度：指发生了引起调度的条件, 这时仅置进程TIF_NEED_RESCHED调度标志。调度标志设置有以下四种情况：
  - （1）时钟中断中调用函数scheduler_tick()，查看当前进程的时间片是否耗尽，如果是，则设置重调度标志；
  - （2）函数try_to_wake_up( )将阻塞的进程唤醒，把它加入运行队列时，如果其优先级比当前正在运行进程的优先级高，设置重调度标志。

Linux调度时机(3)

- 被动调度
  - （3）设置应用进程优先级参数nice值、创建新进程、SMP负载均衡时都可能使高优先级进程进入就绪状态，也可能设置重调度标志；
  - （4）执行sched_setscheduler( )（设置调度策略）、sched_yield(暂时让出处理器)、pause( )（暂停）等系统调用，均要设置重调度标志。

处理器状态转换

- 当中断/系统 调用发生时，暂时中断正在执行的用户进程，把进程从用户状态转换到内核状态，去执行操作系统服务程序以获得服务，这就是一次状态转换，
- 内核在被中断了的进程的上下文中对这个中断事件作处理，即使该中断可能不是此进程引起的。

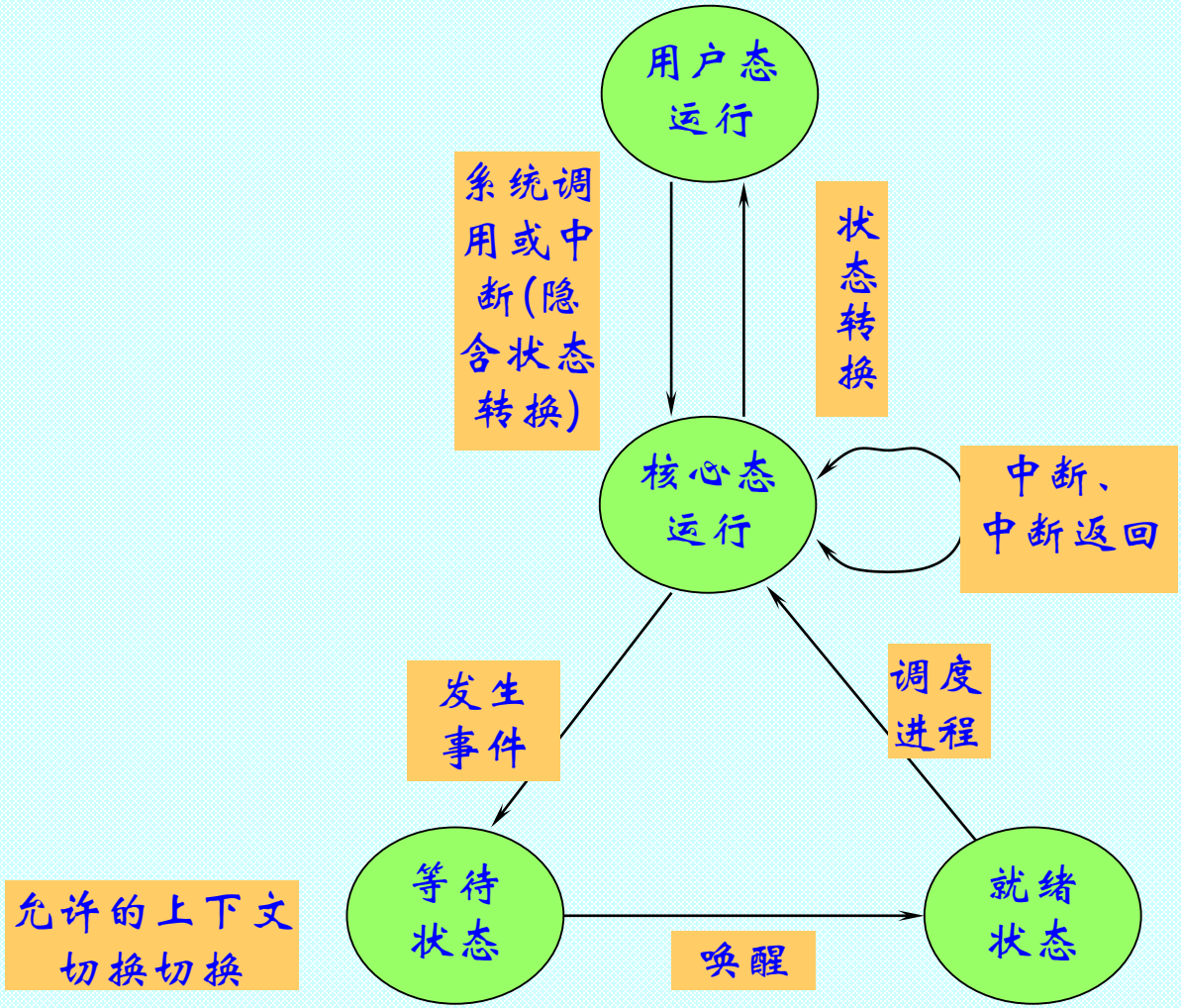
处理器状态转换步骤

- 1)保存被中断进程的处理器现场信息；
- 2)处理器从用户态转换到核心态，以便执行服务程序或中断处理程序；
- 3)如果处理中断，可根据规定的中断级设置中断屏蔽位；
- 4)根据系统调用号或中断号，从系统调用表或中断入口表找到服务程序或中断处理程序地址。

CPU上执行的进程所处活动范围

- 用户空间中，处于进程上下文，用户进程在运行，使用用户栈。
- 内核空间中，处于进程上下文，内核代表某进程在运行，使用核心栈。
- 内核空间中，处于中断上下文，与任何进程无关，中断服务程序正在处理特定中断，Intel x86未提供中断栈，借用核心栈。
- 内核空间中，内核线程(无用户地址空间的进程)运行于内核态。

Linux中进程上下文切换和处理器状态转换



Linux进程与任务

- Linux把内核空间中运行的程序称为任务，而在用户空间中运行的程序称为进程。
- 系统中存在两种进程(任务)：系统进程(任务)和用户进程(任务)，实质上是指一个进程(任务)的两个侧面，。
- 两个进程(任务)所执行的程序不同，映射到不同物理地址空间、使用不同的堆栈。

2.3.5 进程控制和管理(1)

- 处理器管理的一个主要工作是对进程的控制，包括：创建进程、阻塞进程、唤醒进程、挂起进程、激活进程、终止进程和撤销进程等。这些控制和管理功能由操作系统中的原语实现。
- 原语是在管态下执行、完成系统特定功能的过程。
- 原语和机器指令类似，其特点是执行过程中不允许被中断，是一个不可分割的基本单位，原语的执行是顺序的而不可能是并发的。

进程的控制和管理(2)

- 进程创建
- 进程撤销
- 进程阻塞
- 进程唤醒
- 进程挂起
- 进程激活

进程创建

- 步1：在进程列表中增加一项，从PCB池中申请一个空闲PCB，为新进程分配惟一的进程标识符；
- 步2：为新进程的进程映像分配地址空间，以便容纳进程实体。进程管理程序确定加载到进程地址空间中的程序；
- 步3：为新进程分配除主存空间外的其他各种所需资源；
- 步4：初始化PCB，如进程标识符、处理器初始状态、进程优先级等；
- 步5：把新进程状态置为就绪态，并移入就绪进程队列；
- 步6：通知操作系统的某些模块，如记账程序、性能监控程序。

Linux创建进程/线程

- fork( )-----父子进程是独立的进程
- clone( ) --父子进程允许共享资源
- vfork( )---子进程租用父进程地址空间

进程撤销

- 步1：根据撤销进程标识号，从相应队列中找到并移出它；
- 步2：将该进程拥有的资源归还给父进程或操作系统；
- 步3：若该进程拥有子进程，先撤销它的所有子进程，以防它们脱离控制；
- 步4：回收PCB，并归还到PCB池。

进程阻塞和唤醒

- 进程阻塞步骤：
  - 步1：停止进程执行，保存现场信息到PCB；
  - 步2：修改进程PCB有关内容，如进程状态由运行态改为等待态等，并把修改状态后的进程移入相应事件的等待队列中；
  - 步3：转入进程调度程序去调度其他进程运行。
- 进程唤醒步骤：
  - 步1：从相应的等待队列中移出进程；
  - 步2：修改进程PCB的有关信息，如进程状态改为就绪态，并移入就绪队列；
  - 步3：若被唤醒进程比当前运行进程优先级高，重新设置调度标志。

2.4 线程及其实现

- 主要内容
  - 引入多线程的动机
  - 多线程环境中的进程和线程
  - 线程的实现

2.4.1 引入多线程的动机

- 对进程系统必须完成的操作:
  - 创建进程
  - 撤消进程
  - 进程切换
- 缺点:
  - 时间空间开销大，限制并发度的提高

2.4.1 引入多线程的动机

- 进程的局限性:
  - 在操作系统中，进程的引入提高了计算机资源的利用效率。但在进一步提高进程的并发性时，人们发现进程切换开销占的比重越来越大;
  - 传统的进程不能很好的利用多处理器，因为一个进程在某个时刻只能使用一个处理器;
  - 进程间通信的效率受到限制。
- 引入线程的目的:
  - 减小（进程/线程）上下文切换开销;
  - 更好支持多处理器（MP），达到最大程度的并行;
  - 简化进程间的通信。

线程的概念(1)

- 操作系统中引入进程的目的是为了使多个程序并发执行，以改善资源使用率和提高系统效率，
- 操作系统中再引入线程，则是为了减少程序并发执行时所付出的时空开销，使得并发粒度更细、并发性更好。

线程的概念(2)

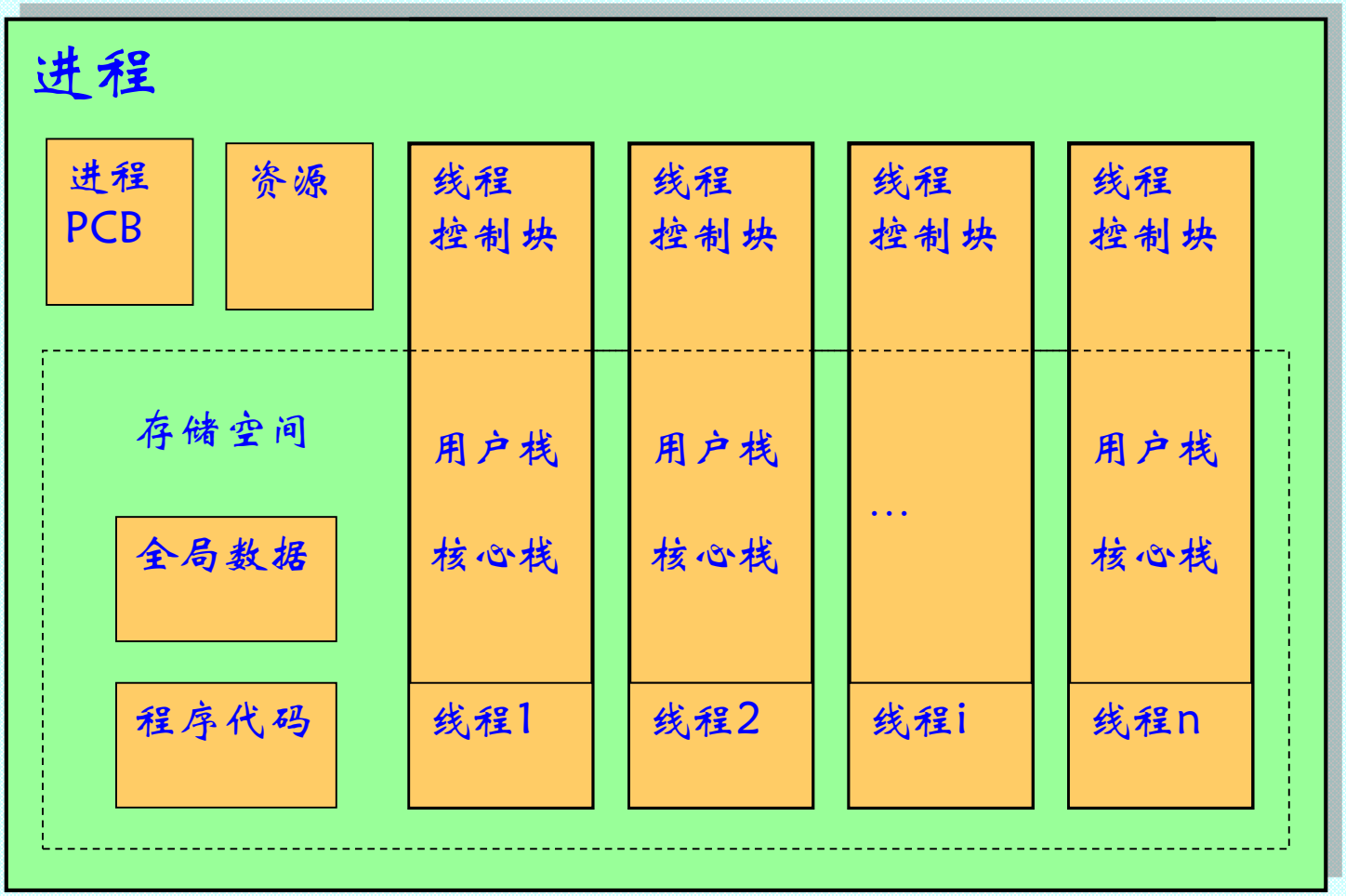
- 解决问题的基本思路：
  - 把进程的两项功能——“独立分配资源”与“被调度分派执行”分离开来。
  - 进程作为系统资源分配和保护的独立单位，不需要频繁地切换；
  - 线程作为系统调度和分派的基本单位，能轻装运行，会被频繁地调度和切换，在这种指导思想下，产生了线程的概念。

多线程结构进程的优点

- 1.快速线程切换
- 2.通信易于实现
- 3.减少管理开销
- 4.并发程度提高

2.4.2 多线程环境中的进程与线程

多线程结构进程



多线程环境中进程的定义

- 进程是操作系统中除处理器外进行的资源分配和保护的基本单位。
- 它有独立的虚拟地址空间，容纳进程映像(如与进程关联的程序与数据)，并以进程为单位对各种资源实施保护，如受保护地访问处理器、文件、外部设备及其他进程(进程间通信)。

多线程环境中的线程概念

- 线程是操作系统进程中能够独立执行的实体（控制流）,是处理器调度和分派的基本单位。
- 线程是进程的组成部分，每个进程内允许包含多个并发执行的实体（控制流），这就是多线程。

线程又称轻量进程

- 线程运行在进程的上下文中，并使用进程的资源和环境。
- 系统调度的基本单位是线程而不是进程,每当创建一个进程时，至少要同时为该进程创建一个线程。
- Linux线程做法不一样。

线程组成

- （１）线程唯一标识符及线程状态信息(运行态、就绪态、阻塞态和终止态)；
- （２）线程是一条执行路径，有独立的程序计数器；未运行时保护线程上下文。
- （３）线程有执行栈和存放局部变量的私用存储空间。
- （4）可访问所属进程的内存和资源，并与该进程中的其他线程共享这些资源。

线程的状态

- 线程状态有：运行、就绪、等待和终止，状态转换也类似于进程。
- 挂起状态对线程是没有意义，如果进程挂起后被对换出主存，则它的所有线程因共享进程的地址空间，也必须全部对换出去。

多线程技术的应用

- 进程中线程多种组织方式
  - 第一种是调度员 / 工作者模式
  - 第二种是组模式
  - 第三种是流水线模式
- 多线程技术的应用
  - 前台和后台工作
  - C/S应用模式
  - 异步处理
  - 加快执行速度
  - 设计用户接口

2.4.3 线程的实现

- 从实现角度看，线程分成
  - 用户级线程ULT(如Java， Informix)。
  - 内核级线程KLT(如OS/2)。
  - 混合式线程(如Solaris)。

用户级线程（ULT）

- 由用户应用程序建立、调度和管理的线程。
  - 不依赖于OS内核，应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程。如：数据库系统informix，图形处理Aldus PageMaker。
  - 调度由应用软件内部进行，通常采用非抢先式和更简单的规则，也无需用户态/核心态切换，所以速度特别快。一个线程发起系统调用而阻塞，则整个进程在等待。时间片分配给进程，多线程则每个线程就慢。
- 线程库：基于多线程的应用程序的开发和运行环境。

用户级线程的活动

- 内核不知道线程的活动，但仍然管理线程所属进程的活动；
- 当线程调用系统调用时，整个进程阻塞；
- 但对线程库来说，线程仍然是运行状态
- 即线程状态是与进程状态独立的。

用户级线程优缺点

- 优点
  - 线程切换不调用内核
  - 调度是应用程序特定的：可以选择最好的算法。
  - ULT可运行在任何操作系统上（只需要线程库）。
- 缺点
  - 大多数系统调用是阻塞的，因此核心阻塞进程，故进程中所有线程将被阻塞。
  - 核心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于两个处理器上。

内核级线程（KLT）

- 由操作系统的内核建立、调度和管理的线程。
  - 所有线程管理由内核完成
  - 没有线程库，但对内核线程工具提供API
  - 内核维护进程和线程的上下文
  - 线程之间的切换需要内核支持
  - 以线程为基础进行调度
  - 例子：Windows NT，OS/2

内核级线程的优点及缺点

- 优点：
  - 对多处理器，内核可以同时调度同一进程的多个线程
  - 阻塞是在线程一级完成
  - 内核例程是多线程的
- 缺点：
  - 在同一进程内的线程切换调用内核，导致速度下降

用户级和内核级线程比较

- 调度和切换速度

  ULT切换快，KLT切换与进程切换相似。ULT通常发生在一个应用进程的诸线程中，无需通过中断进入内核。

- 系统调用

  ULT进行系统调用时，会引起进程的阻塞；KLT进行系统调用时只会引起该线程阻塞。

- 线程执行时间

  ULT以进程为单位调度，KLT以线程为单位调度。

  ULT：进程A有1个线程，进程B有100个线程，则A的线程比B快

  KLT：进程A有1个线程，进程B有100个线程，则B比A快

- 使用范围：

  ULT广，任何OS，KLT需OS内核支持

- 调度算法

  ULT与OS调度算法无关，可针对应用优化

- 多处理器支持

  KLT可充分利用多处理器

ULT和KLT结合方法

- 线程创建在用户空间完成
- 大量线程调度和同步在用户空间完成
- 程序员可以调整KLT的数量
- 可以取两者中最好的
- 例子：Solaris

NT线程的有关API

- CreateThread()函数在调用进程的地址空间上创建一个线程，以执行指定的函数；返回值为所创建线程的句柄。
- ExitThread()函数用于结束本线程。
- SuspendThread()函数用于挂起指定的线程。
- ResumeThread()函数递减指定线程的挂起计数，挂起计数为0时，线程恢复执行。

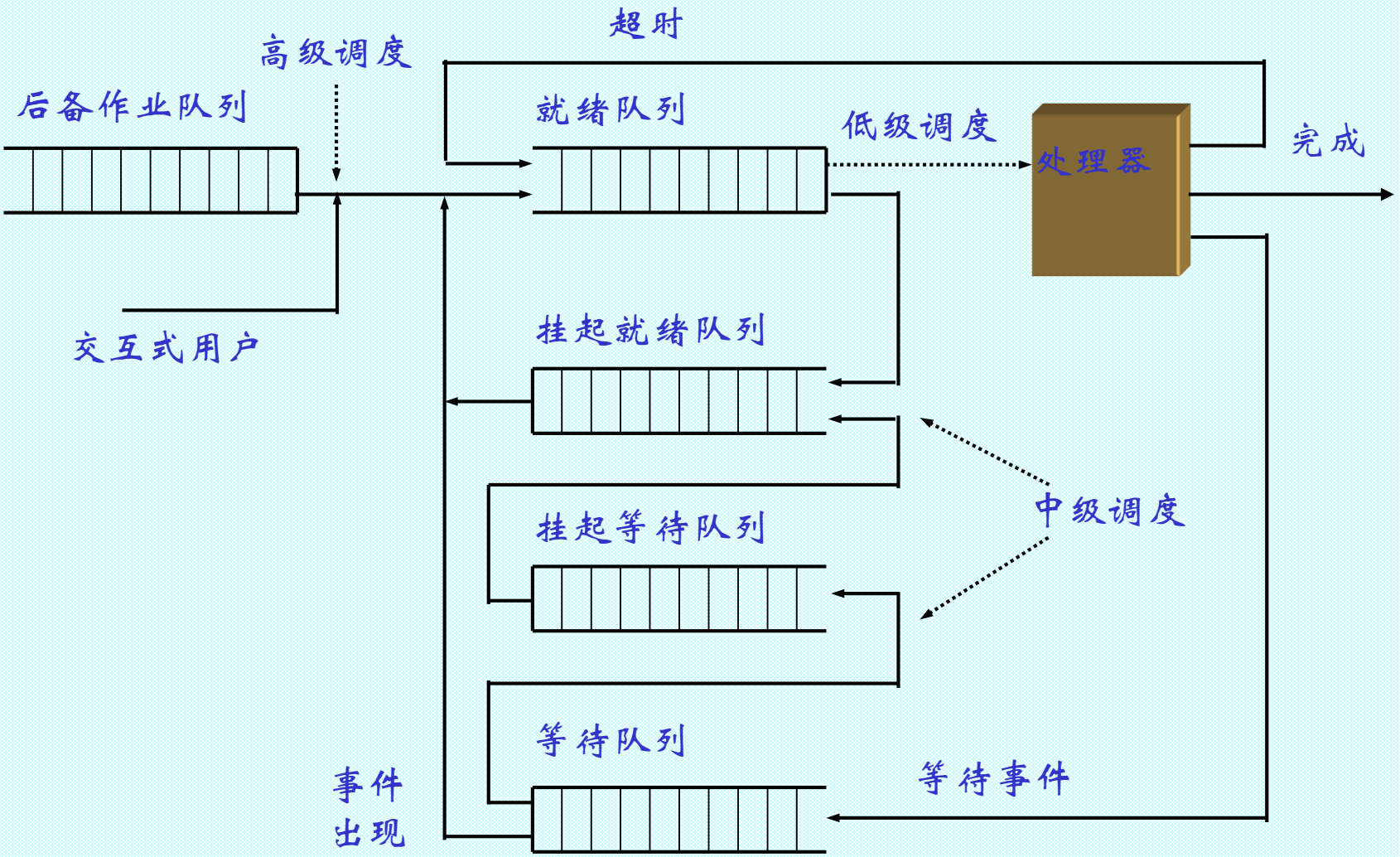
2.5处理机调度

- 处理机调度层次
- 选择调度算法的原则
- 作业管理与调度
- 低级调度功能和类型
- 作业调度和低级调度算法

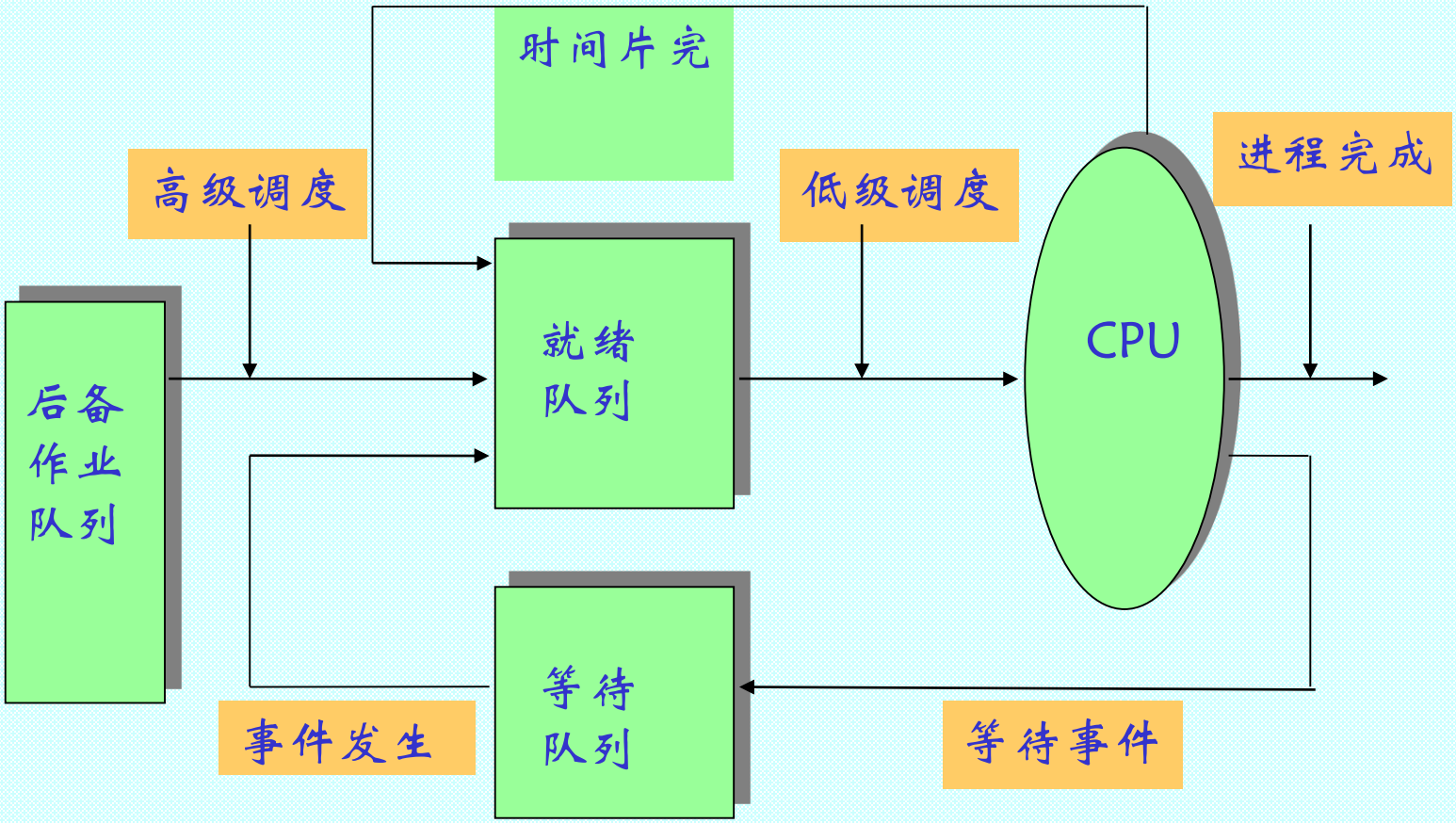
2.5.1 处理机调度层次

- 作业从进入系统成为后备作业开始，直到运行结束退出系统为止，需经历不同级别的调度。
  - 高级调度：也叫作业调度
  - 中级调度：也叫平衡调度
  - 低级调度：也叫处理器调度、进程调度或线程调度。

处理器三级调度模型


超时
高级调度
后备作业队列
就绪队列
低级调度
处理器
完成
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

处理器两级调度模型



2.5.2 选择调度算法原则(1)

- 资源利用率
  - CPU利用率=CPU有效工作时间/CPU总的运行时间，
  - CPU总的运行时间=CPU有效工作时间+CPU空闲等待时间

选择调度算法的原则(2)

- 响应时间
  - 交互式进程从提交一个请求(命令)到接收到响应之间的时间间隔称响应时间。
  - 使交互式用户的响应时间尽可能短，或尽快处理实时任务。
  - 这是分时系统和实时系统衡量调度性能的一个重要指标。

选择调度算法的原则(3)

- 周转时间
  - 批处理用户从作业提交给系统开始，到作业完成为止的时间间隔称作业周转时间，应使作业周转时间或平均作业周转时间尽可能短。
  - 这是批处理系统衡量调度性能的一个重要指标。

选择调度算法的原则(4)

- 吞吐率
  - 单位时间内处理的作业数。
- 公平性
  - 确保每个用户每个进程获得合理的CPU份额或其他资源份额，不会出现饿死情况。

作业周转与平均周转时间

- 如果作业i提交给系统的时刻是ts，完成时刻是tf，该作业的周转时间ti为：

$$ti = tf - ts$$

- 实际上，它是作业在系统里的等待时间与运行时间之和。
- 为了提高系统的性能，要让若干个用户的平均作业周转时间和平均带权周转时间最小。

$$\text{平均作业周转时间 } T = \left(\sum_{i=1}^{n} t_i\right) / n$$

作业带权周转时间和平均作业带权周转时间

- 如果作业i的周转时间为ti，所需运行时间为tk，则称wi=ti /tk为该作业的带权周转时间。
- ti是等待时间与运行时间之和，故带权周转时间总大于1。
- 平均作业带权周转时间W=$(\sum_{i=1}^{n} w_i)/ n$

2.5.3 作业管理与调度

- 作业管理任务
  - （1）作业组织；
  - （2）作业调度；
  - （3）运行控制。

1.作业和进程的关系

- 作业：用户提交给操作系统计算的一个独立任务；由程序、数据和作业说明书组成。
- 作业步：作业的每个加工步骤。
- 作业是任务实体，进程是完成任务的执行实体；没有作业任务，进程无事可干，没有进程，作业任务没法完成。
- 作业概念更多地用在批处理操作系统，而进程则可以用在各种多道程序设计系统。

2. 作业组织、调度和控制

- 批作业的组织和管理
  - 批作业的输入：采用脱机控制方式，由SPOOLing系统成批接收并控制作业输入，并将其存放在输入井，然后在系统的管理和控制下被调度和执行。
  - 批作业的建立
    - 作业控制语言
    - 作业说明书
    - 作业控制块

作业控制块

- 多道批处理操作系统具有独立的作业管理模块，必须像进程管理一样为每一个作业建立作业控制块（JCB）。
- JCB通常是在批作业进入系统时，由Spooling系统建立的，它是作业存在于系统的标志，作业撤离时，JCB也被撤销。
- JCB的主要内容包括：
  - 作业情况
  - 资源需求
  - 资源使用情况

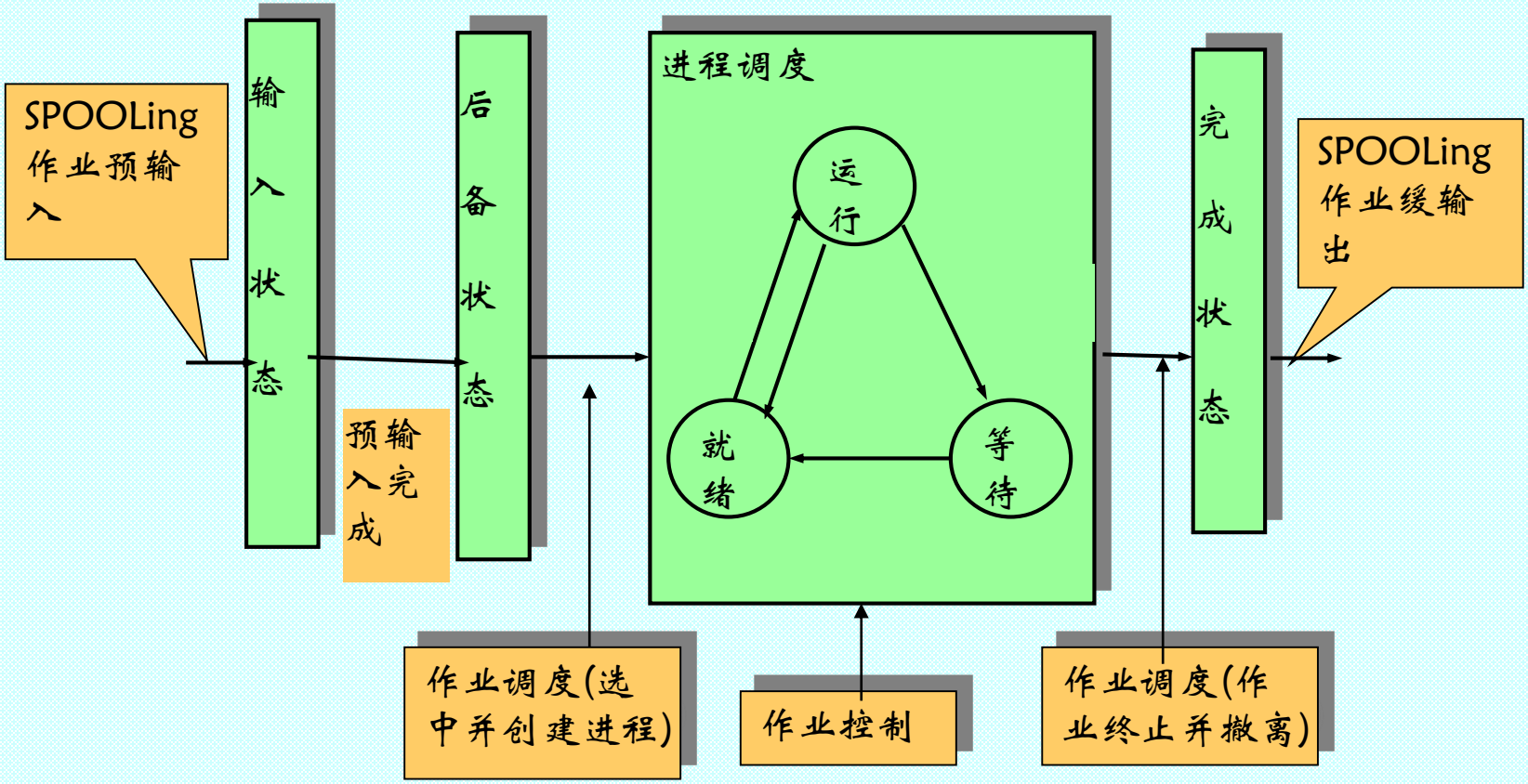
作业生命周期状态

- 输入状态
  - 作业被提交给机房后或用户通过终端键盘向计算机中键入其作业时所处的状态。
- 后备状态
  - 作业的全部信息都已通过输入设备输入，并由操作系统将其存放在磁盘的某些盘区中等待运行。
- 执行状态
  - 作业调度程序选中而被送入主存，并建立进程投入运行。
- 完成状态
  - 作业完成其全部运行，释放其所占用的全部资源。

批作业的调度

- 作业调度：指按照某种算法从后备作业队列中选择部分作业进入内存运行，当作业运行结束时做好善后工作。
- 作业调度程序的任务：
  - (1) 选择作业；
  - (2) 分配资源；
  - (3) 创建进程；
  - (4) 作业控制；
  - (5) 后续处理；

作业调度与进程调度的关系



交互作业的组织和管理

- 分时系统的作业就是用户的一次上机交互过程，可认为终端进程的创建是一个交互型作业的开始，退出命令运行结束代表用户交互型作业的中止。
- 交互作业的情况和资源需求通过操作命令告知系统，分时用户逐条输入命令，即提交作业（步）和控制作业运行，系统则逐条执行并给出应答，每键入一条或一组有关操作命令，便在系统内部创建一个进程或若干进程来完成相应命令。
- 键盘命令有：作业控制类；资源申请类；文件操作类；目录操作类；设备控制类等。

2.5.4低级调度的功能和类型

- 低级调度的主要功能
  - 调度程序两项任务：调度和分派。
  - **调度**--实现调度策略，确定就绪进程/线程竞争使用处理器的次序的裁决原则，即进程/线程何时应放弃CPU和选择哪个来执行；
  - **分派**--实现调度机制，确定如何时分复用CPU，处理上下文交换细节，完成进程/线程和CPU的绑定和放弃的实际工作。

低级调度的基本类型

- 第一类称剥夺式：
  - 又称抢占式。当进程/线程正在处理器上运行时，系统可根据规定的原则剥夺分配给此进程/线程的处理器，并将其移入就绪队列，选择其他进程/线程运行。
  - 两种处理器剥夺原则：
    - 高优先级进程/线程可剥夺低优先级进程/线程，
    - 当运行进程/线程时间片用完后被剥夺。
- 第二类称非剥夺式：
  - 又称非抢占式。一旦某个进程/线程开始运行后便不再让出处理器，除非该进程/线程运行结束或主动放弃处理器，或因发生某个事件而不能继续执行。

2.6.5 作业调度和低级调度算法

- 1.先来先服务算法
- 2. 最短作业优先算法
- 3. 最短剩余时间优先算法
- 4. 最高响应比优先算法
- 5. 优先级调度算法
- 6. 时间片轮转调度算法
- 7. 多级反馈队列调度

1.先来先服务算法

- 先来先服务是按照作业进入系统后备队列的先后次序来挑选作业，先进入系统的作业优先被挑选进入内存。
- 三个作业同时到达系统并立即进入调度：作业名/所需CPU时间。
  - 作业提交顺序：作业1/28，作业2/9，作业3/3。采用FCFS算法，平均作业周转时间为35。
  - 若三个作业提交顺序改为作业2、1、3，平均作业周转时间约为29。
  - 若三个作业提交顺序改为作业3、2、1，平均作业周转时间约为18。
- FCFS调度算法的平均作业周转时间与作业提交的顺序有关。

1.先来先服务算法

作业	提交时刻	运行时间	开始时刻	完成时间	周转时间	带权周转时间
1	0	28	0	28	28	1
2	0	9	28	37	37	4.1
3	0	3	37	40	40	13.3
平均周转时间 t＝35 平均带权周转时间 w ＝6.1						

2. 最短作业优先算法(1)

- SJF算法以进入系统的作业所要求的CPU时间为标准，总选取估计计算时间最短的作业投入运行。
- 算法易于实现，效率不高，主要弱点是忽视了作业等待时间。会出现饥饿现象。
- SJF的平均作业周转时间比FCFS要小，故它的调度性能比FCFS好。
- 实现SJF调度算法需要知道作业所需运行时间，否则调度就没有依据，要精确知道一个作业的运行时间是办不到的。

2.最短作业优先算法(2)

- 四个作业同时到达系统并进入调度： 作业名/所需CPU时间:作业1/9，作业2/4 ，作业3/10，作业4/8。
- SJF作业调度顺序为作业2、4、1、3，

  平均作业周转时间T = 17，

  平均带权作业周转时间W= 1.98。
- 如果施行FCFS调度算法，

  平均作业周转时间T =19，

  平均带权作业周转时间W = 2.61。

2.最短作业优先算法(3)

作业	提交时刻	运行时间	开始时刻	完成时间	周转时间	带权周转时间
1	0	9	12	21	21	2.3
2	0	4	0	4	4	1
3	0	10	21	31	31	3.1
4	0	8	4	12	12	1.5
平均周转时间 t＝17 平均带权周转时间 w ＝1.98						

shortest next CPU burst time(1)

- burst--最短下一个CPU突发周期长度
- 计算进程/线程下一个CPU周期长度

$$\tau_{n+1}=\alpha t_n+(1-\alpha)\tau_n$$

- t_n是进程/线程最近一个CPU周期长度，是最近信息；
- τ_n是估算的第n个CPU周期值，是历史信息。

shortest next CPU burst time(2)

- 实际值(t i)　6　4　6　4　13　13
- 估算值(τ i)　10　8　6　6　5　9　11

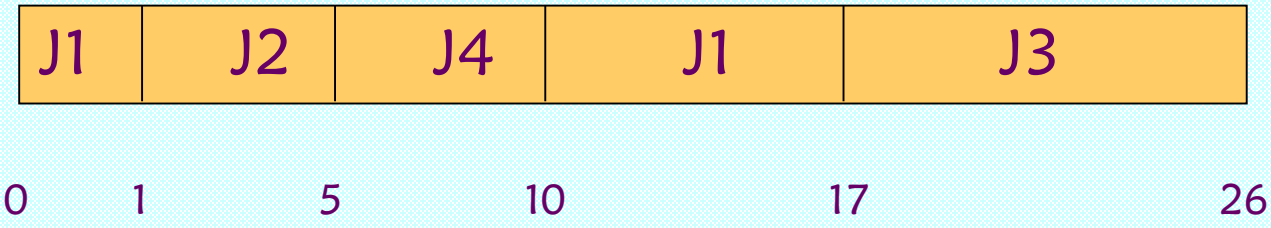
3.最短剩余时间优先算法(1)

- SRTF把SJF算法改为抢占式的。一个新作业进入就绪状态，如果新作业需要的CPU时间比当前正在执行的作业剩余下来还需的CPU时间短，SRTF强行赶走当前正在执行作业。称最短剩余时间优先算法
- 此算法不但适用于JOB调度，同样也适用于进程调度。

最短剩余时间优先算法(2)

- 四个作业其到达系统/所需CPU时间如下：

Job1-0/8，Job2-1/4，Job3-2/ 9，Job4-3/5。



- SRTF调度平均等待时间=6.5毫秒。
- SJF调度平均等待时间=7.75毫秒。

4.最高响应比优先算法

- FCFS与SJF是片面的调度算法。FCFS只考虑作业等候时间而忽视了作业的计算时问，SJF只考虑用户估计的作业计算时间而忽视了作业等待时间。
- HRRF是介乎这两者之间的折衷算法，既考虑作业等待时间，又考虑作业的运行时间，既照顾短作业又不使长作业的等待时间过长，改进了调度性能。

响应比定义

- 响应比 ＝1+已等待时间/估计运行时间

- 短作业容易得到较高响应比，
- 长作业等待时间足够长后，也将获得足够高的响应比，
- 饥饿现象不会发生。

HRRF算法举例

四个作业到达系统时间/所需CPU时间:作业1-0/20，作业2-5/15，作业3-10 /5，作业4- 15/ 10。

- SJF调度顺序为作业1、3、4、2，平均作业周转时间T=25， 平均带权作业周转时间W=2.25 。
- FCFS调度顺序为作业1、2、3、4，平均作业周转时间T=28.75， 平均带权作业周转时间W=3.125 。
- HRRF调度顺序为作业1、3、2、4，平均作业周转时间T=26.25， 平均带权作业周转时间W=2.46 。

HRRF算法举例

作业	提交时刻	运行时间	开始时刻	完成时间	周转时间	带权周转时间
1	0	20	0	20	20	1
2	5	15	25	40	35	2.33
3	10	5	20	25	15	3
4	15	10	40	50	35	3.5
平均周转时间 t＝26.25 平均带权周转时间 w ＝2.46						

5 优先级调度算法(1)

静态优先数法

- 使用外围设备频繁者优先数大，这样有利于提高效率；
- 重要算题程序的进程优先数大，这样有利于用户；
- 进入计算机时间长的进程优先数大，这样有利于缩短作业完成的时间；
- 交互式用户的进程优先数大，这样有利于终端用户的响应时间等等。

优先权调度算法(2)

动态优先数法

①根据进程占有CPU时间多少来决定,当进程占有CPU时间愈长,那么，在它被阻塞之后再次获得调度的优先级就越低，反之,进程获得调度的可能性越大;

②根据进程等待CPU时间多少来决定,当进程在就绪队列中等待时间愈长,那么，在它被阻塞之后再次获得调度的优先级就越高，反之,进程获得调度的可能性越小。

6 时间片轮转调度算法

- 时间片调度：调度程序每次把CPU分配给就绪队列首进程使用一个时间片，例如100ms，就绪队列中的每个进程轮流地运行一个时间片。当这个时间片结束时，强迫一个进程让出处理器，让它排列到就绪队列的尾部，等候下一轮调度。
- 轮转策略可防止那些很少使用外围设备的进程过长的占用处理器而使得要使用外围设备的那些进程没有机会去启动外围设备。
- 轮转策略与间隔时钟

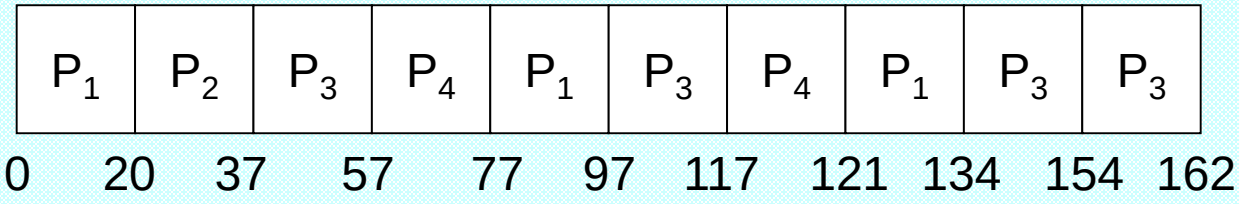
时间片长度的确定

- 时间片长度变化的影响
  - 过长－>退化为FIFO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。

轮转法举例(时间片长= 20)

进程	突发周期
P_1	53
P_2	17
P_3	68
P_4	24

- Gantt 图:



- 平均周转时间比SJF大，但响应较快。

7 多级反馈队列调度

- 又称反馈循环队列或多队列策略。主要思想是将就绪进程分为两级或多级，系统相应建立两个或多个就绪进程队列，较高优先级的队列一般分配给较短的时间片。
- 处理器调度先从高级就绪进程队列中选取可占有处理器的进程，只有在选不到时，才从较低级的就绪进程队列中选取。

一个三级反馈队列调度策略