《操作系统》2.3：进程及其实现

2022-9-15

章节安排

- 进程的定义和属性
- 进程的状态和转换
- 进程的描述和组成
- 进程切换与模式切换
- 进程的控制和管理

- 进程的定义和属性
- 进程的状态和转换
- 进程的描述和组成
- 进程切换与模式切换
- 进程的控制和管理

2.3.1：进程的定义和属性

- 进程是操作系统中最基本、重要的概念
- 是多道程序系统出现后，为了刻画系统内部出现的动态情况，描述系统内部各道程序的活动规律引进的一个概念
- 所有多道程序设计操作系统都建立在进程的基础上

2.3.1（续）

- 理论角度
  - 进程是对正在运行的程序过程的抽象
- 实现角度
  - 目的在于清晰地刻划动态系统的内在规律
  - 有效管理和调度进入计算机系统主存储器运行的程序

2.3.1（续）：引入进程的原因

- 刻画系统的动态性，发挥系统的并发性，提高资源利用率
  - 在多道程序设计环境下，程序可以并发执行，一个程序的任意两条指令之间都可能发生随机事件而引发程序切换
  - 每个程序的执行都可能不是连续的而是走走停停
- 解决系统的“共享性”，正确描述程序的执行状态

2.3.1（续）

- “可再入”程序（re-entrant）
  - 能被多个程序同时调用的程序
  - 是纯代码，即它在执行中自身不被改变；调用它的各程序应提供工作区，因此，“可再入”程序可被多个程序同时调用
- “可再用”程序
  - 被调用过程中自身会被修改，在调用它的程序退出之前不允许其他程序来调用它

2.3.1（续）

```
int g_var = 1;
int f() {
  g_var = g_var + 2;
  return g_var;
}

int g() {
  return f() + 2;
}
```

函数 f，g 是可再入的吗？

2.3.1（续）

```
int f(int i) {
  return i+2;
}

int g(int i) {
  return f(i) + 2;
}
```

现在呢？

2.3.1（续）：可再入程序 示例

- 编译程序P编译源程序甲，从A点开始工作，执行到B点时需将信息记到磁盘上，且程序P在B点等待磁盘传输
- 为提高系统效率，利用编译程序的“可再入”性，让编译程序P再为源程序乙进行编译，仍从A点开始工作
  - **现在怎样来描述编译程序P的状态呢?**
  - **称它为在B点等待磁盘传输状态?**
  - **还是称它为正在从A点开始执行的状态?**

2.3.1（续）：可再入程序 示例

- 把编译程序P，与服务对象联系起来，P为甲服务就说构成进程P甲，P为乙服务则构成进程P乙
- 两个进程虽共享程序P，但它们可同时执行且彼此按各自的速度独立执行。
- 程序与计算(程序的执行)不再一一对应

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一个基本单位。

2.3.1（续）：进程的定义

- 进程是程序的执行
- 并发程序称为进程
- 进程是可以和别的计算并发执行的计算
- 进程是一个数据结构及其上进行操作的程序
- 进程是一个程序及其数据在处理机上顺序执行时所发生的活动
- 进程是程序在一个数据集合上运行的过程，它是系统进行资源分配和调度的一个独立单位
- 进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位（进程实体是由程序段、相关的数据段和进程控制块组成的）

2.3.1（续）：进程的定义

- 进程是程序的执行
- 并发程序称为进程
- 进程是可以和别的计算
- 进程是一个数据结构及
- 进程是一个程序及其数据在处理机上顺序执行时所发生的活动
- 进程是程序在一个数据集合上运行的过程，它是系统进行资源分配和调度的一个独立单位
- 进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位（进程实体是由程序段、相关的数据段和进程控制块组成的）

进程是一个可并发执行的具有独立功能的程序关于某个数据集合的一次执行过程，也是操作系统进行资源分配和保护的**基本单位**

2.3.1（续）

```
int f(string file) {
   ifstream fin(file);
   int val;
   fin >> val;
   val += 2;
   ofstream fout(file);
   fout << val;
}
```

若并发执行多个f(FILE1)会如何？

fout对象的作用可以理解成：向文件中进行写操作 【从缓冲区->硬盘】

fin对象的作用可以理解成：对文件中的内容进行读操作 【从硬盘->缓冲区】

2.3.1（续）：进程的含义

- 进程（process）是一个动态的概念，而程序(program)是一个静态的概念
- 进程包含了一个数据集合和运行其上的程序
- 同一程序同时运行于若干不同的数据集合上时，它将属于若干个不同的进程，或者说，两个不同的进程也可以包含相同的程序
- 系统分配资源是以进程为单位的，所以只有进程才可能在不同的时刻处于几种不同的状态
- 既然进程是资源分配的单位，处理机也是按进程分配的，因此，从微观上看，进程是轮换占有处理机而运行的，从宏观上看，进程是并发地运行的；从局部上看，每个程序是串行执行的，从整体上看，多个进程是并发地执行的

2.3.1（续）：进程的属性（vs. 程序）

1) 结构性

- 进程包含了数据集合和运行于其上的程序。每个进程至少包含三个组成要素：程序块、数据块和进程控制块

2) 共享性

- 同一程序运行于不同数据集合上时，构成不同的进程。多个不同的进程可以共享相同的程序，所以进程和程序不是一一对应的
- 进程之间可以共享某些公用变量
- 进程的运行环境不再是封闭的

2.3.1（续）：进程的属性（vs. 程序）

3) 动态性

- 进程由创建而产生，由调度而执行，由撤销而消亡
- 程序是一组有序指令序列，作为一种系统资源是永久存在的

4) 独立性

- 进程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）

2.3.1（续）：进程的属性（vs. 程序）

5) 制约性

- 并发进程之间存在着制约关系，进程在进行的关键点上需要相互等待或互通消息，以保证程序执行的可再现性和计算结果的唯一性

6) 并发性

- 在一个单处理器系统环境下，各个进程轮流占用处理器

2.3.1（续）：总结

- 进程能更真实的描述并发，而程序不能
- 进程是由程序和数据两部分组成的
- 进程是动态的，而程序是静态的
- 进程有生命周期，而程序相对长久
- 一个程序可以对应多个进程
- 进程可以创建其他进程而程序不能
- 进程是一个与时间和空间相关的概念，而程序不是

- 进程的定义和属性
- 进程的状态和转换
- 进程的描述和组成
- 进程切换与模式切换
- 进程的控制和管理

2.3.2：进程的状态和转换

1. 三态模型

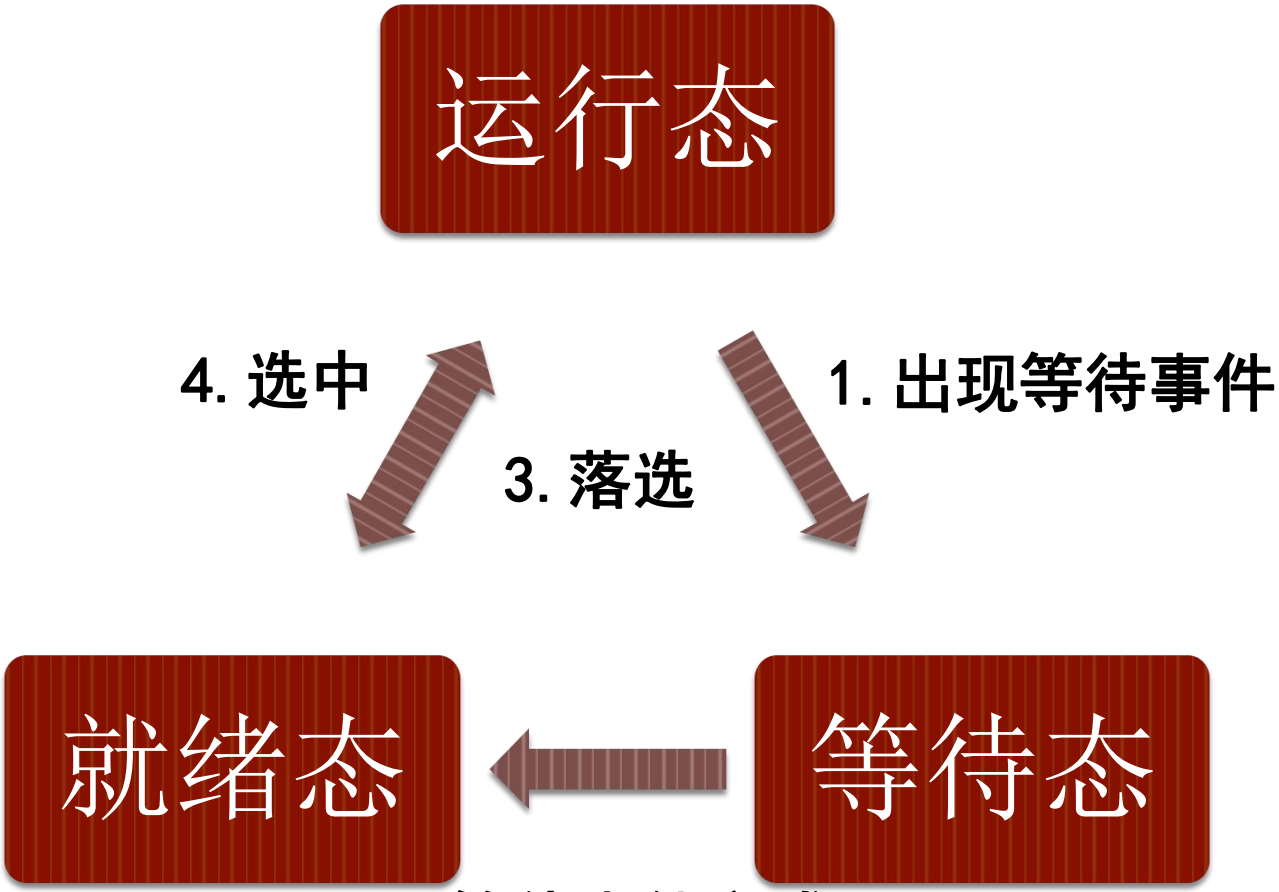
2. 五态模型

3. 七态模型
   - 具有挂起功能的进程状态及转换

2.3.2：三态模型

- 一个进程从创建而产生至撤销而消亡的整个生命周期，可用一组状态加以刻划，按进程在执行过程中的状况至少定义三种不同的进程状态：
  - 运行态（running）：进程占有处理器正在运行
  - 就绪态（ready）：进程具备运行条件，等待系统分配处理器以便运行（进程已经获得了除cpu以外的所有资源后处的状态）
  - 等待态（wait）：又称为阻塞（blocked）态或睡眠（sleep）态，进程不具备运行条件，正在等待某个事件的完成

2.3.2：三态模型（续）



2.3.2：三态模型（续）

- 通常，当一个进程创建后，就处于就绪状态
- 每个进程在执行过程中，任一时刻当且仅当处于上述三种状态之一
- 在一个进程执行过程中，它的状态将会发生变化

2.3.2：三态模型（续）：转换原因

1. 运行态→等待态：等待使用资源或某事件发生
2. 等待态→就绪态：资源得到满足或事件发生
3. 运行态→就绪态：运行时间片到；出现有更高优先权进程
4. 就绪态→运行态：CPU空闲时选择一个就绪进程

2.3.2：三态模型（续）：注意！

- 进程之间的状态转换并非都是可逆的
- 进程的转换并非都是主动的，只有运行到阻塞是主动行为，其它的都是被动的
- 一个具体进程在任一时刻必须且只能具有上述诸进程中的某一个状态
- 当进程处于运行态时它是微观意义下的运行，而不管进程处于何种状态，它都是宏观意义下的运行，只要它是处于已经开始执行和尚未结束执行的过程中

回顾

- 1、什么叫中断优先级？
- 2、中断优先顺序的设计原则？
- 3、什么叫中断屏蔽？有什么好处
- 4、多重中断的三种处理方式？
- 5、什么叫可再入程序？
- 6、什么叫可再用程序？
- 7、进程的定义？
- 8、进程和程序的区别？
- 9、进程的五大特性？
- 10、请画出进程的三态模型

2.3.2：五态模型：新建态和终止态

- 新建态：对应进程刚被创建的状态
  - 为一个新进程创建必要的管理信息，它并没有被提交执行，而是在等待操作系统完成创建进程的必要操作
- 终止态：进程的终止
  - 首先，等待操作系统进行善后处理
  - 然后，退出主存
  - 进入终止态的进程不再执行，但依然临时保留在系统中等待善后
  - 一旦其他进程完成了对终止态进程的信息抽取之后，系统将删除该进程

2.3.2：五态模型（续）

- 一个新的用户登陆分时系统或者一个新的批处理作业被提交执行，操作系统分两步定义新进程：
  1. 首先，OS执行一些必需的辅助工作，将标识号关联到进程。当进程处于新建状态时，OS所需要的关于该进程的信息（PCB）保存在主存中的进程表中，但进程本身还没有进入主存
  2. 如果系统中有足够的内存资源时，或者系统需要提交一个进程时，处于新建态的进程被提交进入就绪态，代码加载到内存

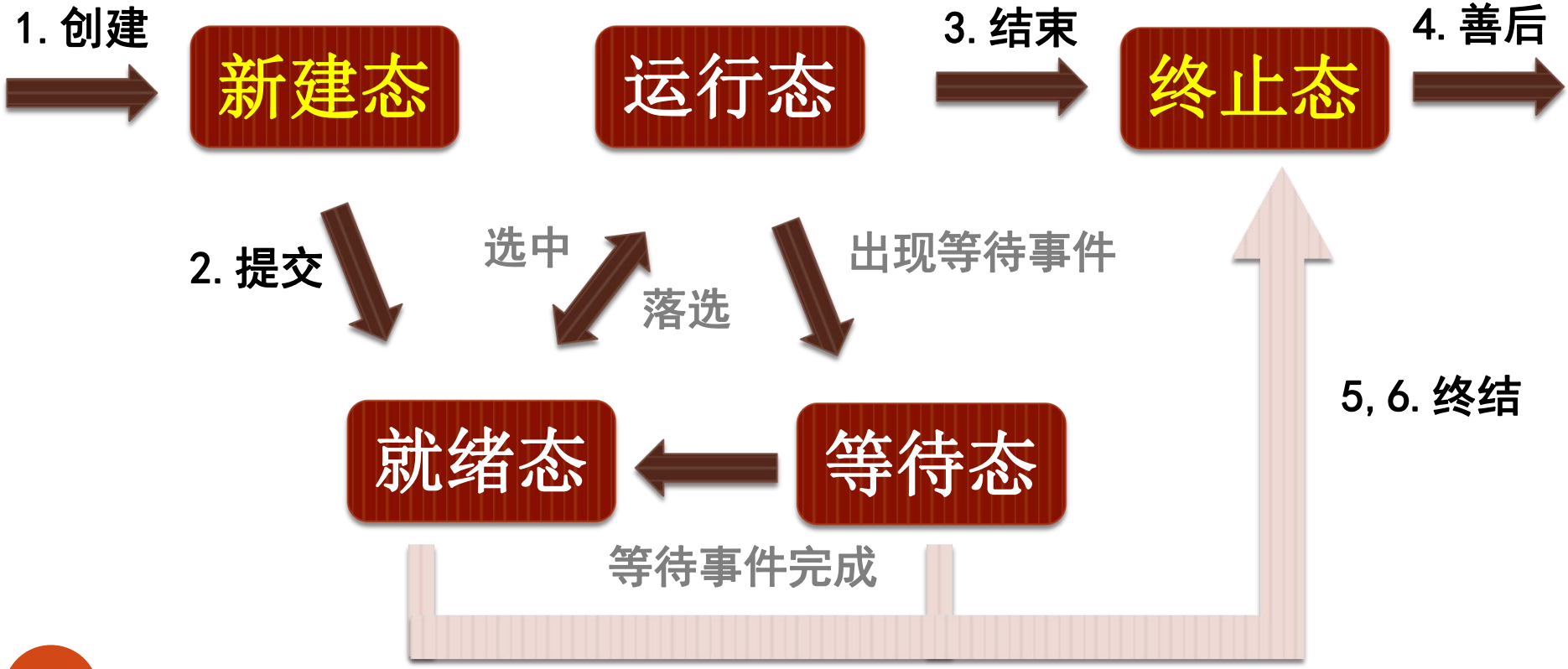
2.3.2：五态模型（续）

- 一个进程退出系统也分为两步：
  1. 首先，A. 当进程到达一个自然结束点时，B. 或者出现不可恢复的错误而被取消，C. 或当具有相应权限的另一个进程引发该进程取消时，进程被终止，终止使进程转换到退出状态，进程不再适合被执行
  2. 然后由辅助程序或支持程序提取所需信息
     - E. g. 记帐程序记录处理器时间和其他资源的使用情况
     - E. g. 实用程序为了分析性能和利用率，提取进程的历史信息
     - 一旦信息提取完毕，OS回收占用资源，并将进程从系统中删除

2.3.2：五态模型（续）：转换原因

1. NULL→新建态：创建一个进程
2. 新建态→就绪态：系统完成了进程创建操作，且当前系统的性能和内存的容量均允许
3. 运行态→终止态：一个进程到达自然结束点，或出现了无法克服的错误，或被操作系统所终结，或被其他有终止权的进程所终结
4. 终止态→NULL：完成善后操作
5. 就绪态→终止态：某些操作系统允许父进程终结子进程
6. 等待态→终止态：某些操作系统允许父进程终结子进程

2.3.2：五态模型（续）



2.3.2：七态模型：挂起状态

1) 为什么要有“挂起”状态?

- 由于进程的不断创建，系统资源已不能满足进程运行的要求，就必须把某些进程挂起（suspend），对换到磁盘镜像区中，暂时不参与进程调度，起到平滑系统操作负荷的目的

2.3.2：七态模型（续）

2) 引起进程挂起的主要原因

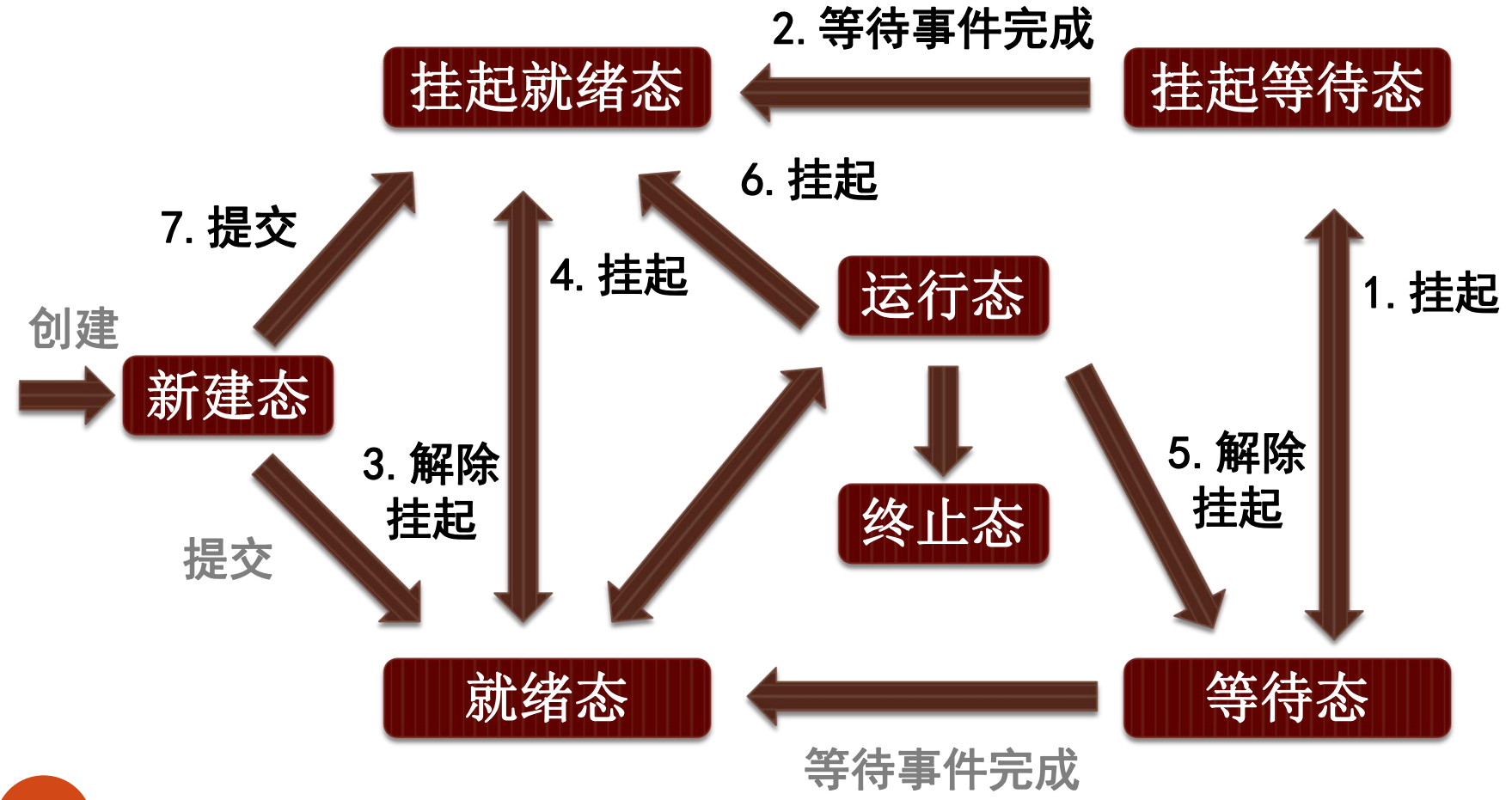
- 系统中的进程均处于等待状态，需要把一些阻塞进程对换出去，腾出足够内存装入就绪进程运行
- 进程竞争资源，导致系统资源不足，负荷过重，需要挂起部分进程以调整系统负荷，保证系统的实时性或让系统正常运行
- 定期执行的进程（如审计、监控、记账程序）对换出去，以减轻系统负荷
- 用户要求挂起自己的进程，以便进行某些调试、检查和改正
- 父进程要求挂起后代进程，以进行某些检查和改正
- 操作系统需要挂起某些进程，检查运行中资源使用情况，以改善系统性能
- 当系统出现故障或某些功能受到破坏时，需要挂起某些进程以排除故障

2.3.2：七态模型（续）

3) 进程增加的两个新状态

- 挂起就绪态（ready suspend）：表明进程具备运行条件但目前在辅助存储器中，当它被对换到主存才能被调度执行
- 挂起等待态（blocked suspend）：表明进程正在等待某一个事件且在辅助存储器中

2.3.2：七态模型（续）


2. 等待事件完成
挂起就绪态
挂起等待态
6. 挂起
7. 提交
4. 挂起
1. 挂起
创建
运行态
新建态
3. 解除
挂起
5. 解除
挂起
终止态
提交
就绪态
等待态
等待事件完成

2.3.2：七态模型（续）：转换原因

1. 等待态→挂起等待态：当前不存在就绪进程，至少一个等待态进程将被对换出去成为挂起等待态
2. 挂起等待态→挂起就绪态：引起进程等待的事件发生之后，相应的挂起等待态进程将转换为挂起就绪态
3. 挂起就绪态→就绪态：内存中没有就绪态进程，或挂起就绪态进程具有比就绪态进程更高的优先级，将把挂起就绪态进程转换成就绪态
4. 就绪态→挂起就绪态：系统根据当前资源状况和性能要求，决定把就绪态进程对换出去成为挂起就绪态

2.3.2：七态模型（续）：转换原因

5. 挂起等待态→等待态：当一个进程等待一个事件时，原则上不需要把它调入内存。但是，当一个进程退出后，主存已经有了足够的自由空间，而某个挂起等待态进程具有较高的优先级并且操作系统已经得知导致它阻塞的事件即将结束，便可能发生这一状态变化
6. 运行态→挂起就绪态：当一个高优先级挂起等待进程的等待事件结束后，它将抢占CPU，而此时主存不够，从而可能导致正在运行的进程转化为挂起就绪态。运行态的进程也可以自己挂起自己
7. 新建态→挂起就绪态：根据系统当前资源状况和性能要求，可以将新建进程对换出去成为挂起就绪态

2.3.2：七态模型（续）：转换原因

5. 挂起等待态→等待态：当一个进程等待一个事件时，原则上不需要把它调入内存。但是，当一个进程退出后，主存已经有了足够的自由空间，而某个挂起等待态进程具有较高的优先级并且操作系统已经得知导致它阻塞的事件[illegible]这一状态变化
6. 运行态→挂起[illegible]级挂起等待进程的等待事件[illegible]而此时主存不够，从而可能导致正在运行的进程转化为挂起就绪态。运行态的进程也可以自己挂起自己
7. 新建态→挂起就绪态：根据系统当前资源状况和性能要求，可以将新建进程对换出去成为挂起就绪态

挂起的进程将不参与**低级调度**直到它们被对换进主存；P94

2.3.2：七态模型（续）

- 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

- 进程的定义和属性
- 进程的状态和转换
- 进程的描述和组成
- 进程切换与模式切换
- 进程的控制和管理

2.3.3 进程的描述和组成

1. 进程映像（process image）
2. 进程控制块（process control block, PCB）
3. 进程队列及其管理

2.3.3：进程映像

- 某时刻进程的内容及其状态集合称为进程映像，主要包括：
  - 进程程序块：即被执行的程序，规定了进程一次运行应完成的功能。通常它是纯代码，可被多个进程共享
  - 进程数据块：即程序运行时加工处理的对象，包括全局变量、局部变量和常量等的存放区以及开辟的工作区，常常为一个进程专用
  - 核心栈：每一个进程都将捆绑一个系统/用户堆栈，用来保存中断/异常现场，保存在函数调用的参数和返回地址

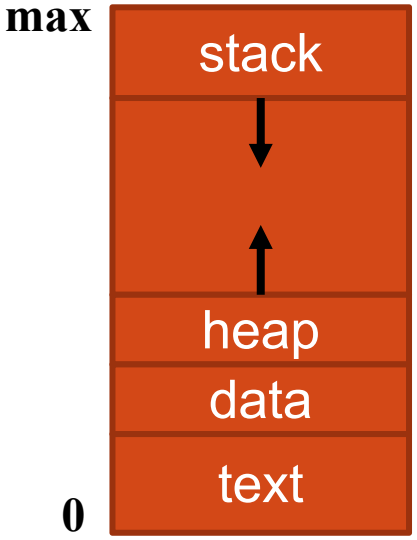


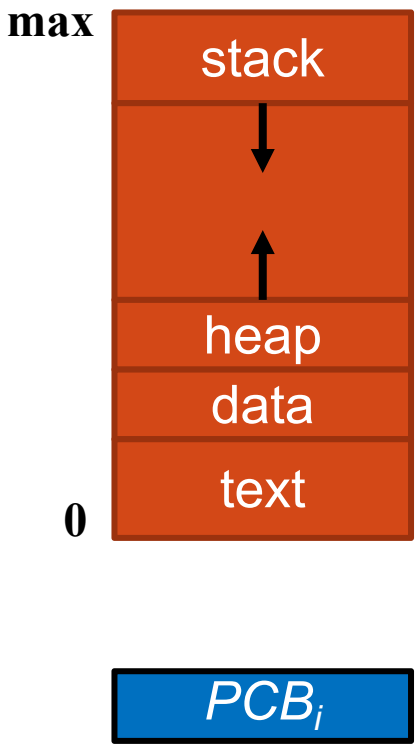
Figure 3.1
Process in memory
"OSC" P82

2.3.3：进程映像（续）

- 系统栈是内存中属于操作系统空间的一块固定区域，其主要用途为：
  1) 保存中断现场，对于嵌套中断，被中断程序的现场信息依次压入系统栈，中断返回时逆序弹出；
  2) 保存操作系统子程序间相互调用的参数、返回值、返回点以及子程序(函数)的局部变量
- 用户栈是用户进程空间中的一块区域，用于保存用户进程的子程序间相互调用的参数、返回值、返回点以及子程序(函数)的局部变量

2.3.3：进程映像（续）

- 进程内存映像的四大基本要素：
  - 程序块
  - 堆栈块
  - 数据块
  - 进程控制块
    - PCB: Process Control Block
    - 进程存在的唯一标识
    - 存储于操作系统空间内



2.3.3：进程上下文

- 进程物理实体和支持进程运行的环境被合称为进程上下文（process context）
  - 用户级上下文
  - 系统级上下文
  - 寄存器上下文
- 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

2.3.3：进程上下文（续）

- 进程上下文的三个组成部分
  1. 用户级上下文(user-level-context)：由用户进程的程序块、用户数据块（含共享数据块）和用户堆栈组成的进程地址空间
  2. 系统级上下文(system-level-context)：包括进程控制块（PCB）、内存管理信息、进程环境块，及系统堆栈等组成的进程地址空间
  3. 寄存器上下文(register-level-context)：由程序状态字（PSW）寄存器和各类控制寄存器、地址寄存器、通用寄存器、用户栈指针等组成

2.3.3：进程上下文（续）

- 如何理解进程映像和进程上下文的关系？
  - 进程映像用于表示进程状态
  - 进程上下文用于表示进程运行环境

2.3.3：进程控制块

- PCB 标志着进程的存在，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，是操作系统控制和管理进程的主要依据
- PCB 主要包含了进程执行时的情况，以及进程让出处理器后所处的状态、断点等信息
- 进程创建时建立进程控制块，进程撤销时回收进程控制块，进程控制块与进程一一对应

2.3.3：进程控制块（续）

- PCB 主要包含三类信息：
  1. 进程的标志信息
  2. 现场信息
  3. 控制信息

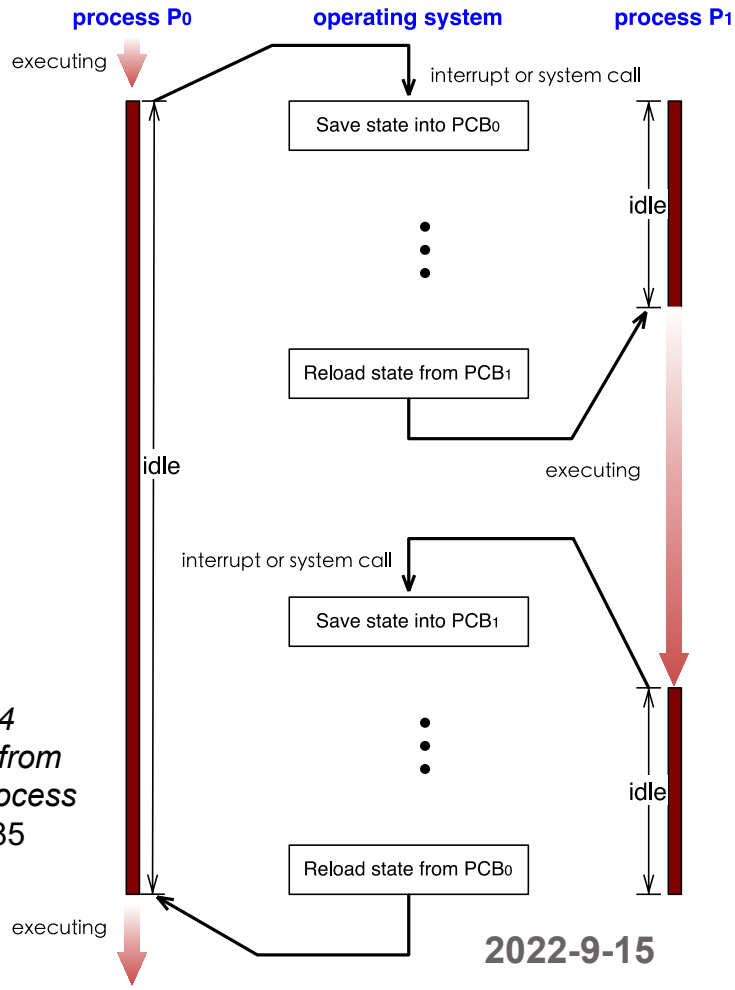


Figure 3.4 CPU switch from process to process "OSC" P85

2.3.3：进程控制块（续）

1. 标识信息
   - 用于唯一地标识一个进程，常常分为由用户使用的外部标识符和被系统使用的内部标识号
   - 几乎所有操作系统中进程都被赋予一个唯一的、内部使用的数值型的进程号，操作系统的其他控制表可以通过进程号来交叉引用进程控制表
   - 常用的标识信息有进程标识符、父进程的标识符、用户进程名、用户组名等

2.3.3：进程控制块（续）

1. 标识信息
   - 用于唯一地标识一个进程，常常分为由用户使用的外部标识符和被系统使用的内部标识号
   - 几乎所有操作系统中进程都被赋予一个唯一的、内部使用的数值型的进程号，操作系统的其他控制表可以通过进程号来交叉引用进程控制表
   - 常用的标识信息有进程标识符、父进程的标识符、用户进程名、用户组名等

2.3.3：进程控制块（续）

2. 现场信息
   - 用于保留进程运行时存放在处理器现场中的各种信息，进程让出处理器时必须把处理器现场信息保存到PCB中，当该进程重新恢复运行时也应恢复处理器现场
   - 现场信息包括：通用寄存器内容、控制寄存器（如PSW寄存器的）内容、用户堆栈指针、系统堆栈指针等

2.3.3：进程控制块（续）

3. 控制信息：常用的控制信息包括：
   i. 进程调度相关信息，如进程状态、等待事件和等待原因、进程优先级、队列指引元等
   ii. 进程组成信息，如正文段指针、数据段指针
   iii. 进程间通信相关信息，如消息队列指针、信号量等互斥和同步机制
   iv. 进程在二级（辅助）存储器内的地址信息
   v. CPU资源的占用和使用信息，如时间片余量、进程已占用CPU的时间、进程已执行时间总和，记帐信息
   vi. 进程特权信息，如在内存访问权限和处理器状态方面的特权
   vii. 资源清单，包括进程所需全部资源、已经分得的资源，如主存资源、I/O设备、打开文件表等

2.3.3：进程控制块（总结）

- 进程控制块的集合事实上定义了一个操作系统的当前状态
- 进程控制块的使用权或修改权均属于操作系统程序，包括：
  - 调度程序
  - 资源分配程序
  - 中断处理程序
  - 性能监视
  - 分析程序等
- 操作系统是根据PCB来对并发执行的进程进行控制和管理的

2.3.3：进程队列及其管理

- 将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(Process Queues)
  - 同一状态进程(就绪、等待、运行等)的PCB既可按先来先到FIFO的原则排成队列
  - 也可按优先数或其它原则排成队列
  - 等待态进程队列可以进一步细分，每一个进程按等待原因进入相应的等待队列

2.3.3：进程队列（续）

- 三种通用的队列组织方式：
  1. 线性方式
  2. 链接方式
  3. 索引方式

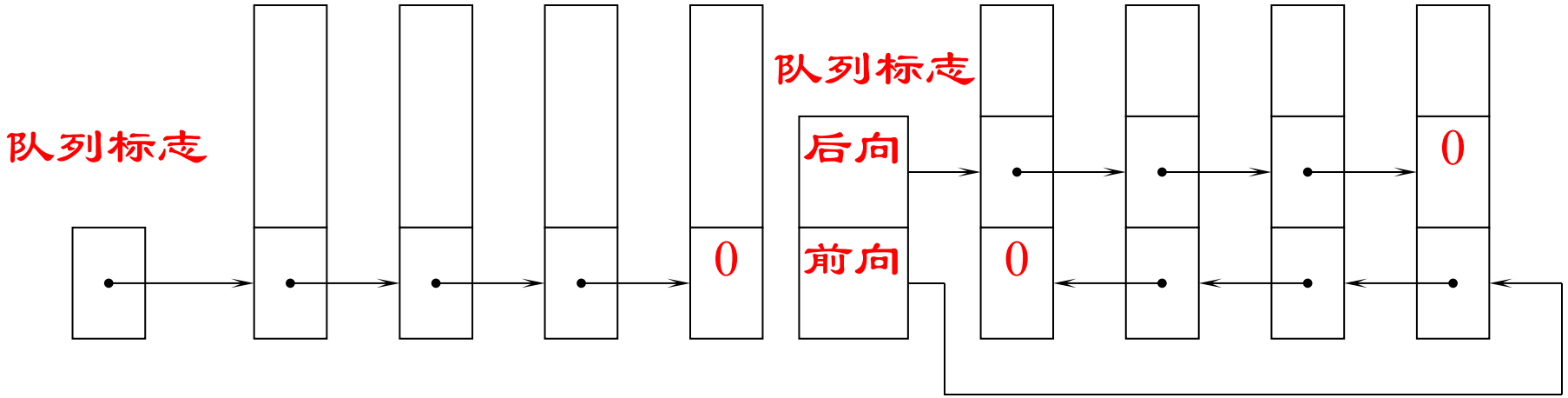
2.3.3：进程队列（续）：线性

- 操作系统根据系统内最大的进程数目，静态地分配主存中的某块空间，所有的进程的PCB都组织在一个线性表中
- 优点
  - 简单易行
- 缺点
  - 限定系统中的进程最大数
  - 经常要扫描整个线性表
  - 调度效率低

2.3.3：进程队列（续）：链接

- 对于同一状态的PCB，可以通过链接指针将其链接成队列
  - 单向链接
  - 双向链接

2.3.3：进程队列（续）：链接

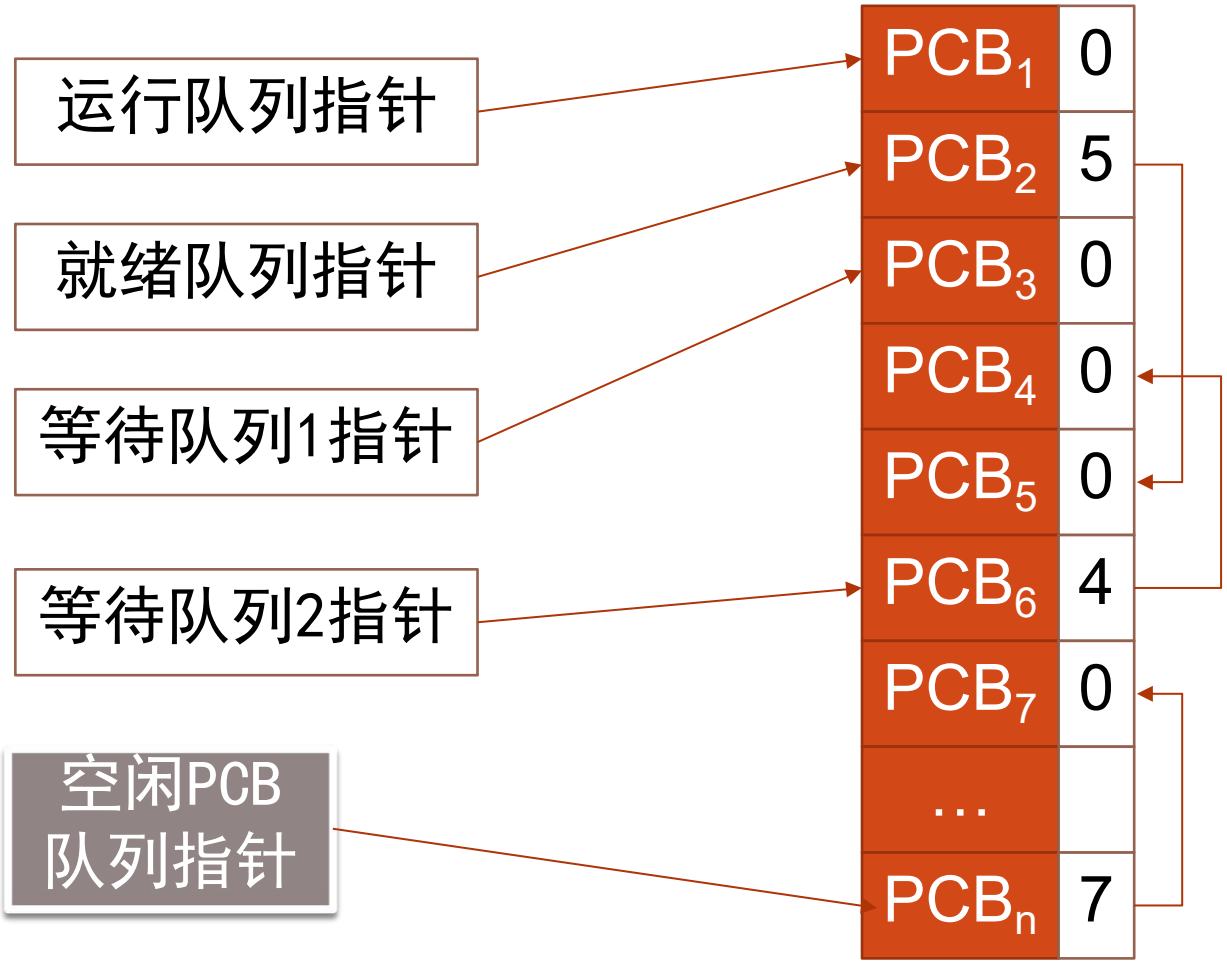


（a）单向连接　　（b）双向连接

→ 是队列指引元

进程控制块的链接

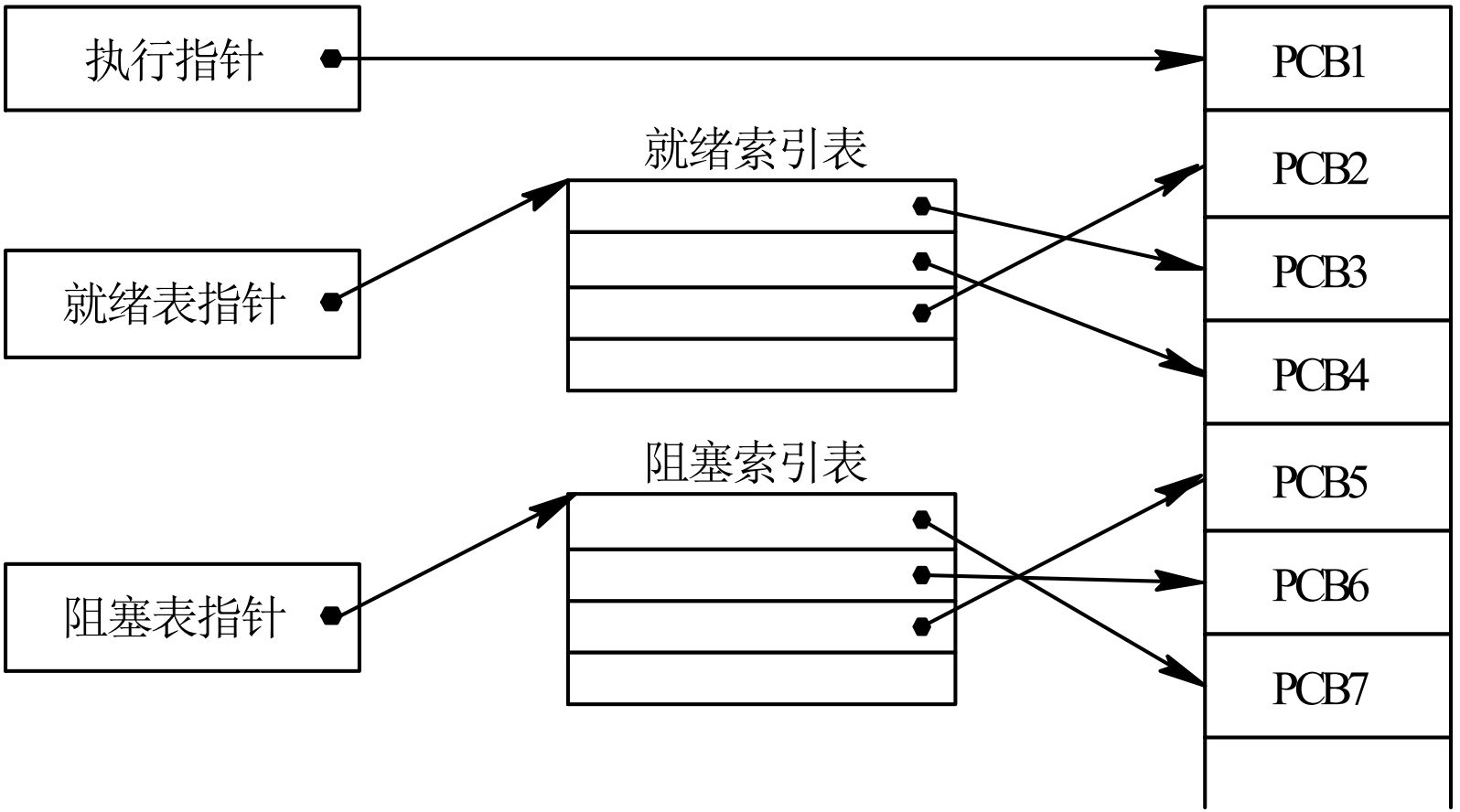
2.3.3：进程队列（续）：链接


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PCB
队列指针
PCB1 0
PCB2 5
PCB3 0
PCB4 0
PCB5 0
PCB6 4
PCB7 0
…
PCBn 7

2.3.3：进程队列（续）：索引

- 索引方式：利用索引表记录不同状态进程的PCB地址
- 系统建立若干索引表
  - 就绪索引表
  - 等待索引表
  - 空闲索引表

2.3.3：进程队列（续）：索引


执行指针
就绪表指针
阻塞表指针
就绪索引表
阻塞索引表
PCB1
PCB2
PCB3
PCB4
PCB5
PCB6
PCB7

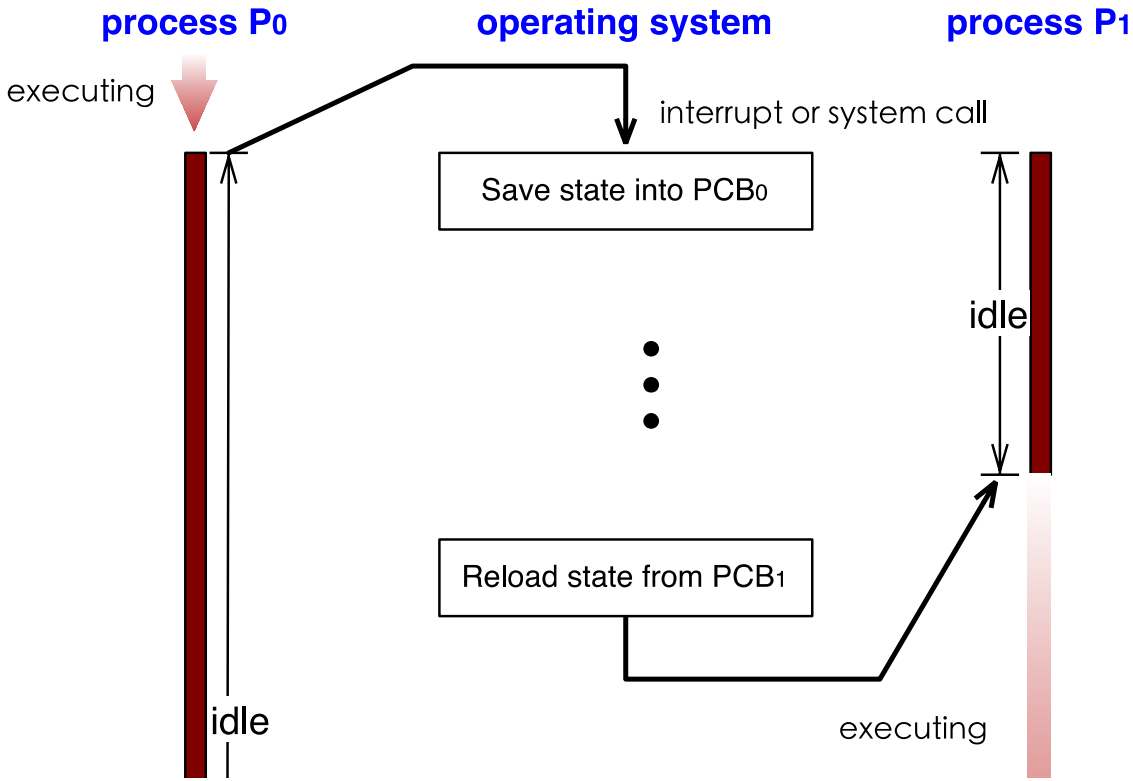
- 进程的定义和属性
- 进程的状态和转换
- 进程的描述和组成
- 进程切换与模式切换
- 进程的控制和管理

2.3.4 进程切换与模式切换

1. 进程上下文切换
2. 进程上下文切换时机
3. 处理器模式切换
4. UNIX/Linux中的进程切换与模式切换

2.3.4：进程上下文切换

进程切换是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、即保存老进程状态而装入被保护了的新进程的状态，以便新进程运行



进程切换必定在核心态

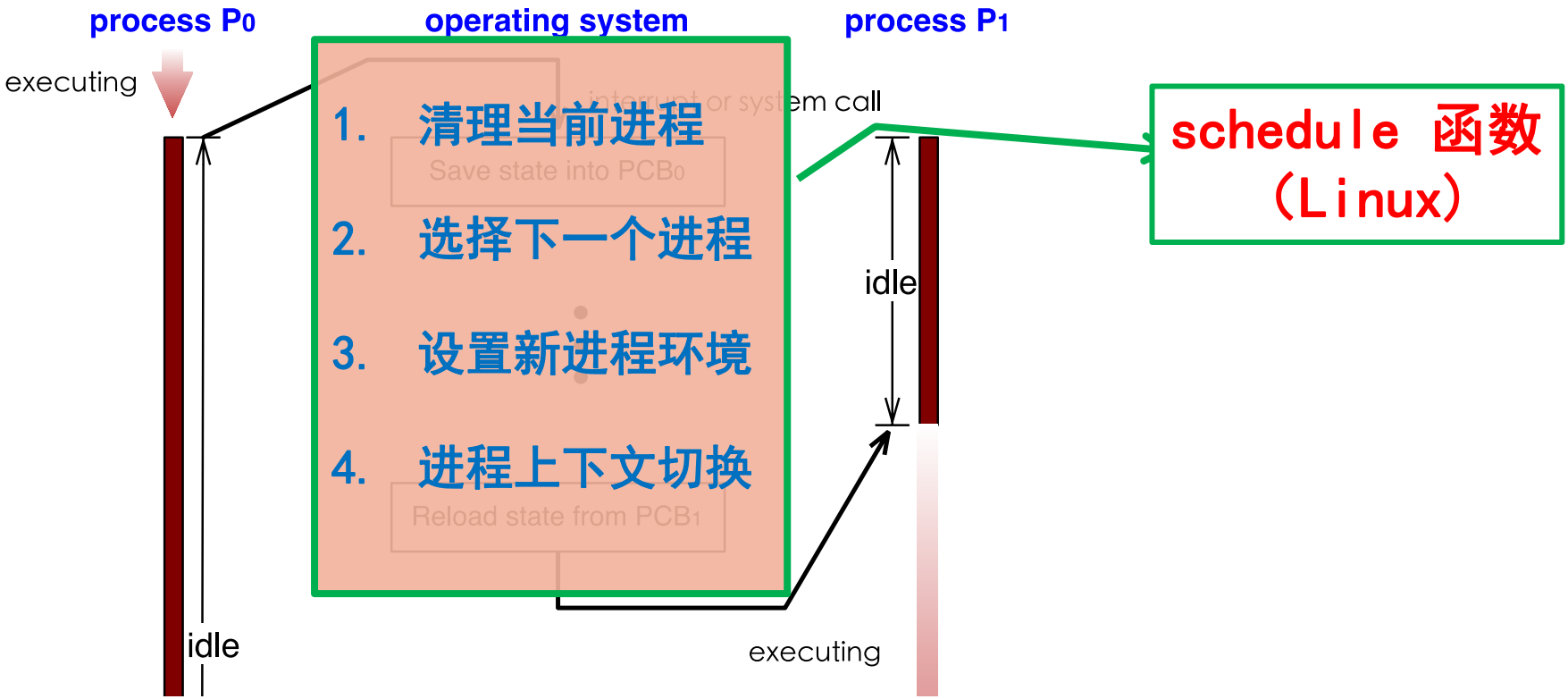
2.3.4：进程上下文切换（续）

- 引起进程切换的情况
  1. 当进程因各种原因进入等待态时
  2. 当进程完成其系统调用返回用户态，但尚无资格获得CPU时
  3. 当内核完成中断处理，进程返回用户态但尚无资格获得CPU时
  4. 当进程执行的时间片到时或进程结束时

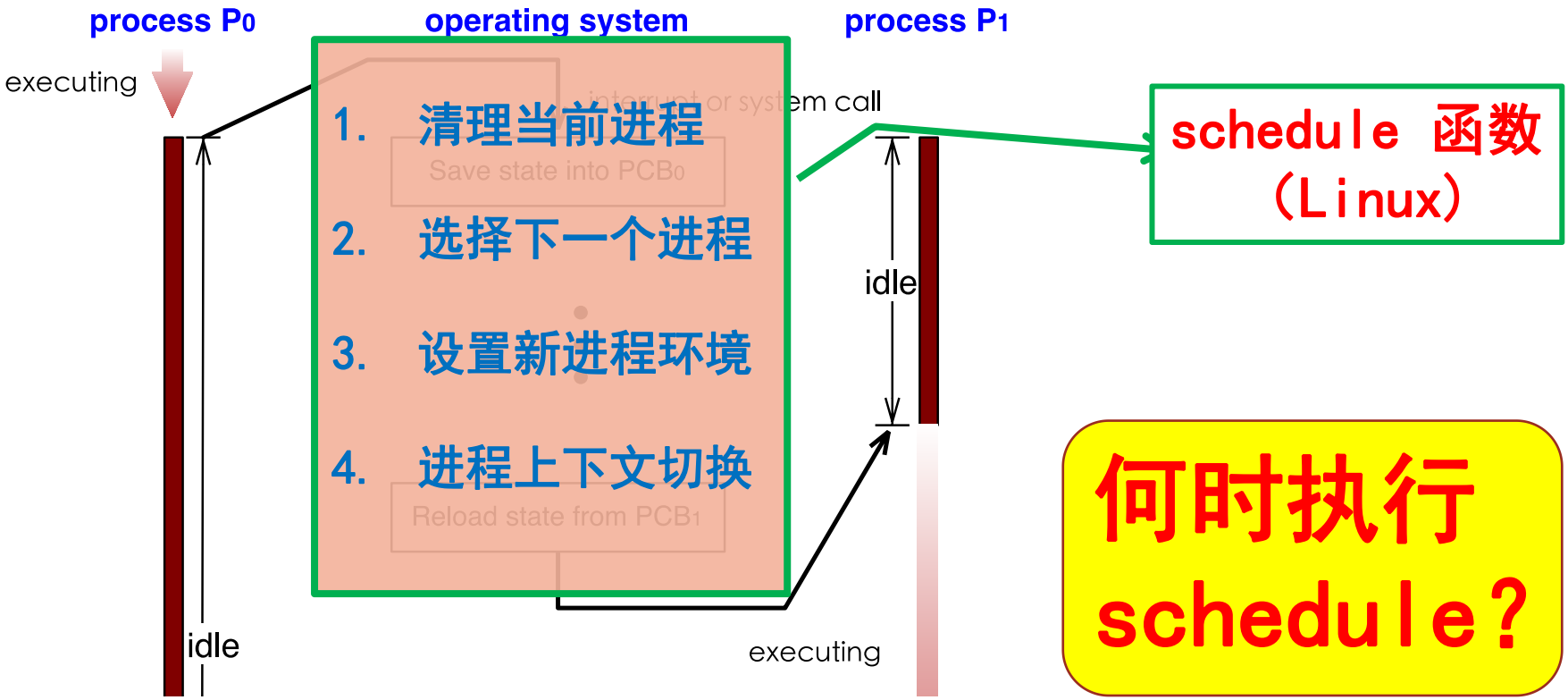
2.3.4：进程上下文切换（续）

- 进程切换的实现步骤：
  1. 保存被中断进程的处理器现场信息
  2. 修改被中断进程PCB的有关信息，如进程状态等
  3. 把被中断进程的PCB加入相关队列
  4. 选择占有处理器运行的另一个进程
  5. 修改被选中进程的PCB的有关信息，如改为就绪态
  6. 修改被选中进程的地址空间，恢复存储管理信息
  7. 设置被选中进程的上下文信息来恢复处理器现场

2.3.4：进程上下文切换时机



2.3.4：进程上下文切换时机（续）



2.3.4：进程上下文切换时机（续）

主动调度
（Linux 2.4）

被动调度
（Linux 2.6）

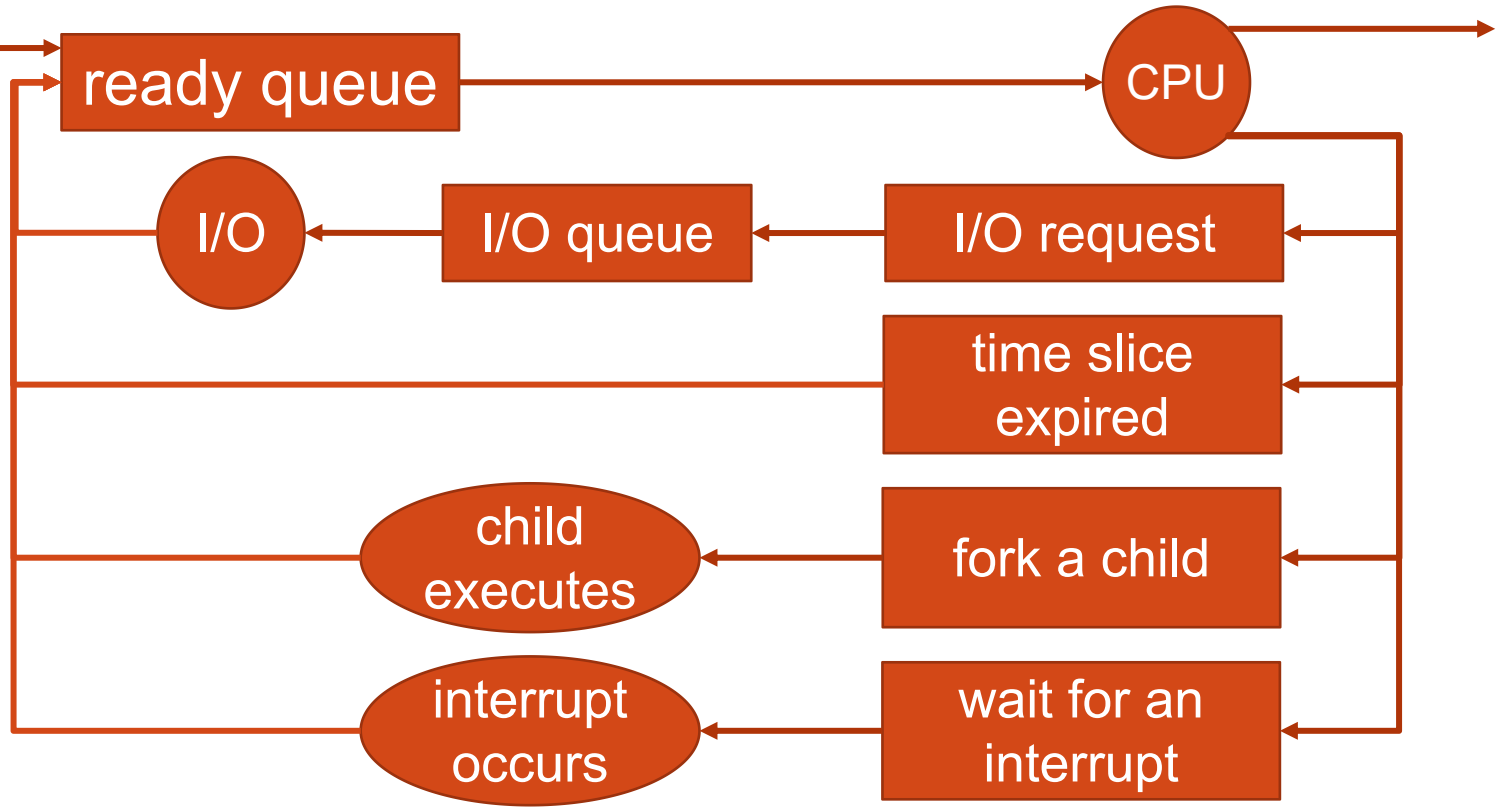


Figure 3.7
Queueing-diagram representation of process scheduling
"OSC" P88

2.3.4：进程上下文切换时机（续）

- 不能立即进行调度和切换的情况有：
  - 在运行内核中断处理程序期间
  - 若内核正在系统的临界区执行
  - 在各种原语操作、现场保护和恢复等过程中

2.3.4：处理器模式切换

- 当前进程从用户态到核心态或者从核心态到用户态的转换称为模式切换
- 当发生中断或者系统调用时，暂停正在运行的进程，把处理器从用户态切换到核心态，执行系统服务程序，这就是一次模式切换
- 发生模式切换时，进程仍在自己的上下文中执行
- 内核在被中断进程的上下文中进行处理

2.3.4：处理器模式切换（续）

- 模式切换步骤：
  1. 保存被中断进程的处理器现场信息
  2. 处理器从用户态切换到核心态，以便执行系统服务程序或中断处理程序
  3. 如果处理中断，可根据所规定的中断级别设置中断屏蔽位
  4. 根据系统调用号或中断号，从系统调用表或中断入口地址标中找到系统服务程序或中断处理程序的地址

2.3.4：处理器模式切换（续）

- CPU上所执行程序在任何时刻必定处于三个活动范围：
  1. 进程正常运行：在用户空间中，处于进程上下文，用户进程在运行，使用用户栈
  2. 进程运行系统调用：在内核空间中，处于进程上下文，内核代表某进程在运行，使用核心栈（模式切换）
  3. 发生中断：内核空间中，处于中断上下文，与进程无关，中断服务程序正在处理特定的中断（模式切换）

2.3.4：UNIX下进程与模式切换

- 系统进程和用户进程，这两种进程并不是指两个具体的进程实体，而是指一个进程的两个侧面：
  - 系统进程：是在核心态下执行操作系统代码的进程，系统进程的地址空间中包含所有的系统核心程序和各进程的进程数据区，各进程的系统程序除数据区不同外，其余部分完全相同
  - 用户进程：用户进程是在用户态下执行用户程序的进程。各进程的用户进程部分各不相同

2.3.4：UNIX下进程与模式切换（续）

- 简单的说就是把程序分成两类来运行（进程），一类级别高的一类级别低的
  - 高级别的是系统进程，低级别的是用户进程
  - 用户进程运行在虚拟机上，系统进程运行在物理机上
    - driver 运行在 kernel mode
    - application应用程序,普通的service 运行在 user mode
  - kernel 本身还有很多级别

回顾

- 1、什么叫进程映像？包括哪些内容？
- 2、什么叫进程控制块？包括哪些内容？
- 3、什么叫进程上下文？
- 4、什么叫进程队列？包括哪几种？
- 5、什么叫进程切换？
- 6、进程切换的实现步骤？
- 7、什么叫处理机模式切换？
- 8、处理机模式切换步骤?
- 9、请画出进程的七态模型？

- 进程的定义和属性
- 进程的状态和转换
- 进程的描述和组成
- 进程切换与模式切换
- 进程的控制和管理

2.3.5 进程的控制和管理

- 处理器管理的一个主要工作是对进程的控制
- 这些控制和管理功能是由操作系统中的原语（primitive action）来实现的
  - 在管态下执行、完成系统特定功能的过程

2.3.5：原语

- 原语是机器指令的延伸，往往是为完成某些特定的功能而编制的一段系统程序。为保证操作的正确性，在许多机器中规定，执行原语操作时，要屏蔽中断，以保证其操作的不可分割性
- 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
- 原语的实现方法是以系统调用方式提供原语接口，且采用屏蔽中断的方式来实现原语功能，以保证原语操作不被打断的特性

2.3.5：原语 vs. 系统调用

- 调用形式相同
  - 原语和系统调用都使用访管指令实现
- 实现方式不同
  - 原语由内核实现，系统调用通常由系统进程或系统服务器实现

2.3.5：原语 vs. 系统调用

- 运行方式不同
  - 原语不可中断，系统调用执行时允许被中断，甚至有些操作系统中系统进程或系统服务器在用户态运行
- 服务的对象不同
  - 通常情况下，原语提供给系统进程或系统服务器使用（反之决不会形成调用关系），系统进程或系统服务器提供系统调用给系统程序（和用户）使用，系统程序提供高层功能给用户使用

2.3.5：进程控制管理原语

- 进程创建
- 进程撤销
- 进程阻塞与唤醒
- 进程挂起和激活

2.3.5：进程创建:事件来源

- 提交一个批处理作业
- 在终端上一个交互式作业登录
- 操作系统创建一个服务进程
- 存在的进程孵化（spawn）新的进程

生成其它进程的进程称**父进程**(Parent Process)，被生成进程称**子进程**(Child Process)、即一个父进程可以创建子进程，从而形成树形结构

2.3.5：进程创建（续）：进程树

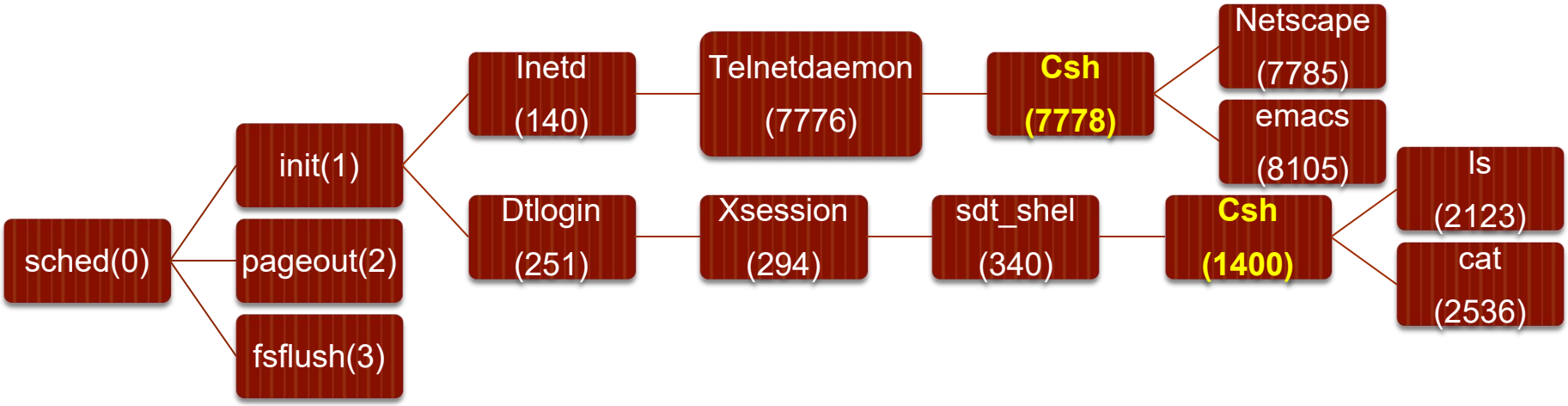


Figure 3.9
*A tree of processes on a typical **Solaris** system*
"OSC" P91

2.3.5：进程创建（续）：步骤

1. 在主进程表中增加一项，并从PCB池中取一个空白PCB
2. 为新进程的进程映像分配地址空间。传递环境变量，构造共享地址空间
3. 为新进程分配资源，除内存空间外，还有其他各种资源
4. 初始化进程控制块（如状态、PSW、栈等），为新进程分配进程标识符
5. 把进程加入就绪进程队列，或直接将进程投入运行
6. 通知操作系统的某些模块，如性能监控程序

2.3.5：进程创建（续）：fork()

输入：无
输出：对父进程是子进程的 PID 对子进程是0
{ 检查可用的核心资源
  取一个空闲的进程表项和唯一的 PID 号
  检查用户没有过多的运行进程
  将子进程的状态设置为“创建”状态
  将父进程的进程表中的数据拷贝到子进程表中
  当前目录的索引节点和改变的根目录(如果可以)的引用数加1
  文件表中的打开文件的引用数加1
  在内存中作父进程上下文的拷贝
  在子进程的系统级上下文中压入虚设系统级上下文层
  if (正在执行的进程是父进程)
   { 将子进程的状态设置为“就绪”状态
     return (子进程的 PID)
   }
  else { 初始化计时区 return 0; }
 }

2.3.5：进程创建（续）：fork()

- PCB：
  - 完全区分
- 代码段：
  - 父子进程共享 / 调用 execve() 替换新程序
- 数据、堆栈段：
  - 初始共享读
  - 某一进程写，则物理区分，以降低系统开销

2.3.5：进程创建（续）

- Linux fork( ) 创建子进程但父子进程不定义内容共享
- Linux clone( ) 允许定义父子进程共享的内容
- 操作系统最多进程数的限制
- UNIX(早期)最多创建几十个进程
  - Solaris可在启动时根据内存容量自动调整创建数
  - Linux2.4中，最多进程数是运行时可调参数，缺省设置为：size-of-memory-in-the-system/kernel-stack-size/2
  - 假如机器有512MB内存，缺省可创建进程的上限为：512×1024×1024/8192/2=32768

2.3.5：进程撤销

- 正常撤销
  - 分时系统中的注销
  - 批处理系统中的撤离作业步
- 非正常撤销
  - 程序运行过程中出现错误或异常

2.3.5：进程撤销：原因

- 进程运行结束
- 进程执行非法指令
- 进程在用户态执行特权指令
- 进程的运行时间超过所分配的最大时间配额
- 进程的等待时间超过所设定的最长等待时间
- 进程所申请的贮存空间超过系统所能提供的最大容量
- 越界错误
- 对共享主存区的非法使用
- 算术错误，如除数为0，操作数溢出
- 操作员或操作系统干预
- 父进程撤销子进程
- 父进程撤销，其所有子进程被撤销
- 操作系统终止

2.3.5：进程撤销：步骤

1. 根据撤销进程标识号，从相应队列中找到它的PCB
2. 将该进程拥有的资源归还给父进程或操作系统
3. 若该进程拥有子进程，应先撤销它的所有子孙进程，以防它们脱离控制
4. 撤销进程出队，将它的PCB归还到PCB池

2.3.5：进程阻塞和唤醒

- 进程阻塞是指一个进程让出处理器，去等待一个事件
- 通常，进程自己调用阻塞原语阻塞自己，所以，阻塞是自主行为，是一个同步事件。当一个等待事件结束时会产生一个中断，从而激活操作系统，在系统的控制之下将被阻塞的进程唤醒
- 进程的阻塞和唤醒是由进程切换来完成的

2.3.5：进程阻塞:原因

- 请求系统服务
- 启动某种操作
- 新数据未到达
- 无新工作可做
- 执行p操作没有满足

2.3.5：进程阻塞:步骤

主动行为

1. 停止进程执行，保存现场信息到PCB
2. 修改PCB的有关内容，如进程状态由运行改为等待等
3. 把修改状态后的PCB加入相应等待进程队列
4. 转入进程调度程序调度其他进程运行

2.3.5：进程唤醒:原因

- 当进程期待的事件发生时，有关进程便调用唤醒原语wakeup把等待该事件的进程唤醒
  - 资源具备
  - 数据到达
  - 休眠时间截止
  - …

2.3.5：进程唤醒:原因

被动行为

1. 从相应等待进程队列中取出PCB
2. 修改PCB有关信息，如进程状态改为就绪等
3. 把修改后PCB加入有关就绪进程队列

2.3.5：进程阻塞和唤醒（续）

- 在UNIX/Linux中，与进程的阻塞与唤醒相关的原语主要有：
  - sleep（暂停一段时间）：在指定的时间内阻塞本进程
  - pause（暂停并等信号）：阻塞本进程以等待信号
  - wait（等待子进程暂停或终止）：阻塞本进程以等待子进程结束，子进程结束时返回
  - kill（发信号）：可以发送信号到某个进程或某组进程
  - signal是发送的数字信号，9为杀掉进程，17是停止进程

2.3.5：进程挂起和激活:原因

- 挂起进程
  - 系统资源不足
  - 请求挂进指定进程（主动或被动）
- 激活进程
  - 主存资源充裕
  - 请求激活指定进程（被动，具需有相应权限）

2.3.5：进程挂起和激活:步骤

- 挂起进程
  - 检查要被挂起进程的状态，若处于活动就绪态就修改为挂起就绪，若处于阻塞态，则修改为挂起阻塞。被挂起进程PCB的非常驻部分要交换到磁盘对换区
- 激活进程
  - 把进程PCB非常驻部分调进内存，修改它的状态，挂起等待态改为等待态，挂起就绪态改为就绪态，排入相应队列中。