# 第 2 章

# 处理器管理

处理器是计算机系统中最重要的部件，计算机通过执行处理器指令对硬件及软件资源进行控制。

## 2.1 处 理 器

处理器用于执行指令对数据进行加工处理，处理器具有暂存数据的寄存器、加工数据的指令系统及指令执行权限控制机制。

### 2.1.1 寄存器

处理器属于时分复用型的共享资源，其中的寄存器被操作系统和各个进程所共享，任意瞬时的寄存器内容构成处理器工作现场。当进程或任务发生切换时，寄存器内容必须被保存，以便进程或任务恢复执行时，还原处理器工作现场。

根据寄存器的内容，寄存器分为数据类寄存器、地址类寄存器、控制类寄存器。数据类寄存器用于存放操作数或计算结果。地址类寄存器用于存放要访问的内存地址或设备 I/O 地址。控制类寄存器用于存放处理器的控制和状态信息。

Intel x86 处理器的寄存器主要如下。

（1）通用寄存器：EAX、EBX、ECX、EDX、EBP、ESP、ESI、EDI。

（2）段寄存器：CS、DS、SS、ES。

（3）标志寄存器：EFLAGS。

（4）指令指针寄存器：EIP。

（5）控制寄存器：CR0～CR3。

（6）系统地址寄存器：包括全局描述符表寄存器 GDTR、局部描述符表寄存器 LDTR、中断描述符表寄存器 IDTR、任务状态寄存器 TR。

### 2.1.2 指令系统、特权指令与非特权指令

#### 1．指令分类

指令大致分为如下 4 类。

（1）数据传输类指令：完成处理器与存储器之间、处理器与 I/O 设备之间的数据传送操作。

（2）数据处理类指令：完成运算器对数据的算术操作、逻辑操作等计算功能。

（3）控制类指令：这类指令可以改变指令执行顺序及控制资源的使用权限。

#### 2．特权指令与非特权指令

指令的执行涉及系统资源的操作，系统资源供多个用户程序共享，必须赋予操作系统高于普通用户的资源管理权限，以便维护资源的有序使用。根据指令执行主体为操作系统还是普通用户程序，指令划分为特权指令和非特权指令。

特权指令是仅提供给操作系统核心程序使用的指令，执行特权指令时，操作系统可以对整个系统进行控制。特权指令包括启动 I/O 设备、设置时钟、控制中断屏蔽位、清内存、建立存储键、加载 PSW（程序状态字）等。特权指令与共享资源的使用相关。共享资源在操作系统的调度下被各个用户进程有序共享，避免恶性竞争，导致系统崩溃。只有操作系统才能执行指令系统中的全部指令（包括特权指令和非特权指令），用户程序只能执行指令系统中的非特权指令。如果用户程序试图执行特权指令，则将产生保护性中断，转交给操作系统的“用户非法执行特权指令”的特殊处理程序处理。

### 2.1.3　处理器状态及切换

操作系统可以执行特权指令，而用户程序只能执行非特权指令。处理器如何知道当前执行的程序是操作系统程序还是普通用户程序，以便对程序所能执行的指令集加以限制呢？此时，需要在执行两种不同权限的程序时对处理器设置不同状态。

#### 1．处理器状态分类

处理器状态又称为处理器运行模式，一般把处理器状态简单划分为管理状态（特权状态、系统模式，简称特态或管态）和用户状态（目标状态、用户模式，简称目态或常态）。当处理器处于管理状态时，程序可以执行全部指令，访问所有资源，并具有改变处理器状态的能力；当处理器处于用户状态时，程序只能执行非特权指令。

#### 2．Intel Pentium 的处理器状态

Intel Pentium 的处理器状态有 4 种，支持 4 个保护级别，0 级权限最高，3 级权限最低。一般的，典型应用中的 4 个特权级别依次如下。

（1）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。

（2）1 级为系统调用处理程序级。用户程序调用这里的过程执行系统调用。

（3）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。

（4）3 级为用户程序级，受到的保护最少。

各个操作系统可以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。

#### 3．处理器状态之间的转换

1）用户状态向管理状态的转换

下面两种情况会导致处理器从用户状态转变为管理状态：一是执行系统调用，请求操

作系统服务；二是程序运行时，一个中断事件产生，运行程序被中断，操作系统接管处理器，中断处理程序开始工作。这两种情况都是通过中断机构发生的。中断是目态到管态转换的唯一途径。

2）管理状态向用户状态的转换

每台计算机通常会提供一条特权指令，称为加载程序状态字（Load PSW，LPSW），用来实现操作系统向用户程序的转换。

### 2.1.4 程序状态字寄存器

处理器的工作状态记录在程序状态字（Program Status Word，PSW）寄存器中，每个正在执行的程序都有一个与其执行相关的 PSW，而每个处理器都设置了一个程序状态字寄存器。程序状态字寄存器一般包括以下内容。

（1）程序的基本状态。

① 程序计数器：指明下一条执行的指令地址。

② 条件码：表示指令执行的结果状态。

③ 处理器状态位：指明当前的处理器状态，如目态或管态、运行或等待。

（2）中断码：保存程序执行时当前发生的中断事件。

（3）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。

大多数计算机的处理器现场中可能找不到一个称为程序状态字寄存器的具体寄存器，但总是有一组控制与状态寄存器实际上起到了这一作用。

在 Intel Pentium 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 位称为 FLAGS，标志可划分为 3 组：状态标志、控制标志、系统标志。

① 状态标志：使得一条指令的执行结果影响后面的指令，如溢出标志、符号标志、结果为零标志、辅助进位标志、进位标志、奇偶校验标志等。

② 控制标志：如串指令操作方向标志、虚拟 86 方式标志、步进标志、陷阱标志等。

③ 系统标志：与进程管理有关，如 I/O 特权级标志、嵌套任务标志和恢复标志等。

## 2.2 中　　断

中断是改变指令执行流程、实现操作系统并发多任务功能的重要硬件机构，也是操作系统实现计算机控制的重要途径。

### 2.2.1 中断概念

请求系统服务、实现并行工作、处理突发事件、满足实时要求，都需要利用中断机制打断处理器正常工作。

**中断**：指程序执行过程中，当发生某个事件时，中止 CPU 上现行程序的运行，引出处理该事件的程序执行的过程。

在提供中断装置的计算机系统中，在每两条指令或某些特殊指令执行期间都检查是否有中断事件发生，若无则立即执行下一条指令，否则响应该事件并转去处理中断事件。

中断源是引起中断的事件。中断装置是发现中断源并产生中断的硬件。中断机制的重要特征在于：当中断事件发生后，它能改变处理器内操作执行的顺序。因此，中断是现代操作系统实现并发性的基础之一。

## 2.2.2　中断源分类

### 1．按照中断事件的性质和激活的手段分类

从中断事件的性质和激活的手段可以把中断源分成两类：强迫性中断事件和自愿性中断事件。

1）强迫性中断事件

强迫性中断事件不是正在运行的程序所期待的，而是由于随机发生的某种事故或外部请求信号引起的。正在运行的程序不可预知强迫性中断事件发生的时机。这类中断事件大致有：

（1）机器故障中断事件，如电源故障、主存储器出错等。

（2）程序性中断事件，如定点溢出、除数为 0、地址越界等，又称异常。

（3）外部中断事件，如键盘中断、时钟定时中断等。

（4）输入输出中断事件，如设备出错、传输结束等。

2）自愿性中断事件

自愿性中断事件是正在运行的程序所期待的事件。这种事件由程序执行访管指令而引发，表示用户进程请求操作系统服务。

### 2．按照中断信号的来源对中断源分类

按照中断信号的来源，可把中断分为外中断和内中断。

1）外中断

外中断（又称中断）是指来自处理器和主存之外的中断。

外中断包括：电源故障中断、时钟中断、控制台中断、打印机中断和 I/O 中断等。

不同的中断具有不同的中断优先级，处理高一级中断时，往往会屏蔽部分或全部低级中断。

2）内中断

内中断（又称异常）是指来自处理器和主存内部的中断。

内中断包括：通路校验错、主存奇偶错、非法操作码、地址越界、页面失效、调试指令、访管中断、算术操作溢出等各种程序性中断。

内中断是不能被屏蔽的，一旦出现应立即响应并加以处理。

3）中断和异常的区别

中断是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，通常在两条机器指令之间才可响应中断，一般来说，中断处理程

序提供的服务不是当前进程所需的，如时钟中断。异常是由处理器正在执行现行指令而引起的，异常处理程序提供的服务是当前进程所需要的。

IBM 中大型机操作系统使用上述第一种分类方法，Windows 2000/XP 则采用上述第二种分类方法。

#### 3．硬中断与软中断

（1）硬中断：硬中断是由硬件设施产生的中断信号。

（2）软中断：软中断是利用软件模拟产生的中断信号，用于实现内核与进程或进程与进程之间的通信。

硬中断发生时会立刻响应。软中断发生时，如果接收中断信号的进程处于非运行状态，则软中断不会立即响应。

### 2.2.3 中断处理

所有计算机系统都采用硬件和软件（硬件中断装置和软件中断处理程序）结合的方法实现中断处理。硬件（即中断装置）发现中断源并产生中断信号，硬件包括中断逻辑线路和中断寄存器。

中断寄存器用来记录中断事件，中断寄存器的内容称为中断字，中断字的每一位对应一个中断事件。每当一条机器指令执行结束时，中断控制部件扫描中断字，查看是否有中断事件发生，若有则处理器响应此中断请求。

中断发生后，中断字的相应位会被置位。由于同一时刻可能有多个中断事件发生，中断装置将根据中断屏蔽要求和中断优先级选取一个，然后把中断寄存器的内容送入程序状态字寄存器的中断码字段，且把中断寄存器相应位清“0”。

#### 1.（硬件）中断装置

中断装置对中断做如下处理。

（1）发现中断源，响应中断请求。

（2）保护现场。将运行程序中断点在处理器中某些寄存器的现场信息（又称运行程序的执行上下文）存放于内存储器。

（3）启动处理中断事件的程序。

例如，在 IBM PC 上，设备 I/O 操作完成后中断装置的典型操作序列如下。

① 设备给处理器发送一个中断信号，处理器发现中断源。

② 处理器向设备发送确认信号，处理器响应中断请求。

③ 中断现行程序，保护现场：处理器保存当前程序断点信息到系统栈以备将来恢复执行还原现场，断点信息包括程序状态字（PSW）、程序计数器 IP（包含下一条要执行的指令地址）及段寄存器（CS）。

④ 启动中断处理程序：处理器根据硬件中断装置提供的中断向量号，获得被接收的中断请求的中断向量地址，再按照中断向量地址把中断处理程序的 PSW 送入现行程序状态字寄存器，加载新的程序状态字，将中断处理程序入口地址装入程序计数器 PC。

⑤ 中断返回：中断处理程序执行结束，立刻或者将来返回原程序时，把系统栈顶内容送入程序计数器 IP、CS 和 PSW。

#### 2.（软件）中断处理程序

处理中断事件的程序称为中断处理程序，它的主要任务是处理中断事件和恢复正常操作。

中断处理程序由硬件中断装置激活后继续进行中断处理，主要完成以下 4 项工作。

（1）保护未被硬件保护的一些必需的处理器状态，如保存通用寄存器的内容到主存储器中。

（2）识别各个中断源，分析产生中断的原因。

（3）处理发生的中断事件。

（4）中断返回。恢复中断前的程序，使其从断点执行或者重新启动一个新的程序，甚至可以重新启动操作系统。

#### 3. 中断处理程序入口地址的寻找

不同中断源对应不同的中断处理程序，寻找中断处理程序入口地址的方法如下：在主存储器中设置一张向量地址表，存储单元的地址对应向量地址，存储单元的内容为入口地址。CPU 响应中断后，根据预先规定的次序找到相应的向量地址，便可获得该中断事件处理程序的入口地址。

## 2.3　进程及其实现

计算机工作起来后，其中存在各种活动，操作系统的重要任务之一就是对这些活动进行管理，进程是最基本的活动单位。

### 2.3.1　引入进程概念的必要性

进程与程序是极其容易混淆的两个概念。两者虽然都包含程序，但是程序是静态的，进程是动态的，进程是程序执行时的动态过程。同一个程序在一段时间内可以同时存在多个执行活动（即进程），分别对不同的数据进行处理。这时，程序与进程之间存在一对多的关系。多个进程执行了相同的程序。例如，用于计算阶乘的程序既可以计算 10 的阶乘，又可以计算 15 的阶乘，两个计算工作（进程）可以同时进行，互不影响。虽然计算阶乘的具体数值不同，但是算法相同，即都执行同一个程序。这时，两个数值的计算活动不能称为程序，因为“程序”这个名词无法将两者区分开来。必须称这两个计算活动为进程，而且是不同的、相互独立的进程。

### 2.3.2　进程定义和属性

#### 1. 进程的概念

**进程**：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。

该定义表明：进程至少包含程序和数据两个部分。有些系统称进程为“任务”或“活动”。

**2．进程的属性**

（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 3 个组成要素：程序块、数据块和进程控制块。程序也具有结构性，外存上的程序文件包括文件头（也称为程序前缀控制块）、代码及数据结构部分。代码及数据结构部分定义了程序的功能，进程的程序块和数据块就来自于程序代码及数据结构部分。

（2）共享性：同一程序运行于不同数据集合上构成不同的进程。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。

（3）动态性：进程由创建而产生，由调度而执行，由撤销而消亡。进程运行时需要使用处理器、内存、外设、外存等资源。程序是静态的、不活动的，并不使用处理器、内存等执行资源。

（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。

（5）制约性：并发进程之间存在着制约关系，进程在执行的关键点上需要相互等待、互通消息。

（6）并发性：在单处理器系统环境下，各个进程轮流占用处理器。

### 2.3.3 进程状态与切换

操作系统必须对各个进程的活动进行控制，进程控制的依据是进程的状态，进程状态刻画了进程在不同运行阶段所具备的资源利用条件。

**1．三态模型**

1）进程的 3 种基本状态

一个进程从创建而产生至撤销而消亡的整个生命周期至少有如下 3 种基本状态。

（1）运行态（Running）：进程占用处理器正在运行。

（2）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。

（3）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，正在等待某个事件完成。

通常，当一个进程创建后，就处于就绪状态。进程在执行过程中处于上述 3 种状态之一。随着进程的执行，其状态将会发生变化，如图 2-1 所示。

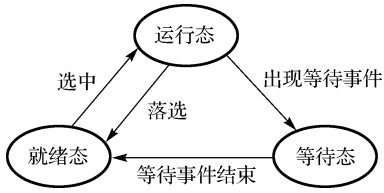


图 2-1 进程的三态模型

2）引起进程状态转换的具体原因

运行态→等待态：等待使用资源或某事件发生。

等待态→就绪态：资源得到满足或事件发生。

运行态→就绪态：运行时间片到或者出现更高优先级进程。

就绪态→运行态：CPU 空闲时选择一个就绪进程。

### 2．五态模型

五态模型在三态模型的基础上，引进了新建态和终止态，如图 2-2 所示。新建态进程刚被创建，尚未提交参与处理器竞争，正在等待操作系统完成创建进程的必要操作。终止态进程已经终止，不再参与处理器竞争，但是进程尚未退出主存。一旦其他进程完成了对终止态进程的信息抽取，则系统将删除该进程。

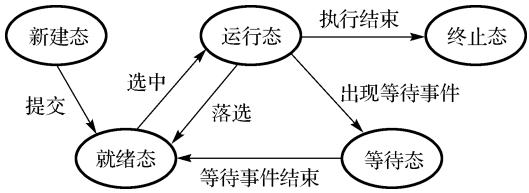


图 2-2　进程的五态模型

进程状态转换的具体原因如下。

NULL→新建态：创建一个子进程。

新建态→就绪态：系统完成进程创建操作，且当前系统性能和内存容量均允许接纳新进程。

运行态→终止态：进程自然结束，或出现无法克服的错误，或被操作系统终结，或被其他有终止权的进程终结。

终止态→NULL：完成善后操作，进程撤离系统。

就绪态→终止态：父进程终结子进程，进程被强行终止。

等待态→终止态：父进程终结子进程，进程被强行终止。

### 3．具有挂起状态的七态模型

1）引入“挂起”状态的原因

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，必须把某些进程挂起，对换到磁盘镜像区中，暂时不参与进程调度，起到平滑系统负荷的目的。

2）引起进程挂起的主要原因

① 当系统中的进程均处于等待状态时，需要把一些阻塞进程对换出去，以腾出足够内存来装入就绪进程运行。

② 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使系统正常运行。

③ 将定期执行的进程（如审计、监控、记账程序）对换出去，以减轻系统负荷。

④ 用户要求挂起自己的进程，以便进行某些调试、检查和改正。

⑤ 父进程要求挂起后代进程，以进行某些检查和改正。

⑥ 操作系统需要挂起某些进程，检查运行中资源的使用情况，以改善系统性能；或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。

3）两个挂起状态

① 挂起就绪态（Ready Suspend）：表明进程就绪但位于外存，待对换到内存后方可调度执行。

② 挂起等待态（Blocked Suspend）：表明进程阻塞并位于外存。

具有挂起状态的进程状态转换关系如图 2-3 所示。

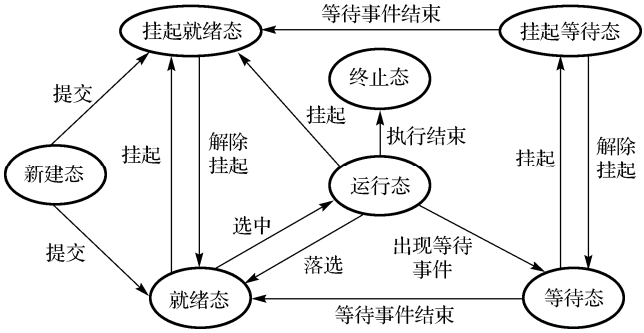


图 2-3 进程七态模型

4）引起进程状态转换的具体原因

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。

挂起等待态→挂起就绪态：引起进程等待的事件结束之后，相应的挂起等待态进程将转换为挂起就绪态。

挂起就绪态→就绪态：内存中没有就绪态进程，或挂起就绪态进程优先级高于就绪态进程，挂起就绪态进程将转换为就绪态。

就绪态→挂起就绪态：为了减轻系统负荷，满足运行进程的资源使用和性能要求，将就绪态进程对换出去成为挂起就绪态。

挂起等待态→等待态：当内存空间充足，某个挂起等待态进程优先级较高，并且导致该进程阻塞的事件即将结束时，该等待进程将被对换到内存中。

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。

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到主存中。

5）挂起进程的特征

① 挂起进程不能立即被执行。

② 挂起进程所等待的事件独立于挂起条件，事件结束并不能导致进程具备执行条件。

③ 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。

④ 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。

## 2.3.4　进程描述

### 1．操作系统的控制结构

为了管理进程和资源，操作系统需要构造相应的数据结构来登记各个进程和资源信息，同样类型的数据结构组成表，称为控制表。操作系统的控制表分为如下 4 类。

（1）进程控制表：管理进程及其相关信息。

（2）存储控制表：管理主存和外存，主要内容包括主存储器的分配信息（分配给进程的内存），辅助存储器的分配信息（分配给进程的外存），存储保护和分区共享信息（哪些进程可以访问共享内存区域），虚拟存储器管理信息。

（3）I/O 控制表：管理计算机系统的 I/O 设备和通道，主要内容包括 I/O 设备和通道是否可用，I/O 设备和通道的分配信息，I/O 操作的状态和进展，I/O 操作传输数据所在的主存区。

（4）文件控制表：管理文件，主要内容包括被打开文件的信息，文件在主存储器和辅助存储器中的位置信息，被打开文件的状态和其他属性信息。

以上 4 种表分别管理计算机系统中的硬件资源（如 CPU、内存、I/O 设备）和软件资源（如文件），如图 2-4 所示。

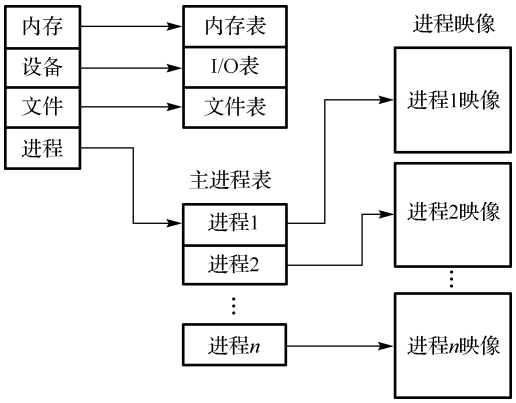


图 2-4　操作系统控制表结构

进程是内存、I/O 设备和文件资源的使用者，为了使操作系统能够跟踪进程对资源的使用情况，内存表、I/O 表和文件表需与进程表关联。主进程表的每项都分别包含一个指向进程映像的指针。在执行期间，进程映像需要驻留内存。

### 2．进程实体（进程映像）的组成

进程实体包括：程序块、数据块、进程控制块和核心栈，如图 2-5 所示。其中，程序块和数据块定义进程的行为和功能，进程控制块用于操作系统跟踪、管理进程，核心栈用于进程运行在核心态下时跟踪过程调用和过程间参数传递信息。


图 2-5　进程实体结构

1）进程控制块

每一个进程都捆绑一个进程控制块，用来存储进程的标志信息、现场信息和控制信息。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）程序块

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。程序块通常是纯代码，可被多个进程共享。

3）数据块

数据块是进程的私有地址空间，是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、用户栈等的存放区，常常为一个进程专用。

4）核心栈

每一个进程都将捆绑一个核心栈，进程在核心态工作时使用，用来保存中断/异常现场，保存函数调用的参数和返回地址。

进程实体的内容随着进程的执行不断发生变化，某时刻进程实体的内容及其状态集合称为**进程映像**。

### 3．进程上下文

**进程上下文**：指进程物理实体和支持进程运行的环境。

UNIX 进程上下文包括如图 2-6 所示的 3 个组成部分。

（1）**用户级上下文**：由正文（用户进程的程序块，只读，用于保存程序指令）、用户数据块、共享存储区和用户栈组成，它们占用进程的虚拟地址空间。用户栈用于保存用户态下过程调用和返回地址、参数传递信息。共享存储区是与其他进程共享的数据区域，用于进程间的通信。

（2）**寄存器上下文**：由程序状态字（PSW）寄存器、指令计数器、栈指针、控制寄存器、通用寄存器等组成。程序未运行时，上述信息保存在寄存器上下文中。

（3）**系统级上下文**：由进程控制块、内存管理信息、核心栈等组成。核心栈用于进程在核心态执行时保存过程调用或中断返回时需恢复的信息。

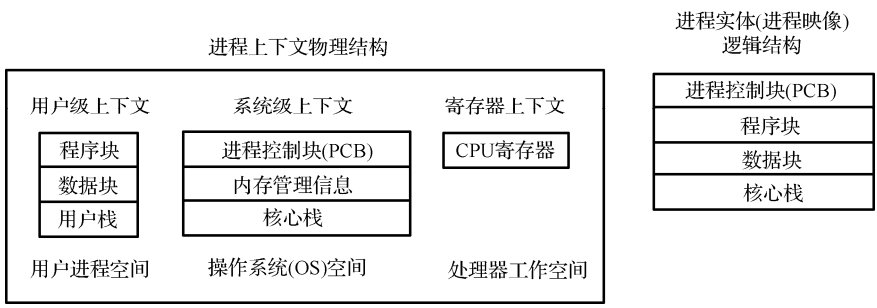


图 2-6　进程上下文与进程实体（进程映像）结构

### 4．进程控制块

每个进程都有且只有一个进程控制块，进程控制块是操作系统用于记录和刻画进程状

态及有关信息的数据结构，也是操作系统掌握进程的唯一结构，是操作系统控制和管理进程的主要依据，是进程存在的唯一标识。它包括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。

进程控制块包含 3 类信息：标识信息、现场信息、控制信息。

（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（进程号）。每个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 ID、进程组标识 ID、用户进程名、用户组名等。

（2）现场信息：用于保留进程运行时存放在处理器现场的各种信息，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、栈指针等。

（3）控制信息：用于管理和调度进程，常用的控制信息如下。

① 进程调度信息，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。

② 进程组成信息，如正文段指针、数据段指针。

③ 进程间通信信息，如消息队列指针、信号量等。

④ 进程在二级（辅助）存储器内的地址信息：如段/页表指针、进程映像在外存中的地址等。

⑤ 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，记账信息。

⑥ 进程特权信息，如内存访问权限和处理器特权。

⑦ 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。

进程控制块各部分的信息所在位置及其与进程实体其他部分的关系，如图 2-7 所示。

进程控制块的使用权和修改权属于操作系统程序，包括调度程序、资源分配程序、中断处理程序、性能监视和分析程序等。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。

### 5．进程队列及其管理

**进程队列**：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）。

进程队列的排队原则如下。

① 同一状态下进程的 PCB 按先来先到、优先级或其他原则排成队列。

② 等待态进程队列按照等待原因细分为多个队列。

进程队列结构如图 2-8 所示。

在一个队列中，链接进程控制块的方法有单向链接和双向链接。

队列管理模块的操作有入队和出队。新提交进程进入就绪队列，操作系统进程调度程序依次调度就绪队列的每个进程占用处理器并获得运行机会。获得处理器的进程进入运行状态，如果在分得的时间片内进程执行完毕，则撤离系统，否则重返就绪队列。如果正在运行的进程产生等待事件，则该进程释放处理器并加入该事件等待队列。待等待事件结束时，进程离开事件等待队列并加入就绪队列，开始等待获得处理器的新一轮的机会。进程在各个队列中入队、出队的情况如图 2-9 所示。

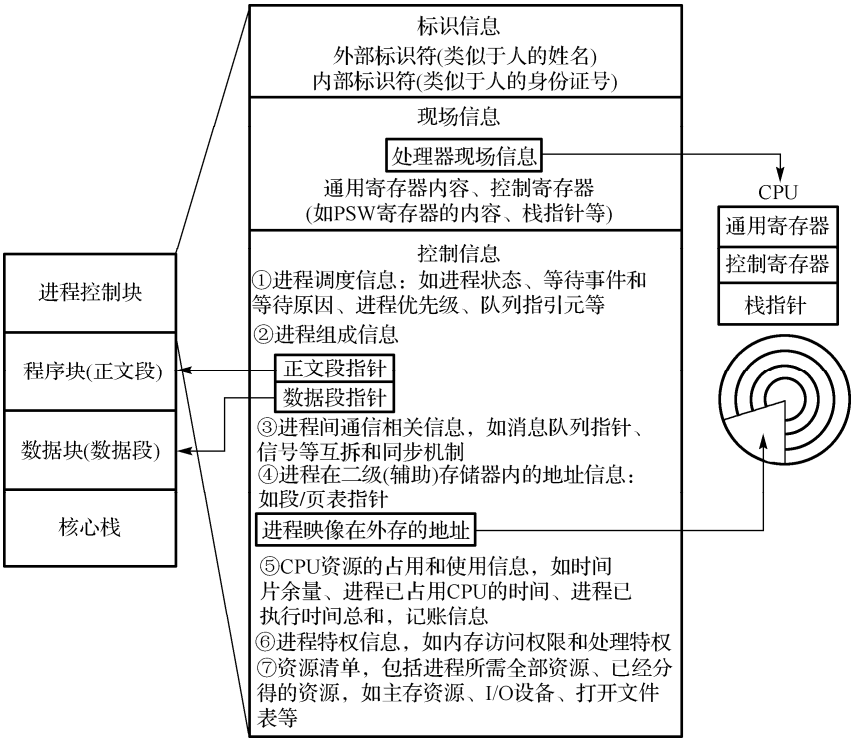


图 2-7　进程控制块的信息

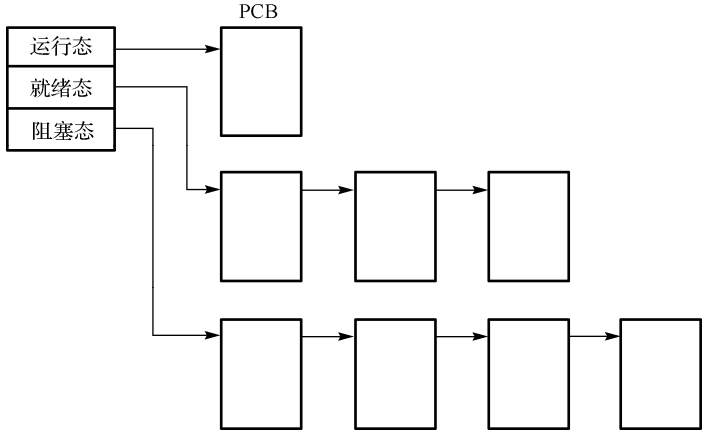


图 2-8　进程队列结构

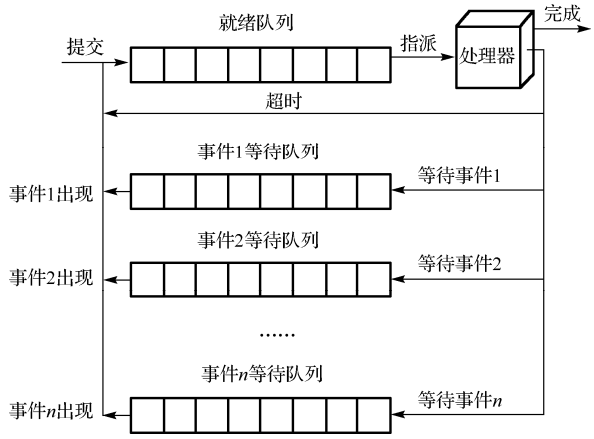


图 2-9　队列管理和状态转换示意图

## 2.3.5　进程切换

进程切换即中断一个进程的执行转而执行另一个进程，被中断进程上下文需要保存到进程控制块中，然后装入新进程上下文，使其从上次断点恢复执行，或者调度一个新的进程。进程切换意味着处理器、输入/输出设备等共享资源的切换，必须将其中的现场信息保存起来，以便进程重新获得调度时恢复现场信息。

**1．允许发生进程上下文切换的 4 种情况（即进程调度时机）**

（1）当进程进入等待态时：一个进程等待时处理器会空闲下来，一个就绪进程应该获得处理器。进程等待事件激活操作系统，并实施进程切换。

（2）当进程完成其系统调用返回用户态，但不是最有资格获得 CPU 时：如图 2-10 所示，当进程 $P_1$ 完成其系统调用即将返回用户态时，操作系统调度程序从众多进程中挑选最有资格获得 CPU 的进程，发现 $P_2$ 而不是 $P_1$ 最有资格获得 CPU，则 $P_2$ 获得 CPU，进程切换发生。

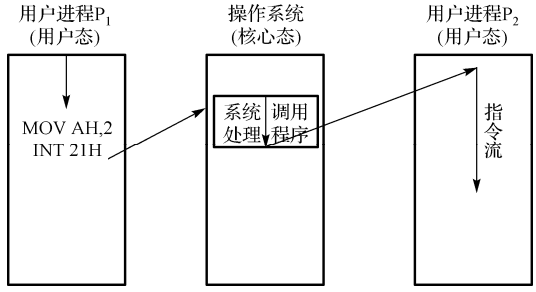


图 2-10　发生进程切换的一种情形（系统调用返回）

（3）当内核完成中断处理，进程返回用户态但不是最有资格获得 CPU 时：这种情况类似于第二种情况，不同之处在于第二种情况中的系统调用是自愿性中断，由当前进程主动引发，其发生时机可以预知。这里说的中断是强迫性中断，非当前进程主动引发，其发生时机不可预知。其处理过程类似于第二种情况。

（4）当进程执行结束时：当进程执行结束时，不再需要处理器，处理器理应分配给其

他进程使用，需要实施进程切换。通常，进程调用“结束进程，返回操作系统”系统调用显式请求操作系统做结束处理，包括进程切换。

例如，IBM PC 汇编程序末尾的语句：

```
MOV AH,4CH
INT 21H
```

表示“结束程序，返回操作系统”。

上述 4 种情况实际上可归纳为一种情况：中断发生时才有可能发生进程上下文切换。上述 4 种情况是对中断事件的细分。只有中断发生时，操作系统才能接管处理器，才有机会把处理器由一个进程转交给另一个进程，进程切换才会发生。

### 2．Linux 进程调度时机

Linux 进程调度时机主要有以下几种。

（1）进程状态转换的时刻，如进程终止（结束）、进程睡眠（阻塞等待），进程调用 sleep()或 exit()等函数进行状态切换，这些函数会主动调用调度程序实施进程切换。

（2）当前进程的时间片用完时，时间片由时钟中断更新，时钟中断处理程序隶属于操作系统内核程序，该情况与第 4 种情况相同。

（3）设备驱动程序执行时，设备驱动程序执行重复而耗时的任务时，会根据调度标志决定是否调用调度程序。

（4）进程从中断、异常或系统调用将要返回用户态时，操作系统将调度和进程切换的时机安排在处理中断事件的时候，因为中断是激活操作系统的唯一方法。

### 3．进程切换的步骤

（1）保存被中断进程的处理器现场信息。

（2）修改被中断进程的进程控制块的有关信息，如进程状态等。

（3）把被中断进程的进程控制块加入有关队列。

（4）选择下一个占用处理器运行的进程。

（5）修改被选中进程的进程控制块的有关信息。

（6）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。

（7）根据被选中进程的信息恢复处理器现场。

进程切换的步骤如图 2-11 所示。

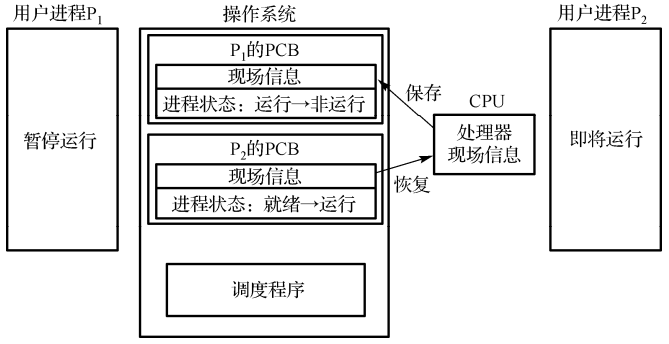


图 2-11　进程切换的步骤

### 2.3.6　模式切换

CPU 模式切换即处理器管态（核心态）与目态（用户态）之间的切换。中断发生时，处理器由执行用户进程代码的用户态切换为执行操作系统内核程序（中断处理程序）的核心态，这是由用户态到核心态的模式切换。内核中断处理程序执行结束后，通过执行程序状态字加载指令可以使处理器由核心态切换为用户态，这是核心态到用户态的模式切换。被中断的进程可以是正在用户态下执行的，也可以是正在核心态下执行的（这属于中断嵌套或多重中断），内核都要保留足够信息以便以后恢复被中断了的进程。

模式切换的步骤如下。

（1）保存被中断进程的处理器现场信息。

（2）处理器由用户状态切换到内核状态，准备执行中断处理程序。

（3）根据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表中找到系统服务程序或中断处理程序的地址。

进程切换包含两次模式切换，一次是处理器由一个进程的用户态切换到核心态，另一次是处理器由核心态切换到另一个进程的用户态。

如果两次模式切换仅仅发生在一个用户进程与操作系统内核程序之间，则进程切换并未发生。进程切换与模式切换的区别如图 2-12 所示。

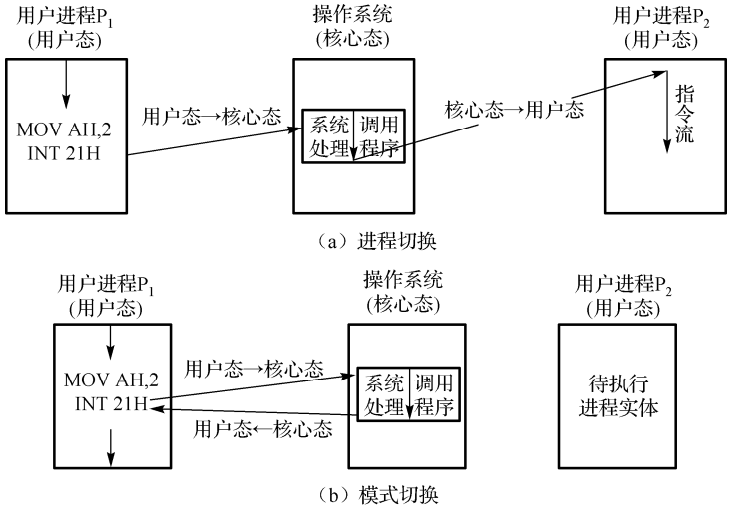


图 2-12　进程切换与模式切换的区别

### 2.3.7　进程控制与管理

进程控制是处理器管理的主要工作，包括进程创建、进程阻塞、进程唤醒、进程挂起、进程激活、进程终止和进程撤销等。这些控制和管理功能是由操作系统中的原语来实现的。

**原语**（Primitive）：在管态下执行、完成系统特定功能的不可中断的过程，具有原子操作性。

根据定义，原语的执行是顺序的而不可能是并发的。

原语的实现方法：原语可以采用屏蔽中断的系统调用来实现，以保证原语操作不被打断。也就是说，原语和系统调用都使用访管指令实现，具有相同的调用形式。但普通系统调用可以被中断，原语不可中断。

### 1．进程的创建

1）进程创建的事件来源

① 提交一个批处理作业。

② 交互式作业登录。

③ 操作系统创建一个服务进程。

④ 存在的进程创建（孵化）新的进程。

生成其他进程的进程称为父进程（Parent Process），被生成的进程称为子进程（Child Process），即一个父进程可以创建子进程，从而形成树形结构。

2）进程的创建过程

① 在主进程表中增加一项，并从 PCB 池中取一个空白 PCB，为新进程分配唯一的进程标识符。

② 为新进程的进程映像分配地址空间，装入程序和数据。

③ 为新进程分配内存空间外的其他资源。

④ 初始化进程控制块，如进程标识符、处理器初始状态、进程优先级等。

⑤ 把进程状态置为就绪态并加入就绪进程队列。

⑥ 通知操作系统的某些模块，如记账程序、性能监控程序。

### 2．进程的阻塞和唤醒

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，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。

1）进程阻塞的步骤

① 停止进程执行，保存现场信息到 PCB 中。

② 修改 PCB 的有关内容，如进程状态由运行改为等待，并把修改状态后的 PCB 加入相应等待队列。

③ 转入进程调度程序，调度其他进程并运行。

2）进程唤醒的步骤

① 从相应等待队列中移出进程。

② 修改 PCB 的有关信息，如将进程状态改为就绪并把修改 PCB 后的进程加入就绪队列。

③ 若被唤醒的进程优先级高于当前运行的进程，则重新设置调度标志。

在 UNIX/Linux 中，与进程的阻塞与唤醒相关的原语主要有：sleep（暂停）、pause（暂停并等待信号）、wait（等待子进程暂停或终止）和 kill（终止进程）。

### 3．进程的撤销

1）进程撤销的主要原因

① 进程正常运行结束。

② 进程执行了非法指令，或在常态下执行了特权指令。

③ 进程运行时间或等待时间超越了最大限定值。

④ 进程申请的内存超过最大限定值。

⑤ 越界错误、算术错误、严重的输入/输出错误。

⑥ 操作员或操作系统干预。

⑦ 父进程撤销其子进程、父进程撤销、操作系统终止。

2）进程撤销步骤

① 根据撤销进程标识号，从相应队列找到它的 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，则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 归还给 PCB 池。

### 4．进程的挂起和激活

挂起原语执行过程：检查要被挂起进程的状态，若处于活动就绪态，则修改为挂起就绪态；若处于阻塞态，则修改为挂起阻塞态。被挂起 PCB 的非常驻部分要交换到磁盘对换区中。

激活原语主要处理过程：把 PCB 非常驻部分调入内存，修改它的状态，将挂起等待态改为等待态，将挂起就绪态改为就绪态，加入相应队列。挂起原语既可由进程自己也可由其他进程调用，但激活原语只能由其他进程调用。

## 实验 4　Linux 进程控制实验

### 1．进程的创建、阻塞、终止

使用 kill 终止进程，使用 sleep 使进程自己睡眠，使用 waitpid 等待子进程结束。

编写程序 killer.c：

```
#include <sys/wait.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
int main( void )
{
    pid_t childpid;
    int status;
    int retval;
    childpid = fork();
    if ( -1 == childpid )
    {
        perror( "fork()" );
```