# 第八章 操作系统支持

8.1操作系统概述

8.1.1操作系统的目标和功能

8.1.2操作系统的类型

8.2调度

8.2.1长期调度

8.2.2中期调度

8.2.3短期调度

8.3内存管理

8.3.1交换

8.3.2分区

8.3.3分页

8.3.4虚拟存储器

  8.3.5 快表（转译后备缓冲器，TLB）

8.3.6分段

8.4英特尔x86内存管理

8.4.1地址空间

  8.4.2分段

8.4.3分页

8.5 ARM存储器管理

  8.5.1存储系统组织

8.5.2虚拟内存地址转换

  8.5.3内存管理格式

8.5.4访问控制

8.6 关键术语、复习题和问题

学习目标

学习本章后，你应该能够：

● 在最高层级上概述操作系统（操作系统）的关键功能。

● 讨论操作系统从早期简单批处理系统到现代复杂系统的演变过程。

● 解释长期、中期和短期调度之间的差异。

● 理解内存分区的原因，并解释所使用的各种技术。

● 评估分页和分段的相关优势。

● 定义虚拟内存。

虽然本文的重点是计算机硬件，但软件中的一大领域----计算机操作系统必须要说明。操作系统是管理计算机资源、为程序员提供服务并调度其他程序执行的程序。了解操作系统对于理解CPU控制计算机系统的机制至关重要。特别是，它能很好地解释中断的作用和存储器层次结构的管理。

    本章先从操作系统的概述和简要历史讲起。本章的大部分内容将着眼于介绍两个操作系统功能，也是与计算机组织和体系结构研究最相关的两个功能：调度和内存管理。

## 8.1操作系统概述

### 操作系统的目标和功能

操作系统是一个控制应用程序执行的程序，它充当应用程序和计算机硬件之间的接口。它有两个目标：

便捷：操作系统使计算机更方便使用。

高效：操作系统使得计算机系统资源得到高效利用。

下面将依次研究操作系统的这两个方面。

1. 操作系统提供用户和计算机之间的接口  用于向用户提供应用程序的硬件和软件可以分为不同的层次，如图8.1所示。这些应用程序的用户，即最终用户，通常不关心计算机的体系结构。因此，最终用户将计算机看作一个应用系统。它可以用编程语言来表示，并由应用程序员开发。使用一组处理器指令来开发完全管理计算机硬件的应用，这是一项极其复杂

的任务。为了简化这一任务，提供了一套系统程序。这些程序中的一些被称为实用程序。这些程序用于程序创建、文件管理以及I/O设备的控制。程序员在开发应用程序时使用这些工具，这些应用程序在运行时会调用实用程序来执行某些功能。最重要的系统程序是操作系统。操作系统向程序员隐藏硬件细节，并为程序员使用系统提供了方便的接口。它充当中介，使程序员更容易访问应用程序并使用系统提供的设施和服务。

简言之，操作系统通常提供以下领域的服务：

■程序创建：操作系统提供各种工具和服务，如编辑器和调试器，用以帮助程序员创建程序。通常，这些服务是以实用程序的形式提供的，这些实用程序实际上不是操作系统的一部分，但是可以通过操作系统访问。

■程序执行：执行程序需要执行许多步骤。指令和数据必须加载到主存中，I/O设备和文件必须初始化，并且必须准备其他资源。操作系统为用户处理所有这些操作。

■访问I/O设备：每个I/O设备需要它自己特定的一组指令或控制信号来操作。操作系统负责处理细节，从而使程序员能够通过简单的读写进行操作。

■对文件的受控访问：对于文件，控制不仅需要了解I/O设备（磁盘驱动器、磁带驱动器）的属性，同时需要了解文件的存储格式。同样的，操作系统关注实现细节。此外，在多用户同时系统中，操作系统还可以提供文件访问的保护机制。

■系统访问：在共享或公共系统中，操作系统整体控制系统访问并分配系统资源。访问功能必须对资源和数据提供保护，使其免受未授权用户的攻击，并且必须解决资源争用的冲突。：

■错误检测和响应：在计算机系统运行时，可能发生各种错误。这些错误包括内部和外部硬件错误，例如内存错误或设备故障或故障；还包括各种软件错误，例如算术溢出、企图访问受禁止的内存区域，以及操作系统未授权的应用请求。在各种情况中，操作系统做出响应清除错误，尽量减少对执行中的应用程序的影响。响应方法有终止引起错误的程序，重试，也可以简单地向应用发送错误报告。

■统计：一个好的操作系统，可以收集统计各种资源的使用信息，并监视性能参数，如响应时间。在任何系统上，这些信息都为将来增加系统功能，改善系统性能方面提供有效的参考。在多用户系统中，这些信息可以用于计费目的。图8.1还展示了典型计算机系统中的三个关键接口：

■指令集体系结构（ISA）：ISA定义了计算机遵循的机器语言指令集。这个接口是硬件和软件之间的边界。请注意，应用程序和实用程序都可以直接访问ISA。对于这些程序，指令指令集的子集是可用的（用户ISA）。操作系统可以访问管理系统资源（系统ISA）的其他机器语言指令。

■应用二进制接口（ABI）：ABI定义了跨程序的二进制可移植的标准。ABI通过用户ISA定义了对操作系统、硬件资源以及系统中可用服务的系统调用接口。

■应用程序编程接口（API）：API通过用户ISA以及高级语言（HLL）库调用，使程序访问系统中可用的硬件资源和服务。系统调用通常是通过库执行的。使用API使得应用程序软件能够通过重新编译，轻松地移植到支持相同API的其他系统中。

2. 作为资源管理器的操作系统

     计算机一组具有移动、存储和处理数据功能以及控制这些功能的资源。操作系统负责管理这些资源。

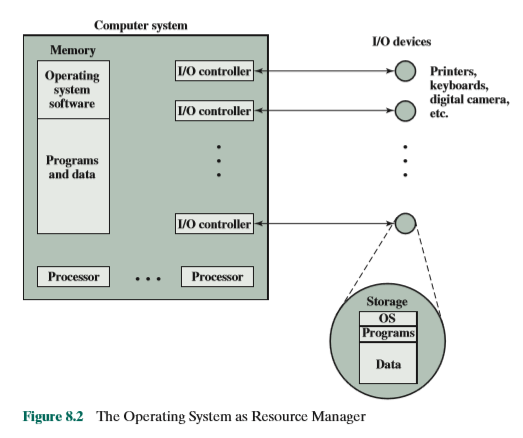
       我们可否认为操作系统控制数据的移动、存储和处理？从某种程度上说，答案是肯定的：通过管理计算机的资源，操作系统控制着计算机的基本功能。但这种控制是以一种独特的方式进行的。通常，我们认为控制机构是被控制的事物之外的东西，或者至少是与受控对象分离的的独立部分（例如，住宅供暖系统由温控器控制，其完全不同于发热装置和热供应装置)对于操作系统来说并非如此，它作为控制机构在两个方面是独特的：

■ 操作系统的功能与普通计算机软件相同，即它是由处理器执行的程序。

■ 操作系统经常放弃控制，并且必须依赖处理器才能重新获得控制。

和其他计算机程序一样，操作系统为进程提供指令。主要差别在程序执行的目的。操作系统指导处理器使用其他的系统资源，为其他的程序定时。但是，为了让处理器完成这些事情，它必须终止执行操作系统程序并执行其他程序。因此，操作系统放弃控制，让处理器做一些“有用”的工作，然后恢复控制，使处理器有足够长的时间做好准备进行下一项工作。随着本章的深入，这一机制将逐渐清晰。

图8.2显示了操作系统管理的主要资源。操作系统的一部分在主存储器中，包括**内核**或者成为**核**，内核包含操作系统中最常使用的函数，以及在给定的时刻，当前使用的操作系统的其他部分。主存储器的其余部分包含用户程序和数据。我们将看到的，这个资源（主存）的分配由操作系统和处理器中的内存管理硬件共同控制。操作系统决定程序何时可以使用I/O设备，并控制文件的访问和使用。处理器本身是一种资源，操作系统必须确定要给一个



特定的用户程序分配多少处理器时间。在多处理器系统中，这个决策必须对所有处理器进行裁决。

### 操作系统的类型

可以根据一些主要特征来对操作系统进行分类。分类方式有两种：第一种分类将操作系统分为批处理和交互式处理。在**交互式**系统中，用户或程序员通常通过键盘或显示终端直接与计算机交互，来请求执行作业或执行事务。此外，根据应用程序的性质，用户可以在执行作业期间与计算机通信。**批处理系统**与交互式系统正相反。用户的程序与其他用户的程序打包在一起，并由计算机操作员成批提交处理。程序完成后，再将结果打印给用户。纯批处理系统目前应用的比较少了，然而，简要地了解下批处理系统，可以帮助理解现代操作系统

另一种分类是看系统是否采用了**多道程序设计**。多道程序设计系统通过让处理器一次处理多个程序，来使处理器尽可能繁忙。多个程序被加载到内存中，处理器在它们之间快速切换。另外一种是**单道程序设计**系统，处理器一次只运行一个程序。

**1.早期系统** 最早的计算机出现在20世纪40年代末到50年代中期，那时没有操作系统，程序员直接与计算机硬件交互。控制台控制机器运行，它由指示灯、拨动开关、输入设备和打印机组成。处理器代码中的程序通过输入设备（例如读卡器）加载。如果程序出错中止，指示灯会指示错误状态。程序员可以通过检查寄存器和主存储器来确定错误的原因。如果程序继续正常完成，输出将出现在打印机上。

这些早期系统存在两个主要问题：

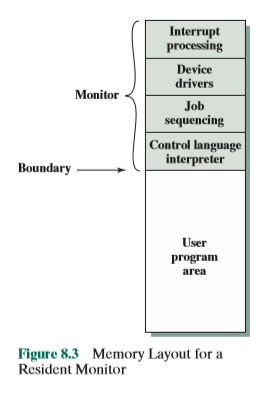
* **调度**：大多数程序会通过注册表来预约处理器时间。通常，用户可以注册预约一段时间，如数个半小时。用户可能签约了一个小时，但在45分钟内完成了，这就浪费了处理器时间。另外，用户可能会遇到问题，导致没有在指定的时间内完成，只能停止等待问题解决后继续任务。
* **安装时间**：一个单一的程序称为**作业**，一个作业的完成分为以下几个步骤：将编译器和高级语言程序（源程序）加载到内存中，保存已编译的程序（目标程序），然后将目标程序和公共函数加载并连接在一起。这些步骤中的每一个都可能要安装或拆卸磁带，或装配卡片。如果发生错误，用户只能重新安装一遍。因此，大量的时间被浪费在了程序的安装上。

这种操作模式被称为串行处理，即用户必须按顺序使用计算机。后来，人们开发出各种各样的系统软件工具，试图提高串行处理的效率。包括通用函数库、链接器、加载器、调试程序和I/O驱动程序程序，这些程序可以作为共享软件被所有用户使用。

**2. 简单批处理系统** 早期的处理器非常昂贵，因此最大化利用处理器非常重要。由于调度和安装而浪费的时间是不可接受的。

为了提高利用率，人们开发了简单的批量操作系统。这种系统也被称为**监控程序**，用户不能再直接访问进程，而是将写在卡片或磁带上的作业提交给计算机操作员，计算机操作员按顺序将作业成批放在在一起，然后放到输入设备上，供监控程序执行。

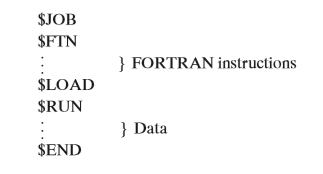
为了理解这种工作机制，我们可以从两个角度来看：监控程序角度和处理器角度。从监控程序的角度来看，监控程序控制事件的顺序。为此，大部分监控程序必须始终在主存中，并且可用于执行（图8.3）。该部分监控程序被称为**常驻监控程序**。监控程序的其余部分由实用程序和公共函数组成，这些程序和函数在需要它们的作业开始执行时，作为子程序加载到用户程序上。监控程序一次从输入设备（通常是读卡器或磁带驱动器）读取一个作业。当读取时，当前作业被放置在用户程序区域中，控制权被递交给该作业。当作业完成时，它将控制返还给监控程序，监控程序立即读取下一个作业。每个作业的结果都被打印出来给用户。



现在，从处理器的角度来看这个执行序列。在某个时间点，处理器执行包含监控程序的来自主存储器部分的指令。这些指令把下一个作业读入到主存储器的另一部分。一旦读入了作业，处理器将在监控程序中收到一条分支指令，该指令指示处理器在用户程序开始的位置继续执行。然后，处理器将在用户的程序中执行指令，直到结束或遇到错误。这时处理器从监视程序获取其下一条指令。因此，术语“控制被传递给作业”仅仅意味着处理器现在在用户程序中获取并执行指令，而“控制被返回给监控程序”意味着处理器现在从监控程序程序获取并执行指令。

应该清楚的是，监控程序处理调度问题。将一批作业排好序，作业被尽可能快地执行，不会浪费处理器的时间。

作业安装时间呢？监控程序也处理这个问题。对于每个作业，指令都包含在**作业控制语言（JCL）**中。这是一种特殊的编程语言，用于向监控程序提供指令。例如，用户提交一个用FORTRAN编写的程序并加上一些要被程序使用的数据。每个FORTRAN指令和每个数据项都放在单独的卡片或磁带的记录区上。除了FORTRAN和数据行之外，作业还包括作业控制指令，这些指令由开头“$”表示。整个程序有如下的格式：

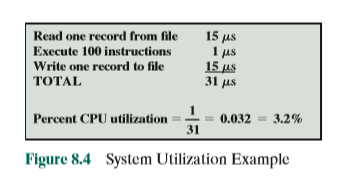


为了执行此作业，监控程序读取$FTN行，并从其大容量存储器（通常是磁带）加载适当的编译程序。编译器将用户的程序翻译成目标代码，存储在内存或大容量存储器中。如果该操作存储在内存中，则称为“编译、装载和运行”。如果是存储在磁带上，则需要$LOAD指令。该指令由监控程序读取，该监控程序在编译操作之后重新获得控制。监控程序调用加载器，该加载器将目标程序加载到内存中，替代编译程序，并将控制传递给目标代码。以这种方式，可以在不同子系统之间共享大段主存储器，尽管一次只能驻留和执行一个这样的子系统。

我们看出，监控程序或批处理操作系统就是简单的计算机程序。它依赖处理器的能力从主存的各个单元取指令来达到获取和放弃控制权交替进行的目的。同时它还需要其他硬件的支持：

* **存储器保护**：当用户程序执行时，它必须不改变包含监控程序的存储区域。如果进行了这样的尝试，则处理器硬件应该检测错误并将控制传送到监控程序。然后，监控程序将中止作业，打印出错误消息，并加载下一个作业。
* **定时器**：定时器用于防止单个作业垄断系统。计时器设置在每个作业的开头。如果定时到期，则发生中断，并将控制权返回给监控程序。
* **特权指令**：某些指令被指定为特权指令，只能由监控程序执行。如果处理器在执行用户程序时遇到这样的指令，将会发生错误中断。特权指令包括/O指令，以便监控程序保持对所有I/O设备的控制。举个例子，这种指令可以防止用户程序意外读取来自下一个作业的作业控制指令。如果用户程序想要执行I/O，则它必须请求监控程序来执行操作。如果处理器在执行用户程序时遇到了一个特权指令，处理器硬件将认为这是一个错误，并将控制权传递给监控程序。：
* **中断**：早期的计算机模型没有这种能力。这个特性使操作系统更方便地挂起和获得控制权。

处理器交替执行用户程序和和监控程序。这就存在两个开销：一是一些主存被监控程序占用，二是监控程序占用了一些处理器时间。这两者都属于系统开销。即使存在这样的开销，简单的批处理系统仍提高了计算机的利用率。

**3. 多道程序批处理系统** 即使简单批处理操作系统提供了自动作业排序，处理器也常常是空闲的。问题在于与处理器相比，I/O设备速度较慢。图8.4详细描述了一个有典型的计算。该计算考虑一个程序，该程序处理记录文件，并且平均每条记录执行100个机器指令。在这个例子中，计算机花费超过96%的时间等待I/O设备完成数据传输！图8.5a说明了这种情况。处理器运行了一段时间，直到达到I/O指令停止。然后它必须等待I/O指令结束之后

才能继续执行。

这种低效率是不必要的。我们知道必须有足够的内存来装载操作系统（常驻监控程序）和一个用户程序。假设有一定空间可以装载操作系统和两个用户程序。现在，当一个作业需要等待I/O时，进程可以切换到另一个作业，该作业可能不等待I/O（图8.5b）。此外，我们可能会扩展内存来容纳三个、四个或更多的程序，并在它们之间切换（图8.5c）。这种技术被称为**多道程序设计**或**多任务化**，这是现代操作系统的中心议题。

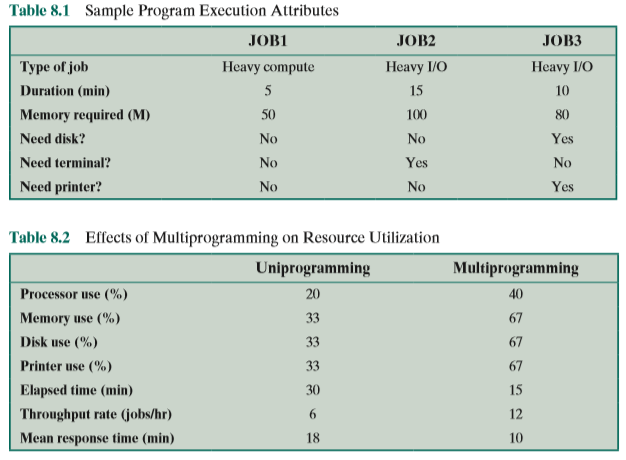
-------注释

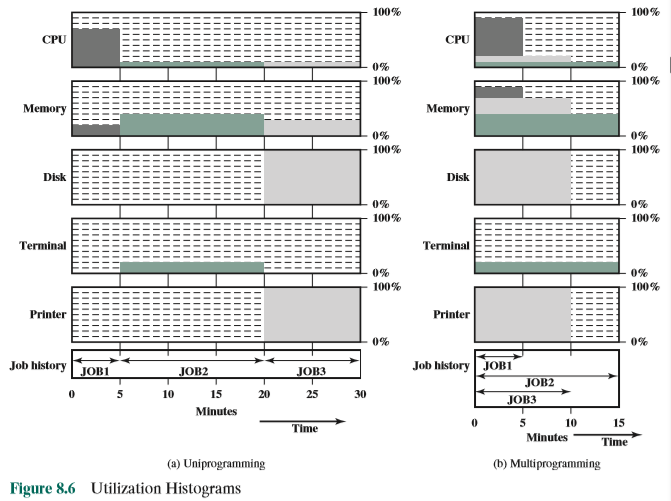
1术语“多任务化”优势专用于指同一程序可被处理器并发处理的多个任务。而术语“多道程序设计”是指来自多个程序的多个进程。但更经常的是将两个术语等价使用，正如大多数标准字典一样（例如，IEEE Std 100-1992，The New IEEE Standard Dictionary of Electrical and Electronics Terms）。

**例8.1**这个例子可以说明多道程序设计的好处。考虑一个具有250M字节可用内存（操作系统不使用）的计算机、一个磁盘、一个终端和一台打印机。同时提交执行三个程序JOB1、JOB2和JOB3，属性如表8.1所示。我们假设JOB1和JOB2有最小的处理器需求，JOB3连续使用磁盘和打印机。在一个简单的批处理环境下，这些作业按顺序执行。因此，JOB1在5分钟内完成。作业2必须等到5分钟结束才开始执行，然后等15分钟后完成。作业3在20分钟后开始，在最初提交时间30分钟后完成。平均资源利用率、吞吐量和响应时间如表8.2的“单道程序设计”列中。图8.6a显示了每个设备的利用率。很显然，在需要的30分钟时间段内进行平均时，所有资源的利用率都严重不足。

现在假设作业在多道程序设计操作系统中并发运行。因为作业之间几乎没有资源争用，所以这三个作业都几乎可以以最短的时间执行，同时与计算机中的其他作业并发执行（假设为JOB2和JOB3分配了足够的处理器时间，可以确保它们的输入和输出操作）。JOB1仍然需要5分钟才能完成，但是到那个时间结束时，JOB2将完成三分之一，而JOB3将完成一半。所有三个工作将在15分钟内完成。通过表8.2的“多道程序”列可以看出这种改进十分明显，其中数据来自图8.6b的直方图。

与简单的批处理系统一样，多道程序设计批处理系统必须依赖于某些计算机硬件特性。



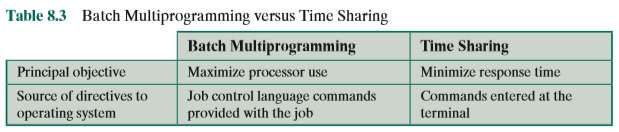
多道程序设计最显著的附加特点是其硬件支持I/O中断和DMA。通过中断驱动的I/O或DMA，处理器可以对一个作业发出I/O命令，并在I/O操作由设备控制器执行的同时，继续执行另一个作业。当I/O操作完成时，处理器被中断，控制被传递给操作系统中的中断处理程序。然后操作系统将控制传递给另一个作业。

与单一程序或单道程序设计系统相比，多道程序设计的操作系统是相当复杂的。为了让多个作业就绪来执行，作业必须保存在主存中，这需要某种形式的**内存管理**。此外，如果多个作业同时就绪，处理器则必须决定运行哪个作业，这需要一些调度算法。本章后面将讨论这些概念。

**4.分时系统** 随着多道程序设计的使用，批处理方法显得非常有效。然而，许多作业需要用户和计算机直接交互。例如一些事务处理的作业，必须采用交互模式。

现在，通常采用专用的微型计算机作为交互式计算设备。在20世纪60年代，由于许多计算机体积大且价格昂贵。采用专用微型计算机来实现交互式是不可选的，因此，分时系统应运而生。

正如多道程序设计允许处理器一次处理多个批处理作业一样，多道程序设计也可以同时处理多个交互式作业。在后一种情况下，该技术被称为分时，因为处理器时间被多个用户共



享。在**分时系统**中，多个用户同时通过终端访问系统。操作系统按很短的时间片或计算量来交错执行每个用户程序。因此，如果同时有n个用户主动请求服务，则每个用户将只看到平均1/n的计算机有效速度，而不计操作系统的开销。然而，鉴于相对缓慢的人类反应时间，适当设计系统，其响应时间应该与专用系统相当。

批处理的多道程序设计和分时系统都采用多道程序设计。表8.3列出了它们的主要区别。

## 8.2调度

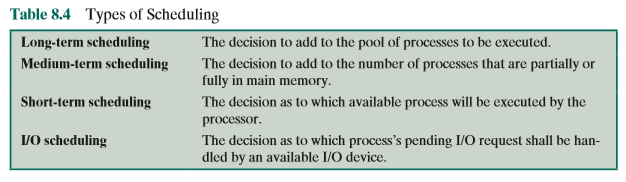
多道程序设计的关键是调度问题。事实上，调度有4中典型的类型（如表8.4所示）。我们现在将再次探讨这些类型。首先介绍一个概念：**进程**。这个术语最早是在20世纪60年代由Multics 操作系统的设计者提出。从某种程度上讲，这个术语比作业更通用。对于术语进程，已经有许多定义，包括：

* 一个执行中的程序
* 一个程序的“活灵魂”
* 一个处理器分配的实体

随着我们讨论的深入，进程的概念将变得更加清晰。

### 长期调度

长期调度程序确定哪些程序提交给系统进行处理。因此，它控制多道程序设计的深度（内存中的进程数）。一旦提交，作业或用户程序就成为进程，并被添加到短期调度器的队列中。在某些系统中，新创建的进程处于换出状态，在这种情况下，它们会被添加到中期调度器的队列中。



在批处理系统或通用操作系统的批处理部分中，新提交的作业被放在磁盘上并被保存在批处理队列中。长期调度程序从该队列创建进程。这里有两个决定性的因素。首先，调度程序必须确定操作系统可以运行一个或多个额外的进程。第二，调度程序必须决定接受哪一个作业或作业组，并将其转换为进程。采用的标准可以包括优先级、预期执行时间和I/O需求。

对于分时系统中的交互式程序，当用户试图访问系统时会产生一个进程请求。分时用户并不只是排队等待系统接受他们。相反，操作系统通过使用一些预先定义的饱和度量，接受所有授权的用户，直到系统饱和。此时，连接请求将会遇到一条“系统已饱和请稍后重试”的信息。

### 中期调度

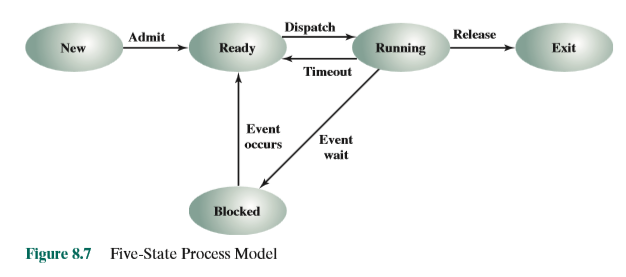
中期调度是8.3节将介绍的交换功能的一部分。通常，换入决策是基于需要管理多道程序的程度。在不使用虚拟存储的系统上，存储管理也是一个问题，因此，换入决策要考虑到换出进程的存储器需求。

### 短期调度

长期调度程序执行相对不频繁，并且它只粗粒度判定是否执行一个新的进程，或者调度哪个进程。短期调度程序（也称为派遣程序）执行频繁，并且细粒度地判定接下来执行哪一个作业。

**1.进程状态**为了理解短期调度程序的操作，我们需要考虑**过程状态**的概念。在进程的生命周期中，它的状态将改变很多次。它在任何时间点的状态被称为状态。之所以使用“状态”这个术语，是因为它意味着存在某些信息，这些信息定义了此时的状态。对于一个进程来说，至少有5种定义的状态（如图8.7所示）：

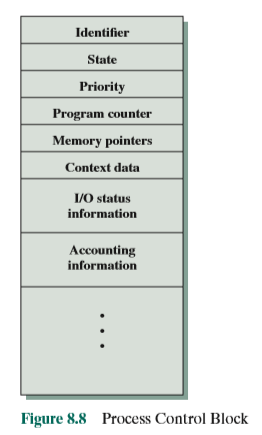
* **新建**：程序被高级调度程序提交，但还没有就绪。操作系统将初始化一个进程，将其移动到就绪状态。



* **就绪**：该进程以准备就绪，正在等待对处理器的访问。
* **运行**：进程正在被处理器执行。
* **等待**：进程被挂起，等待某些系统资源，如I/O。
* **终止**：进程结束，将被操作系统销毁。

对于系统中的每个进程，操作系统必须维护进程状态信息以及进程执行所需的其他信息。为此，每个进程在操作系统中由一个**进程控制块**（如图8.8所示）表示，它通常包含：

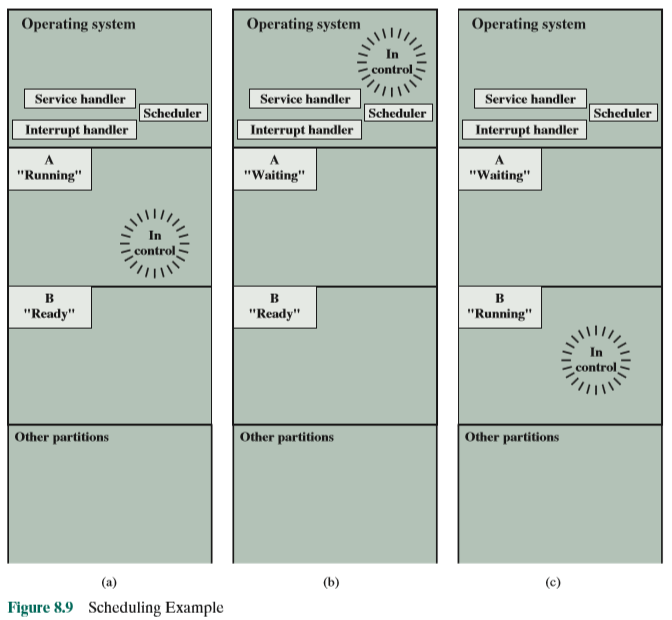
* **标识符**：每个当前进程都有唯一的标识符。
* **状态**：过程的当前状态（新建、就绪等）。
* **优先级**：进程的相对优先级别。
* **程序计数器**：要执行的程序中的下一个指令的地址。
* **存储器指针**：进程在内存中的开始和结束位置。
* **上下文数据**：进程执行时，这些数据保存在处理器中的寄存器里，它们将在本书第三部分中予以讨论。现在，可以说这些数据表示进程的“上下文”。当进程离开“运行”状态时，将保存上下文数据和程序计数器。当恢复执行这个进程时，处理器需要读回这些信息。



* **I/O状态信息**：包括未完成的I/O请求、分配给该进程的I/O设备（例如，磁带驱动器）、分配给该进程的文件列表等。
* **统计信息**：可以包括使用处理器的时间和使用的时钟时间、时间限制、帐户号码等。

当调度程序接受一个新的作业或用户的执行请求时，它创建一个空白的过程控制块，并使相关进程处于新的状态。当系统填完PCB后，进程进入就绪状态。

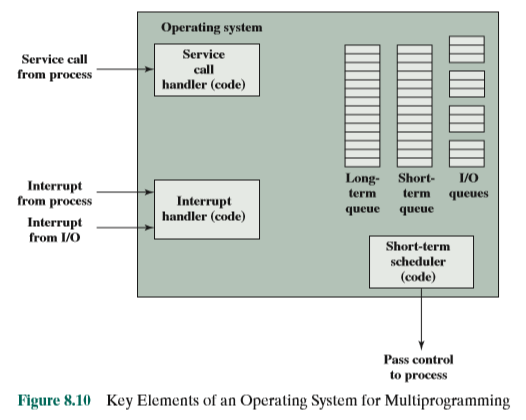
**2.调度技术** 为了理解操作系统如何管理存储器中各种作业的调度，让我们首先考虑图8.9中的简单例子。该图显示了在给定的时间点主存的分区情况。当然，操作系统的内核常驻内存。此外，还有许多活动的进程，包括A和B，都需要分配存储空间。



我们从进程A开始运行的时刻开始考虑。处理器正在执行A程序的指令。在稍后的某个时间点，处理器停止执行A中的指令，开始执行操作系统中的指令。这是由下列三种原因之一引起的：

1. 进程A向操作系统发出一个服务请求（例如一个I/O请求）。A处于挂起状态，直到操作系统满足此请求为止。
2. 进程A产生中断。中断是由硬件产生并发送给处理器的信号。当检测到这个信号时，处理器停止执行A，并转移到操作系统的中断处理程序。与A相关的各种事件都可以引起中断。一个简单的例子是出错，例如试图执行特权指令。另一个例子是超时；为了防止任何一个进程垄断处理器，每个进程一次只被允许占用处理器很短的时间。
3. 一些时间与进程A无关，当仍需要注意其引起中断，例如I/O操作完成。

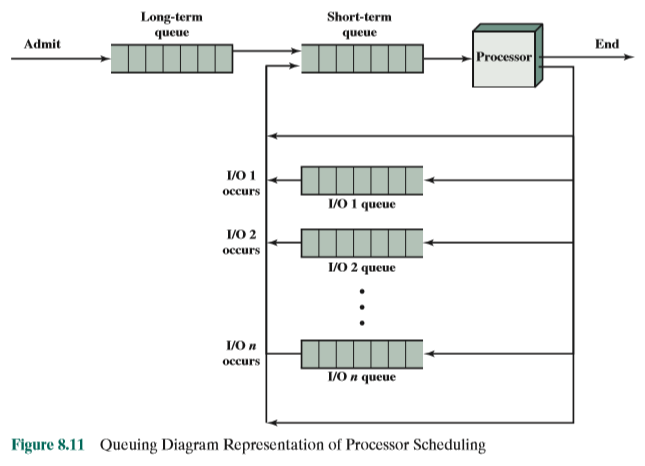
无论何种原因，结果均如下。处理器将当前上下文数据和用于A的程序计数器保存在A的进程控制块中，然后开始在操作系统中执行A。操作系统可以执行一些工作，例如初始化I/O操作。然后，操作系统的短期调度程序决定接下来要执行的进程。在这个例子中是B进程。操作系统指示处理器恢复B的上下文数据，并继续执行B进程。

这个简单的例子描述了短期调度的基本功能。图8.10显示了涉及多道程序设计和进程调度的操作系统中的主要元素。当出现一个中断时，操作系统接收处理器的控制权，执行中断处理程序。如果发生服务调用，则执行服务调用处理程序。一旦中断或服务调用完毕，短期调度则调度某个进程来执行。 

为了实现进程调度机制，操作系统需要维护多个队列。每个队列都是一个简单的等待某些资源的进程列表。**长期队列**是等待系统资源的作业列表。如果条件允许，高级调度程序将为等待的作业分配内存并创建一个进程。**短期队列**由处于就绪状态的所有进程组成。这些进程中的任何一个可能下次被处理器调用。这取决于短期调度程序的选择。一般采用轮转算法，给每个过程轮流分配一些时间。也可以使用优先级算法。最后，每个I/O设备都有一个I/O队列。多个进程可以请求使用相同的I/O设备。等待使用每个设备的所有进程排列在该设备队列中。

图8.11显示了在操作系统的控制下的进程流程。每个进程请求（批处理作业、用户定义的交互作业）都被放在长期队列中。随着资源变得可用，进程请求变成一个进程，然后被置于就绪状态，并被放到短期队列中。处理器在执行操作系统指令和执行用户进程之间交替。当操作系统获得控制权时，它决定短期队列中接下来应该执行的进程。当操作系统完成当前任务时，由处理器处理已选中的进程。

如前所述，正在执行的进程可能由于各种原因而挂起。如果因为进程请求I/O而被挂起，



则将进入相应的I/O队列中。如果由于超时或操作系统必须处理紧急事务而挂起，则它处于就绪状态，并被置于短期队列中。

最后，需要说明的是操作系统还管理I/O队列。当I/O操作完成时，操作系统从I/O队列中移除请求得到满足的进程，并将其置于短期队列中。然后，它选择另一个等待进程（如果有的话）并为I/O设备发送信号来满足该进程的请求。

## 8.3存储器管理

在单道程序设计系统中，主存被分成两部分：一部分用于操作系统（常驻监控程序），一部分用于当前正在执行的程序。在多道程序设计系统中，内存的“用户”存储区被细分给多个进程使用。细分任务由操作系统动态执行，被称为**存储器管理**。

有效的内存管理在多道程序设计系统中至关重要。如果存储器中只有几个进程，那么大部分时间所有进程都将等待I/O，处理器将处于空闲状态。因此，需要有效地分配内存，尽可能地让更多的进程进入存储器。

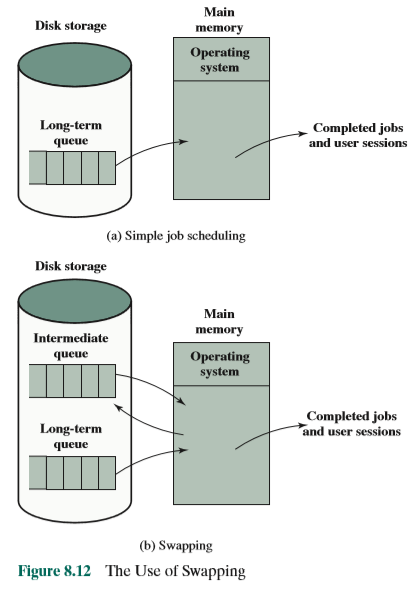
### 交换

回到图8.11，我们已讨论了三种类型的队列：请求新进程的长期队列，进程就绪等待处理器的短期队列，以及进程未就绪的各种I/O队列。重提一下，设计这种处理方法的原因是I/O活动比处理器计算慢得多，因此在单道程序设计系统中，处理器大部分时间是空闲的。

但是图8.11中的队列并不能完全解决问题。的确，在这种情况下，内存保存了多个进程，并且当一个进程正在等待时，处理器可以移动到另一个进程。但是处理器比I/O快得多，所以内存中的所有进程都等待I/O是很常见的。因此，即使使用多道程序设计，处理器大部分时间也可能是空闲的。

怎么办？扩展主存以能够适应更多的进程。但是这种方法存在两个缺点。首先，即使在今天，主存也是很昂贵的。其次，程序对存储器容量的需求增长得很快，如同存储器成本下降一样快。所以更大的存储器扩充会导致更大的进程，而不是更多的进程。

另一个解决方案是**交换**，如图8.12所示。我们有一个长期的进程请求队列，通常存储在磁盘上。当主存有空间时，它们一次一个地被调入。当进程完成时，它们被移出主存。现在会出现一种情况，存储器中的进程都不处于就绪状态（例如，所有进程都在等待I/O操作）。这时，处理器不是保持在空闲状态，而是将这些进程之一交换回磁盘，排入中间队列。该队



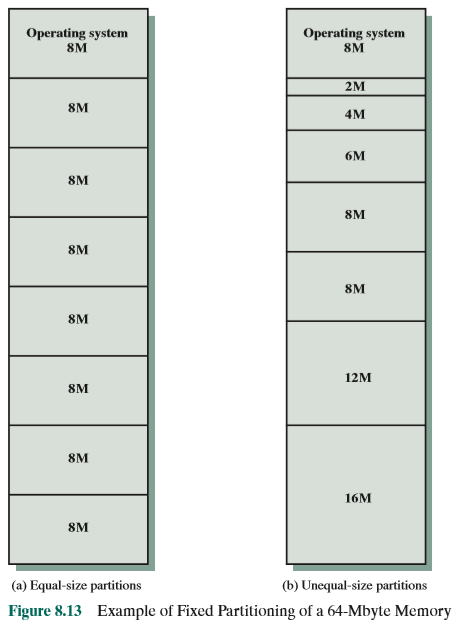
列用以排列临时从内存调出的进程。然后，操作系统从中间队列中调入另一个进程，或者处理长期队列中的一个新进程请求。然后继续执行新到达的进程。

然而，交换是一种I/O操作，因此有可能使问题变得更糟，而不是更好。但是由于磁盘I/O通常是系统上最快的I/O（例如，与磁带或打印机I/O相比），交换通常会提高性能。另一种更复杂的方案是使用虚拟存储器，它比简单的交换更能提高性能，我们将在后面对其做介绍。首先，我们需要介绍分区和分页来。

### 分区

最简单的分区方案是使用固定长度分区，如图8.13所示。注意，虽然分区的长度是固定的，但是它们不需要大小相同。当一个进程加载入内存时，它被放置在可以容纳它的最小可用分区中。

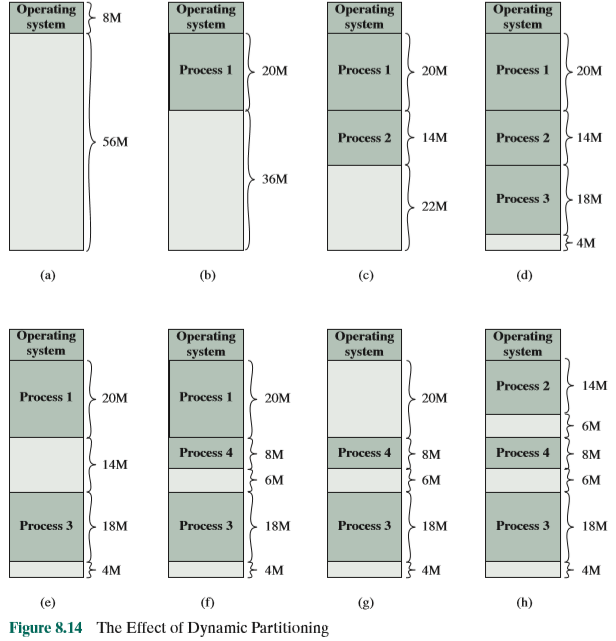
尽管使用了不等的固定长度分区，也会浪费内存。在大多数情况下，进程所需的内存不



会和提供的分区大小完全一样，例如，一个需要3M字节存储空间的进程将被放置在图8.13b的4M字节分区中，这就浪费了另一个进程可以使用的1M字节的空间。

一种更有效的方法是使用可变大小的分区。当一个进程被带入内存时，分配的分区大小可以与进程所需的大小一样。

**例8.2**图8.14给出了一个使用64M字节主的例子。最初，主存除了操作系统占据一些空间外，其余为空(a)。前三个进程被加载，从操作系统结束的地方开始分配，并且占用每个进程（如图b、c、d）恰好需要的空间。这会在内存结尾留下一个“空块”，对于第四个进程来说这个空块太小了。在某个时刻，没有进程进入就绪状态。操作系统交换出进程2(如图e)，这就留出足够的空间来加载新进程，进程4(如图f)。因为进程4比进程2小，所以又会产生另一个小“空块”。随后，某一时刻，再次出现主存中的进程没有一个进入就绪状态的情况，此时进程2进入就绪挂起状态，因为主存中没有足够的空间给进程2，所以操作系统要先换出进程1（如图g），再换入进程2（如图h）。



这个例子表明：这种方法开始时很好，但最终会导致内存中出现许多小的空块。时间越长，存储器中的碎片就会越来越多，从而使存储器利用率下降。解决这个问题的一种技术是**紧缩**。操作系统一次又一次地移动存储器中的进程，将所有空闲块放置在一起，组成一个块。但这个过程很费事，会浪费处理器时间。

在我们考虑克服分区的缺点之前，必须弄清楚一个问题。考虑下图8.14，可以看出，每次交换进程时，都不可能给进程分配到与上次相同的位置。此外，如果使用紧缩技术，则进程在主存中可以移动。主存中的进程由指令和数据组成。指令有两种类型的存储器单元的地址：

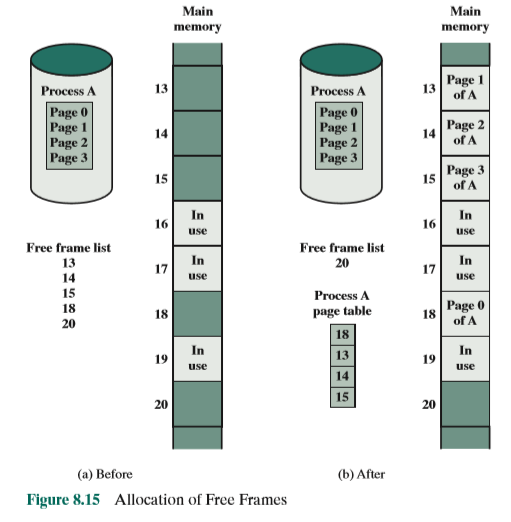
* 数据单元的地址
* 被用于转移指令的指令地址

但是这些地址不是固定的。进程每交换一次，地址都将发生变化。为了解决这个问题，我们把地址分为逻辑地址和物理地址。逻辑地址表示相对于程序起始单元的地址。程序中的指令只包含逻辑地址。物理地址是主存中的实际单元地址。当处理器执行一个进程时，它通过将进程的当前起始单元地址（称为**基址**）加到每个逻辑地址上，自动把逻辑地址转换为物理地址。这中逻辑地址与物理地址的转换机制，是为了满足操作系统的需求而对处理器硬件特性设计的里一个例子。这中硬件特性的精确度取决于使用的存储器管理策略。我们将在本章后面介绍相关的几个例子。

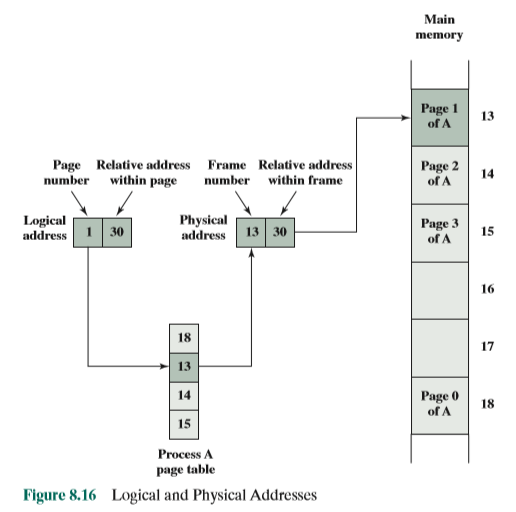
### 分页

不等大小的固定长度分区和可变长度分区，其存储器的使用效率都很低。然而，假设存储器被分成相对小的、相等的固定大小的存储块，并且每个进程也被划分成小的固定长的小块。那么，程序的每个程序块（称为**页**）可以被分配到存储器中可用的存储块（称为**帧**或页帧）。于是，存储器分配进程后在内存中浪费的空间最多只是最后一页的一小部分。

图8.15表示了一个使用页和页帧的例子。在某一给定的时刻，存储器中的一些帧正在被占用，而另一些帧处于空闲状态。空闲帧的列表由操作系统维护。存储在磁盘上的进程A由四个页组成。当操作系统需要加载该进程时，操作系统找到四个空闲帧，并将进程A的四个页面装入到这4个空闲帧中。



现在假设在此例中，没有足够的连续空间帧来存储这个进程。这是否会妨碍操作系统加载进程A？答案是否定的，因为我们可以再一次引入逻辑地址的概念。只是简单的基地址就不再够用了。操作系统为每个进程维护一个页表。页表记录了进程每页的帧地址。在程序中，每个逻辑地址由页号和页内的相对地址组成。回想一下，在简单分区的情况下，逻辑地址是相对于程序起始单元的地址，用一个字表示，处理器负责将逻辑地址转换为物理地址。通过分页技术，逻辑地址到物理地址的转换仍然由处理器硬件来完成。但处理器必须知道如何访问当前进程的页表。处理器使用逻辑地址（页号、相对地址），来生成物理地址（帧号、相对地址）。图8.16中显示了这样的一个示例。



这种方法解决了前面提出的问题。主存分为许多小尺寸的大小相等的帧。每个进程被划分为与帧大小一致的许多页：较小的过程需要较少的页，较大的进程需要较多的页。当一个进程被调入时，它的页被加载到空闲帧中，并且建立页表。

### 虚拟存储器

**请求分页**

分页技术的使用，才产生真正有效的多道程序设计系统。此外，将一个过程分页的简单策略导致了另一个重要概念的产生：虚拟存储器。

为了理解虚拟存储器，我们必须对刚才讨论的分页方案进行改进。改进的方案称为**请求分页**，它意味着进程的每个页面仅在需要时才被调入，即按需调页。

考虑一个大的进程，由一个长程序和许多组数据组成。在任何一小段时间内，程序执行时只需使用其中的一小部分（例如一个子过程）或者一到两组数据。这是我们在附录4A中介绍的局部性原理。显然，在程序被挂起之前只用到其中很少的几页，装入进程的全部页是一种浪费，我们可以通过装入适当的页来更好地利用内存。因此，如果程序转移去执行一个不在主存中的页上的指令，或者程序访问不在主存中的页上的数据，则会触发页**失效**。这就告诉操作系统要调入所需的页面。

于是，在任何时候，进程中只有一小部分在主存中，因此内存可以同时维护更多的进程。此外，因为未使用的页不用在内存中换入换出，从而节省了系统时间。然而，操作系统必须清楚这个管理机制。当它调入一页时，它必须换出另一个页，这被称为**页替换**。如果它换出的页正好是将要使用的页，则又要立即把它调入内存。这种现象的频繁发生被称为**抖动**，这时，处理器花大部分时间处理页面替换而不是执行指令。避免系统抖动是20世纪70年代主要研究的课题，产生了一系列复杂但有效的算法。本质上，操作系统试图根据最近的历史来预测，在不久的将来最不可能被使用的页面。



对页面替换算法的讨论超出了本章的范围。最近最少使用（LRU）是一种比较有效的技术，在第4章高速缓存的替换算法中会介绍这个算法。实际上，LRU很难实现虚拟存储的页替换机制。因此通常是使用一些性能近似于LRU的算法，详细信息请参阅附录K。

使用请求分页，就不需要将整个进程加载到主存中。这将产生一个惊人的结论：进程可能比所有主存都大。程序设计中最基本的限制之一已经被打破。在没有请求分页的情况下，程序员必须非常清楚有多少内存可用。如果正在编写的程序太长，则程序员必须修改结构使程序变成几段，一次装入一段。采用了需求分页，这些工作就交给操作系统和硬件去做。就程序员而言，他或她看到的是一个巨大的存储器，其容量与磁盘存储器有关。

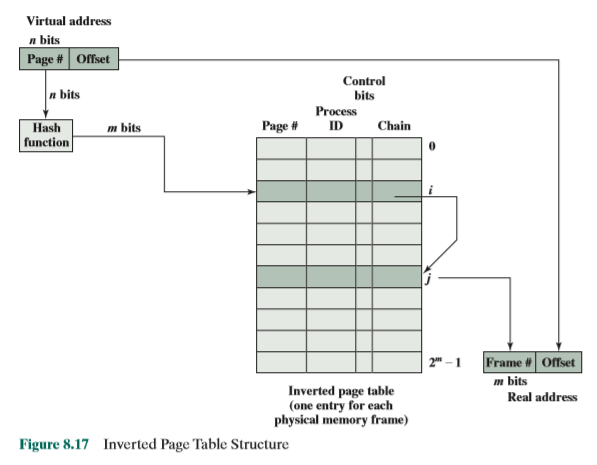
因为进程只在主存中执行，所以主存被称为**实存储器**。但是，程序员或用户能够看到一个大的多的存储器，它分配在磁盘上。因此，后者被称为虚拟存储器。虚拟存储器使多道程序设计更为有效，同时消除了用户使用主存的限制。

页表结构

从存储器中读取一个字的基本机制包括：使用页表将由页号和偏移量组成的虚拟或逻辑地址，转换为由帧号和偏移量组成的物理地址。因为页表的长度是可变的，与进程的长度有关，所以我们不能期望将它存入寄存器中。而必须将它存入主存中。图8.16给出了该方案的硬件实现。当某个特定进程运行时，寄存器保存该进程页表的起始地址。虚拟地址的页号用于索引该页表并寻找相应的帧号。它与虚拟地址的偏移部分相结合来产生实际物理地址。

在大多数系统中，每个进程对应一个页表。但是每个进程都可以占用大量虚拟存储器。例如，在VAX体系结构中，每个进程最多可以有231=2G字节的虚拟存储。使用29=512字节的页，这意味着每个进程需要多达222个页表项。显然，仅分配给页表的存储器容量就高得惊人。为了解决这个问题，大多数虚拟存储器方案将页表存储在虚拟内存中，而不是实存中。这表明对页表也要进行和其他页面一样的分页。当进程运行时，至少有一部分页表（包括当前正在执行的页的页表项）必须在主存中。一些处理器使用两级方案来组织大的页表。在这个方案中，有一个页目录，其中每一项指向一个页表。因此，如果页目录的长度是X，每个页表的最大长度是Y，那么进程可以包括最多X×Y页。通常，一个页表的最大长度被限制为一页。我们将在本章后面讨论Intel x86时，介绍这种两级方案的例子。

实现一级或两级页表的另一种方法是使用倒置页表结构（如图8.17所示）。此方法的不同版本已被用于PowerPC、UltraSPARC和IA-64体系结构。在RT-PC上的Mach操作系统的实现也使用这种技术。



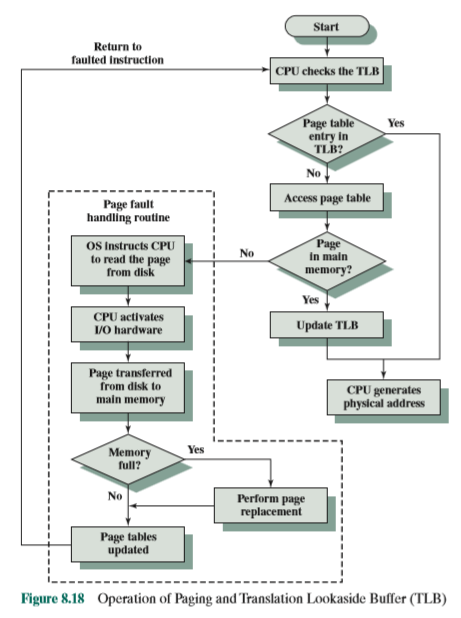
在此方法中，使用简单的散列函数2将虚拟地址的页号部分映射到散列列表中。散列值是一个指向倒置页表的指针，其由页表项组成。在散列表和倒置页表中都有一个指向实存储帧的项，而虚拟页没有。因此，表需要实存的固定部分，无论进程或虚拟页的数量是多少。因为多个虚拟地址可以映射到相同的散列表项，所以使用链技术来管理溢出。散列技术通常位于采用一到两个项组成的短链。此页表的结构称为倒置，因为它按帧号而不是虚拟页号来检索页表项。

### 快表

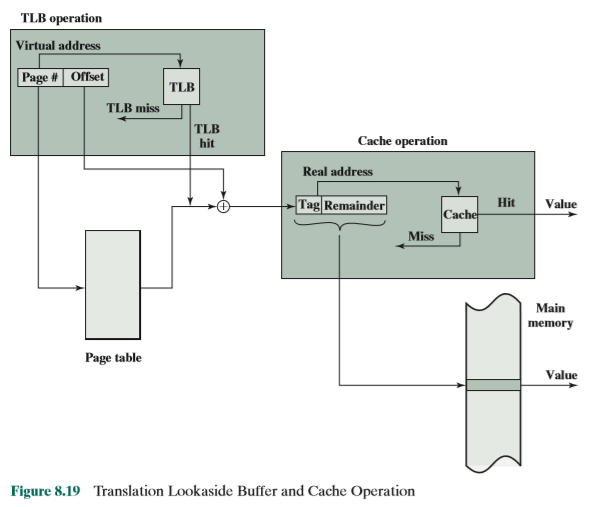
原则上，每次虚拟存储器的访问都会导致两次物理存储器的访问：一次是获取响应的页表项，另一次是获取所需的数据。因此，即使是一个简单的虚拟存储器方案也将使存储器存取时间加倍。为了克服这个问题，大多数虚拟存储器方案使用一个特殊的高速缓存来存放页表项，通常称为快表（TLB）。这个高速缓存功能与存储器中高速缓存工作方式相同，它用来存储最近使用的页的页表项。图8.18显示了TLB使用的流程图。根据局部性原理，大多数虚拟存储器的访问局限在最近使用过的那些页。因此，大多数访问都是此高速缓存中记录的页。对VAX机中TLB的研究表明，该方案可以显著提高性能[CLAR85，SATY81]。

注释---------------------

2散列函数将范围0到M中的数字映射到范围0到N中的数，其中M>N。散列函数的输出用作散列表的索引。由于不止一个输入映射到相同的输出中，因此输入项可以映射到已经被占用的散列表项。在这种情况下，新项必须“溢出”到另一个散列表位置。通常，新项被放置在第一个后续的空项中，并且由原位置提供一个指针指向它。有关散列函数的更多内容，请参见附录L。



注意，虚拟存储机制必须与高速缓存系统（不是TLB高速缓存，而是主存高速缓存）交互。如图8.19所示。虚拟内存通常由页号和偏移量组成。首先，存储器系统查询TLB是否存在匹配的页表项。如果有，则通过组合帧号和偏移量来产生实（物理）地址。如果没有，则从页表中取项。一旦产生了实地址（由标志项和剩余项组成），先查询高速缓存，看是否存在包含该字的块（参见图4.5）。如果有，则把字返回给处理器。如果没有，则从主存中检索单此字。



读者应该能够体会到单个存储器访问中所涉及的处理器硬件的复杂性。虚拟地址被转成实地址，这涉及对页表的访问，该页表可能位于TLB、主存或磁盘上。另外要访问的字可能在高速缓存、主存或磁盘上。如果是在磁盘上，则还需要将包含该字的页加载到主存中，并将它的块加载到高速缓存中。此外，还要更新该页的页表条目项。

### 分段

另一种划分可寻址存储器的方法是分段。分页对程序员是不可见的，其目的是为程序员提供更大的地址空间。但是分段通常对于程序员是可见的，它使组织程序和数据更方便，能将特权、保护属性与指令、数据联系起来。

分段后，存储器由多个地址空间或分段组成。段长度是可变的、可动态分配的。通常，程序员或操作系统为程序和数据分配不同的段。各程序可以有多个程序段和数据段。各个段可以被分配访问权和使用权，存储器访问地址由段号和偏移量组成。

对程序员来说，相对于非分段的地址空间，分段的地址空间有许多优势：

1. 它简化了对增长的数据结构的处理。如果程序员事先不知道数据结构有多大，就没有必要再猜测。该数据结构可以分配它自己的段，并且操作系统可以根据需要动态扩大或收缩这个段。
2. 它允许程序独立修改和重编译，而不需要整个程序重新连接和重装人。即便是使用多个段来完成。
3. 它可以实现进程共享。程序员可以将一个实用程序或有用的数据表放在一个段内，该段可以由其他进程访问。
4. 可以实现段保护。因为一个段可能包含一组定义完好的程序或数据集，所以程序员或系统管理员可以方便地分配访问权限。

这些优点在分页中是没有的，因为分页对程序员是不可见的。另一方面，分页提供了一种有效的存储器管理形式。为了兼顾两者的优点，一些操作系统同时提供了分页和分段两种方式的硬件和操作系统软件支持。

## 8.4 Intel x86存储器管理

自从32位体系结构被引入以来，微处理器已经可以实现复杂的存储器管理机制，这些机制的建立来自于从中型和大型系统中吸取的教训。在许多情况下，微处理器优于先前较大系统版本。因为存储器管理方案是由微处理器硬件厂商开发的，可以被各种操作系统采用，所以它们趋于通用。一个典型的例子是在Intel x86体系结构上使用的体系结构。

### 地址空间

x86体系结构包含分段和分页的硬件支持。同时，两种机制都能禁用，它允许用户从如下4个不同方式中选择其一：

* **不分段不分页存储**：在这种情况下，虚拟地址与物理地址相同。适用于低复杂性、高性能控制器的应用。
* **不分段分页存储**：这里内存是分页的线性地址空间。存储器的保护和管理通过分页来实现。这在某些操作系统（如Berkeley UNIX）中受到青睐。
* **分段不分页存储**：可以把这种存储器看做逻辑地址空间的集合。与分页方法相比，其优点在于，如果需要，它可以提供低至单个字节级别的保护机制。此外，与分页不同，他可以保证当段在内存中的时候，所需要的转换表（段表）在芯片上。因此，分段不分页存储器可以预测时间。
* **分段分页存储**：分段用于存储器逻辑分区，分页用于该分区内的内存再分配。诸如UNIX System V这样的操作系统就采用这种方式。

### 分段

当使用分段时，每个虚拟地址（在x86中称为逻辑地址）由一个16位段号和一个32位偏移量组成。段号中有两位用于保护机制，余下的14位表示一个具体的段。因此，对于未分段的存储器，用户的虚拟存储空间是232=4GB。对于有分段的存储器，用户总的虚拟存储空间为246=64TB。物理地址空间采用32位地址，最大为4GB。

虚拟存储器的容量实际上能大于64TB。这是因为处理器对虚拟地址的译码取决于当前哪个进程是活动的。虚拟地址空间分为两部分：一半（8K段×4GB）是全局性的，被所有进程共享；另一半是局部的，每个进程都不同。

每个段有两种形式的保护机制：优先级和存取属性。有四个优先级，从最高级（级别0）到最低级（级别3）。与数据段相关的优先级称为它的“等级”；与程序段关联的优先级是它的“许可”。只有当执行中程序的许可等级比数据段的特权等级低（更高优先级）或者一样（相同优先级）的时候，程序才可以存取此数据段。

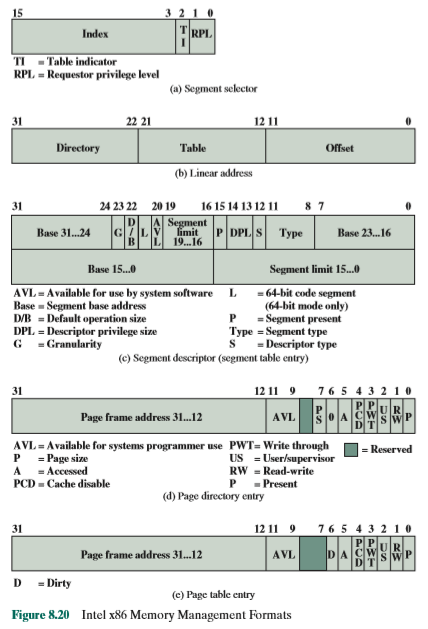
硬件并不规定这些优先级的分配，它取决于操作系统的设计和实现。特权级1通常用于大多数操作系统，第0级只有少量操作系统用于存储器管理、保护和存取控制。剩下两级用于应用。许多系统中，应用程序处于第3级，而第2级未使用。一些特殊的应用程序子系统，由于它们要实现自己的安全机制，因此必须受保护，它们会用到第2级，例如数据库管理系统、办公自动化系统和软件工程环境系统。

除了约束数据段的存取外，优先级机制还用于限制某些指令的使用。一些指令，比如那些处理存储管理寄存器的指令，只能在0级执行。I/O指令只能在操作系统指定的某个级别上执行；通常是第1级。

数据段的存取属性表明是否允许读/写或只读。对于程序段，存取属性表示读/执行或只读访问。

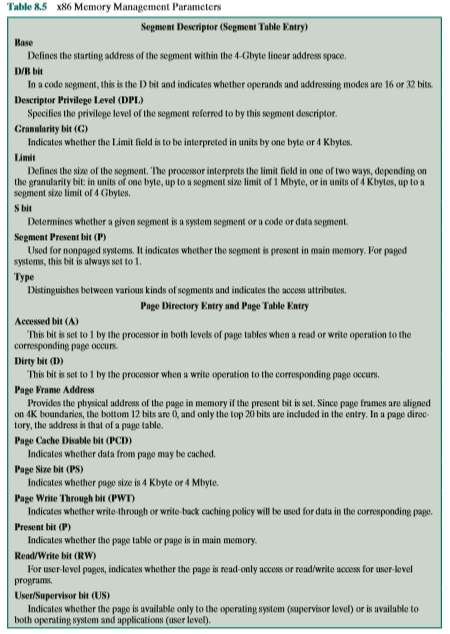
段地址转换机制将虚拟地址映射成线性地址（如图8.20b所示）。虚拟地址由32位偏移量和16位段选择符组成（如图8.20a所示）。获取或存储操作数的指令指定偏移量和存有段选择符的寄存器。段选择符由以下字段组成：

* **段表指示符（TI）**：指示转换使用的是全局段表还是局部段表。



* **段号**：段的编号。段号是段表中的索引。
* **请求优先级（RPL）**：存取请求的优先级。

段表中的每一项由64位组成，如图8.20c所示。域定义如表8.5所示。



### 分页

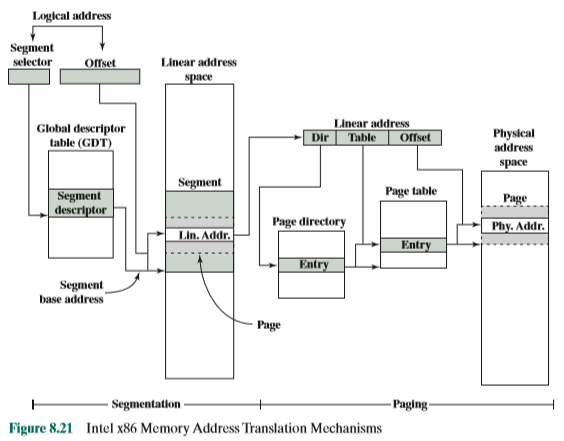
分段是一种可选的特性，可以被禁用。当使用分段时，程序使用的是虚拟地址，需要被转换成线性地址。当分段被禁止时，在程序中直接使用线性地址。无论是否分段，下一步工作都是将该线性地址转换为32位实际地址。

为了理解线性地址的结构，需要知道x86分页机制实际上是一种二级表的检索操作。第1级是一个包含最多1024个项的页目录。它把4GB的线性存储空间分成1024个页组，每个页组都有自己的页表，每个页组的长度为4MB。每个页表最多包含1024个项；每个项对应于一个4KB的页。存储器管理可以有选择地使用页目录，可以对所有进程使用一个页目录、也可以对每个进程使用一个页目录，也可以使用两者的某种组合。当前任务的页目录总是在主存中。页表可能位于虚拟内存中。

图8.20显示了页目录项和页表项的格式，域定义如表8.5所示。注意，访问控制机制可以用于一个页或一个页组。

X86也使用快表。其缓冲区可以保存32个页表项。每次更改页面目录时，都会清空缓冲区。

图8.21显示了分段和分页机制的组合。为了清楚起见，没有显示快表和存储器高速缓存机制。



最后，x86扩展了早期的80386或80486，它提供了两种页面长度。如果将控制寄存器4中的页面长度扩展位（PSE）置为1，那么分页单元允许操作系统程序员定义页长度为4KB或4MB。

当使用4M字节的页时，只有一级表检索。当硬件存取页目录时，页目录项（如图8.20d所示）将PS位置为1。在这种情况下，位9至位21被忽略，而位22至位31定义存储器中一个4MB页的基地址。从而，这是一个单页表。

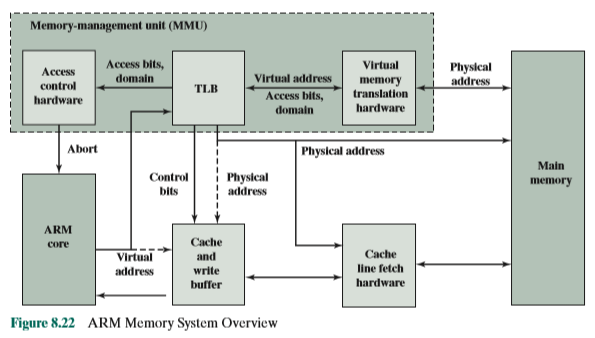
使用4MB页面可减少存储器管理本身对大量主存的需求。使用4K字节的页面，整个4G字节的主存储器仅仅为了存放页面表就需要大约4M字节的存储器；而使用4M字节的页，单个表（长度为4K字节）就足够满足页存储管理的存储需求。

## ARM存储器管理

ARM提供了一种通用的虚拟存储器系统体系结构，可以根据嵌入式系统设计者的需求进行定制。

### 存储器系统组织

图8.22展示了ARM中虚拟内存的内存管理硬件的概述图。如随后所述，虚拟内存转换硬件使用一到两个级别的表，用于将虚拟地址转换为物理地址。快表（TLB）存储最近使用的页表项。如果TLB中有可用的表项，则TLB将直接向主存储器发送物理地址来进行读写操作。如第4章所解释的，处理器和主存之间的数据交换是通过高速缓存进行的。如果采用了逻辑



高速缓存（如图4.7a），则当高速缓存访问失效时，ARM将地址直接提供给高速缓存，同时也将其提供给TLB。如果使用物理高速缓存（如图4.7b所示），则TLB必须向高速缓存提供物理地址。

转换表中的项还包括存取控制位，它决定进程是否可以存取某一段主存。如果访问被拒绝，则访问控制硬件向ARM处理器发送终止信号。

### 虚拟内存地址转换

ARM基于分节或分页来支持存储器访问：

* **超级节（可选）**：由主存的16MB块组成。
* **节**：由主存的1MB块组成。
* **大页**：由主存的64KB块组成。
* **小页面**：由主存的4KB块组成。

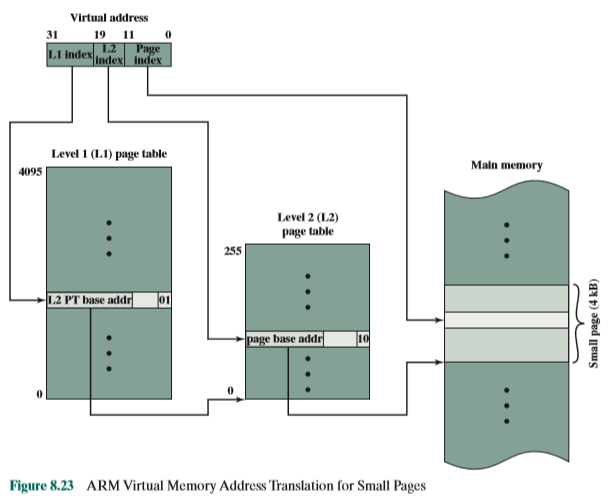
节和超级节可以使用TLB中单个表项映射一大片主存区域。而且存取控制机制可以将小页扩展到1kB子页，将大页扩展到16kB的子页。保存在主存中的转换表有两个级别：

* **第一级表**：包含节和超级节的基本地址和转换属性的第1级描述符；以及转换属性和到第二级表的指针，指针指向大页或小页。
* **第二级表**：包含小页和大页的基本地址和转换属性的第2级描述符。二级表需要1kB的存储空间。

存储器管理单元（MMU）将处理器产生的虚拟地址转换为物理地址来访问主存，并驱动和检查访问权限位。如果发生TLB页缺失，则需要转换地址，首先从一级表中取转换地址。节映射存取仅需要在一级表中取转换地址，而页面映射存取则还需要在第二级表中取转换地址。

图8.23显示了小页的两级地址转换过程。一级（L1）页表包含4K个32位的表项，每个L1表项指向一个二级（L2）页表；二级页表包含256个32位表项。每个L2表项指向主存中一个大小为4KB的页。32位虚拟地址的结构如下：最高12位是指向L1页表的索引。接下来的8位指向响应的L2页表地址。最低的12位指向主存中相应页的一个字节。

大页也使用类似的两页查找。而节和超级节只需要查找L1页表。



### 存储器管理格式

为了更好地理解ARM存储器管理机制，我们考虑其主要的存储器管理格式，如图8.24所示。图中的控制位定义如表8.6所示。

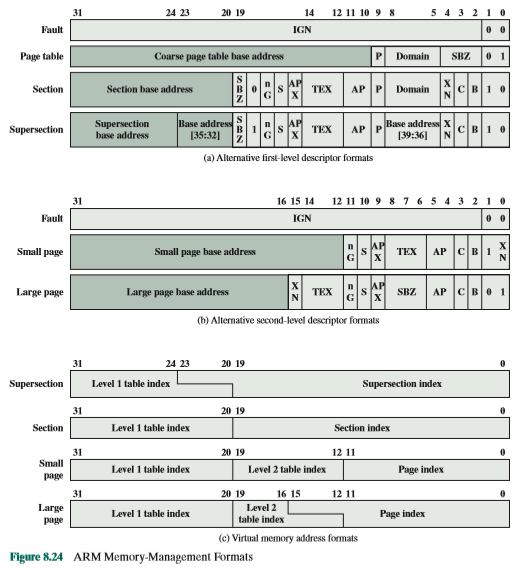
对于L1表，每个表项表示它与1MB虚拟地址的映射关系。每个表项都有四种可选格式：

* 位[1:0] = 00: 对应的虚拟地址未被映射，并且尝试访问它们会产生转换错误。
* 位[1:0] = 01: 表项给出了L2页表的物理地址，它表明了对应虚拟地址的映射关系。
* 位[1:0] = 01: 且第19位为0：表项表示的是对应的虚拟地址的节描述符。
* 位[1:0] = 01：且第19位为1：表项表示的是对应的虚拟地址的超节描述符。

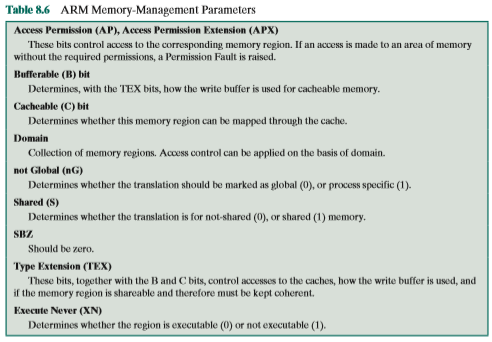
位[1:0] = 11的表项被保留。

对于分页的存储器结构，需要一个二级页表存取。L1页表项的[31:10]包含了一个指向L2页表的指针。对于小页，L2项包含一个20位指针，指向主内存中一个4KB页的基地址。

对于大页，结构更复杂。与小页的虚拟地址结构一样，大页的虚拟地址结构中有12位



指向L1页表、8位指向L2页表。对于64 KB的大页，其虚拟地址的页索引部分必须是16位。为了使这种结构适用于32位的格式，4位页索引域与L2页表域重叠。为了实现这种重叠，ARM要求L2页表中的每个支持大页的页表项复制16次。实际上，如果所有页表项都指向大页，则L2页表中的页表项就从256个减少到16个。然而，一个给定的L2页表能够支持大页和小页的混合，因此需要进行大页项的复制。



对于结构化为节或超级节的存储器结构，需要一级页表存取。对于节结构，L1页表项中的位[31:20]包含一个12位指针，其指向主存中1MB节的基地址。

对于超级节结构，L1页表项的位[31:24]包含一个8位的指针，指向主存中16MB节的基地址。与大页结构一样，需要页表项的复制。在超级节的结构中，虚拟地址的L1页表索引部分与虚拟地址的超级节索引部分有4位重叠。因此，需要16个完全相同的L1页表项。

物理地址范围可通过8个额外的地址位 (位[23:20]和位[8:5])进行扩展。附加位的数目是实现相关的。这些额外的位可以将物理地址扩展为28=256的倍数。因此，对于每一个进程，可寻址的物理存储器范围可能是原来的256倍。

### 存取控制

每个表项中的存取控制位AP控制了某一给定进程对存储区域的存取权限。存储区域可以设定为不可访问、只读或读写。此外，还可以将其设置为特权访问，即只有操作系统可以访问，用户不可以访问。

ARM还引入了“域”的概念，域是一批具有特殊访问权限的节和/或页。ARM体系结构

支持16个域，域特性允许多个进程使用相同的转换表，而且不会相互影响。

每个页表项和TLB项都包含一个字段，该字段用来指明该项所在的域。域存取控制寄存器中的2位的字段控制对每个域的访问。每个字段可以非常快速地标志对一个域的访问是否允许，以便整个存储区域可以非常有效地换入和换出虚拟存储区。可支持的域存取权限有两种：

* **客户**：域的用户（执行程序和存取数据），必须获得构成该域的每个节和每个页的访问权限。
* **管理员**：控制域的行为（域中的当前节和当前页以及该域的存取），并绕过该域中表项的访问权限。

一个程序可以是一些域的客户端，也可以是另外一些域的管理员，而对剩下的域没有访问权限，这就使程序在访问不同的存储器资源时，实现对存储器的灵活保护。

## 8.6关键术语、思考题和习题

### 关键术语

|  |  |  |
| --- | --- | --- |
| 批处理系统 | 多任务化 | 驻留的监控程序 |
| 请求分页 | 核，核心 | 分段 |
| 交互式操作系统 | 操作系统 | 短期调度 |
| 中断 | 页表 | 交换 |
| 作业控制语言 | 分页 | 抖动 |
| 内核 | 分区 | 分时系统 |
| 逻辑地址 | 物理地址 | 快表 |
| 长期调度 | 特权指令 | 实用程序 |
| 中期调度 | 进程 | 虚拟存储器 |
| 存储器管理 | 进程控制块 |  |
| 存储器保护 | 进程状态 |  |
| 多道程序设计 | 实存储器 |  |

### 复习题

8.1 操作系统的两个主要目标是什么？

8.2 列出并简要定义操作系统提供的主要服务。

8.3 列出并简要定义操作系统调度的主要类型。

8.4 多任务和多道程序设计的区别是什么？

8.5 交换的目的是什么？

8.6 如果一个进程可以动态地分配给主存中的不同位置，寻址机制会有什么变化？

8.7 进程执行期间，进程的所有页都需要读入主存中吗？

8.8 主内存中进程的页必须是连续的吗？

8.9 主存中的进程页是否必须按顺序排列？

8.10 快表的作用是什么？

### 习题

8.1 假设我们有一台多道程序设计的计算机，其中每个作业具有一个标识符。在一个计算周期T中，一个作业，一半的时间用于I/O，另一半用于处理器处理。每个作业总共运行N个周期。假设使用了简单的轮转优先权算法，I/O操作可以与处理器操作重叠。定义以下变量：

* 周转时间=实际完成作业时间。
* 吞吐量=平均每个时间周期T内完成的作业数量。
* 处理器利用率=处理器活动（不是等待）时间的百分数。

假设周期T以以下方式分配，同时有一个、两个和四个作业时，计算以上三个量：

(a) I/O占用上半个周期，处理器占用后半个周期。

(b) I/O占用第一个和第四个1/4周期，处理器占用第二个和第三个1/4周期。

8.2 一个I/O受限程序被定义为，如果单独运行，则等待I/O的时间比使用处理器的时间多。处理器受限程序则相反。假设短期调度算法适用那些最近较少使用处理器时间的程序。解释为什么这个算法适用于I/O受限程序，而并不始终拒绝被处理器受限的程序使用？

8.3 一个程序的功能是计算数组A（大小为100×100）某一行的和：

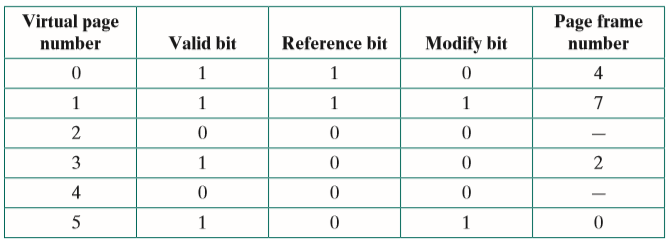


假设计算机使用页面大小为1000字的请求分页机制，主存分配给它们5页帧。如果数组A按行或按列存储在虚拟存储器中，那么页失效率有什么不同吗？解释原因。

8.4 一个容量大小为264字节的存储器采用等长度的分区方案，每个分区的大小为(i)8字节(ii)232字节。进程表存储指向每个驻留进程的分区的指针。在每种情况下指针各需要多少位？

8.5 考虑动态分区方案。请说明，平均而言，内存包含的空块数据数是段的一半。

8.6 假设当前在处理器上执行的进程的页表如下所示。所有的数字都是十进制的，所有的数字都是从零开始计的，所有的地址都是内存字节地址。页面大小为1024字节。



1. 准确描述CPU生成的虚拟地址如何转换为主存的物理地址。
2. 如果有的话，下列每个虚拟地址对应哪个物理地址？（不要尝试处理任何页故障，如果有的话）。
3. 1052
4. 2221
5. 5499

8.7 说明对于给定的进程，存储器中的页数既不能很小也不能很大。

8.8 一个进程按以下顺序访问七个页（A、B、C、D、E、F和G）：

A；C；F；G；F；D；B；E；B；C；D；C；

假设替换算法是最近最少使用（LRU），找出按照此顺序访问时换页的总个数，假设主存初始化为空且有4个页帧。如果有5个页帧，重复以上过程计算。

8.9 在具有虚拟存储器的计算机上执行访问时，遇到以下虚拟页码序列：

9 4 2 5 9 4 1 3 1 2 4 1 3 4 2

假设采用了先进先出的页面替换策略。1...n...6的主存页面容量n，画出页命中率（访问的页面占主存中页面的百分比）到主存页容量n（1≤n≤6）的函数的图。假设主存最初是空的。

8.10 在VAX计算机中，用户页表位于系统空间中的虚拟地址处。把用户的页表存储在虚拟存储器上而不是存储在主存上的优点是什么？缺点是什么？

8.11 假设程序语句

for(I = 1; i <= n; i++)

a[i] = b[i] + c[i];

在页面大小为1000字的存储器上执行。令n = 1000，使用一台具有全范围寄存器到寄存器指令并且可以使用变址寄存器的机器，编写一段机器指令来实现上述语句。然后写出执行期间页访问的序列。

8.12 IBM System/370架构使用段和页两级存储器结构，但其分段法缺少本章前面描述的许多特性。对于基本370架构，页大小是2K字节或4K字节，固定的段大小为64K字节或1M字节。对于370/XA和370/ESA体系结构，页面大小是4K字节，段大小是1M字节。这种方案缺少分段法的什么优点？370的分段法又有什么优点？

8.13 考虑一个同时具有分段和分页的计算机系统。当段在主存中时，其最后一页总有些字是浪费的。此外，段大小为s、页大小为p时，有s/p个页表项。页越小，段中最后一页的浪费就越少，但页表就越大。那么，多大的页可以使总开销最小呢？

8.14 一个计算机有一个高速缓存、主存和用于虚拟存储器的磁盘。如果缓存中有一个字，则需要9ns来存取它。如果它位于主存中而不在cache中，则首先需要80ns将其加载到cache中，然后再开始存取。如果该字不在主存中，则需要8ms从磁盘中获取，然后需要80ns将其复制到cache中，然后重新存取。cache丢失率为9%，主存丢失率为30%。存取此系统上的一个字所需的平均时间是多少纳秒？

8.15 假设一个任务被分成四个大小相等的部分，并且系统为每个段构建一个8项的页描述符表。因此，该系统具有分段和分页的组合。假设页面长度为2K字节。问：

a. 每个分段的最大长度是多少？

b. 任务的最大逻辑地址空间是多少？

c. 假设物理位置00021ABC中的一个元素被此任务存取。则任务产生的逻辑地址的格式是什么？系统的最大物理地址空间是多少？

8.16 假设微处理器能够存取多达232字节的物理主存。它采用了一个最大长度为231字节的分段逻辑地址空间。每条指令包含整个两部分地址。使用外部存储器管理单元（MMU），其管理方案是将固定长度为222字节的物理存储器的连续块分配给段。一个段的起始物理地址总是1024的倍数。画出使用适当数量的MMU将逻辑地址转换为物理地址的外部映射机制的详细连接图，并画出MMU的内部结构图（假设每个MMU包含128个项直接映射段描述符cache），每个MMU如何选择？

8.17 考虑一个分页逻辑地址空间（由32页组成，每页2K字节），映射到1M字节物理内存空间，问：

a. 处理器逻辑地址的格式是什么？

b. 页表的长度和宽度是多少？（忽略“访问权限”位。）

c. 如果物理内存空间减少一半，对页表的影响是什么？

8.18 在IBM的大型机操作系统OS/390中，内核的主要模块之一是系统资源管理器（SRM）。这个模块负责在地址空间（进程）之间分配资源。SRM为OS/390提供了操作系统中独特的精致度。没有任何其他类型的大型机操作系统，当然也没有其他类型的操作系统能够与SRM执行的功能相匹配。资源包括处理器、实存储器和I/O通道。SRM统计处理器、通道和各种关键数据结构的利用率。其目的是在性能监控和分析的基础上提供最佳的性能。建立后面的各种性能目标，并把这些目标作为SRM的指导，SRM基于系统利用率动态地改善安装和工作性能特征。反过来，SRM提供报告以便训练有素的操作员能够改进配置和参数设置来改善用户业务。

这个问题是关于SRM活动的一个例子。实存储器被划分为等长的块，称为帧，其中可能有数千个。每个帧可以保存被称为页面的虚拟内存块。SRM每秒接收大约20次控制信息，每次都会检查每个页帧。如果该页未被访问或更改，则计数器递增1。随着时间的推移，SRM对这些数字进行平均，来确定系统中一页帧未被触动过的平均秒数。这样做的目的是什么？SRM可以采取什么动作？

8.19 根据图8.24所示的每个ARM虚拟地址格式，写出其物理地址格式。

8.20 当主存被分成多个节时，画出一个类似于图8.23的图，用于ARM虚拟存储器转换。