第一章：计算机系统概述

计算机系统基本组成

I. 处理器：控制计算机的操作，执行数据处理功能。当只有一个处理器时，它通常指中央处理器（CPU）。

II. 主存储器：存储数据和程序。

iii. 输入/输出模块：在计算机和外部环境之间移动数据。

iv. 系统总线：为处理器、主存储器和输入输出模块提供通信的设施。

什么是中断？

中断是指计算机的处理机用来处理外来请求或内部错误的一种机制，该机制软硬件结合，使得计算机的处理机能够暂停当前指令系列的执行而转向请求指令系列的执行。

1. 将计算机的处理机正在执行的指令系列称为当前指令系列，当前指令系列通常是用户程序。
2. 将计算机为处理各类突发（非预期）事件请求（I/O 请求,时钟请求，程序错误，硬件错误）而有待执行的指令系列称为请求指令系列，通常称为中断处理程序，是操作系统的一部分。
3. 请求指令系列执行期间，可以被其它事件中断（在允许多重中断的情况下）。
4. 执行请求指令系列完毕后，可以返回被暂停的原始指令系列，也可以不返回（在多道程序设计环境中）。
5. 中断处理程序与社会事务中的应急事件的预案类似。

进程与程序的区别：

进程	程序
1. 进程是执行中的程序，还涉及数据和上下文环境，因而有动态性。	1. 程序具有稳健性。
2. 一个进程可包含多个程序段。	2. 一个程序可对应多个进程（不同时间，不同机器，不同数据，不同上下文）。
3. 进程中的程序是指可执行程序。	3. 广泛意义下的程序可以是任何意义上的程序。
4. 进程中的数据是“值”。	4. 程序中的数据是指数据结构，是“型”，“名”，联系。
5. 进程中涉及上下文（基本性质是与进程本身以及资源有关的状态信息）。	5. 程序不涉及上下文。
6.进程失去逻辑封闭性（原因是多个进程间是异步，并发或并行的），需要一定的互斥机制才能保证逻辑封闭性。	6. 程序具有逻辑封闭性，输入决定输出。

中断处理

中断的发生激活了很多事情，包括处理器硬件中的事件及软件中的事件。

1. 设备给处理器发出一个中断信号。
2. 处理器在响应中断前结束指令系列的执行。
3. 处理器对中断进行测定，确定存在未响应的中断，并给提交中断的设备发送确认信号，确认信号允许该设备取消它的中断信号。
4. 处理器需要把处理权转移到中断程序中去做准备。首先，需要保存从中断点恢复当前程序所需要的信息，要求的最少信息包括程序状态字（PSW）和保存在程序计数器中的下一条执行的指令地址，它们被压入系统控制栈中（参见附录 1B）。
5. 处理器把响应此中断的中断处理器入口地址装入程序的计数器中。
6. 在这一点，与被中断程序相关的程序计数器和 PSW 被保存到系统栈中。此外，还有一些其他信

息被当作正在执行程序的状态的一部分。

7. 中断处理器现在可以开始处理中断，其中包括检查与 I/O 操作相关的信息或其他引起中断的事件，还可能包括给 I/O 设备发送附加命令或应答。
8. 当中断处理结束后，被保存的寄存器值从栈中释放并恢复到寄存器中。
9. 最后的操作是从栈中恢复 PSW 和程序计数器的值，其结果是下一条要执行的指令来自被中断的程序。

处理多重中断的方法：

处理多重中断有两中方法：

1. 当正在处理一个中断时，禁止再发生中断。
   禁止中断的意思是处理器将对任何新的中断请求信号不予理睬。
2. 定义中断优先级，允许高优先级的中断打断优先级低的中断处理器的运行。

第二章：操作系统概述

操作系统通常提供的服务：

1. 程序开发
2. 程序运行
3. I/O 设备访问
4. 文件访问控制
5. 系统访问
6. 错误检测和响应
7. 审计

作为资源管理器的操作系统

1. 操作系统与普通的计算机软件作用相同，也就是说，它由处理器执行的一段程序或一组程序。
2. 操作系统经常会释放可控制，而且必须以来处理器才能恢复。

操作系统开发中的五个主要理论成就

1. 进程
2. 内存管理
3. 信息保护和安全
4. 调度和资源管理
5. 系统结构

进程

进程的概念是操作系统结构的基础。

1. 一个正在执行的程序
2. 计算机中正在运行的程序的一个实例
3. 可分配给处理器并由处理器执行的一个实体
4. 由单一的顺序的执行线索、一个当前状态和一组相关的系统资源所描述的活动单元

计算机系统发展的三条主线：

多道程序批处理操作、分时和实时事务系统，它们在时间和同步中所产生的问题推动了进程概念的发展。

1. 多道程序设计是为了让处理器和 I/O 设备（包括存储设备）同时保持忙状态，以实现最大效率。

其关键机制是，在响应表示 I/O 事务结束的信号时，操作系统将对主存中驻留的不同程序进行处理器切换。

2. 通用的分时。其主要设计目标是能及时响应单个用户的要求，但是由于成本原因，又要可以同时支持多个用户。由于用户反应时间相对较慢，因此这两个目标是可以同时实现的。
3. 实时事务处理系统。在这种情况下，很多用户都在对数据库进行查询或修改，此时系统响应时间是最终要的。

进程的组成：

1. 一段可执行的程序
2. 程序所需要的相关数据（变量、工作空间、缓冲区等）
3. 程序的执行上下文

最后一部分是根本。

执行上下文又称为进程状态，是操作系统用来管理和控制进程所需的内部数据。这种内部信息是分开的，因为操作系统信息不允许被进程直接访问。

上下文包括操作系统管理进程以及处理器正确执行进程所需要的所有信息，包括处理器的内容，如程序计数器和数据寄存器。他还包括操作系统使用的信息，如进程优先级以及进程是否在等待特定 I/O 事件的完成。

操作系统担负的五个基本的存储器管理责任:

1. 进程隔离：操作系统必须保护独立的进程，防止互相干涉数据和存储空间。
2. 自动分配和管理：程序应该根据需要在存储层间动态地分配，分配对程序员来说是透明的。
3. 支持模块化程序设计：程序员应该能够定义程序模块，并且动态地创建、销毁模块，改变模块大小。
4. 保护和访问控制
5. 长期存储

操作系统的几种不同方法和设计要素：

1. 微内核体系结构
2. 多线程
3. 对称多处理
4. 分布式操作系统
5. 面向对象设计

多线程

多线程技术是指把执行一个应用程序的进程划分程可以同时执行的多个线程。

线程和进程的区别：

线程：

可分派的工作单元。它包括处理器上下文（包括程序计数器和栈指针，）和栈中自己的数据区域（为允许子程序分支）。线程顺序执行，并且是可中断的，这样处理器就可以转到另一线程。

进程：

一个或多个线程和相关系统资源（如包含相互据和代码的存储器空间、打开的文件和设备）的集合。这紧密对应于一个正在执行的程序的概念。通过把一个应用程序分解成多个线程，程序员可以在很大程度上控制应用程序的模块性和应用程序相关事件的时间安排。

对称多处理(Symmetric MultiProcessing, SMP)：

对称多处理可定义为具有以下特征的一个独立的计算机系统：

1. 有多个处理器。
2. 这些处理器共享同一个主存储器和 I/O 设备，它们之间通过通信总线或其他内部连接方案互相连接。
3. 所有处理器都可以执行相同的功能（因此称为对称）。

对称多处理操作系统可以调度进程或线程得到所有的处理器运行。对称多处理器结构比单处理器结构具有更多的潜在优势。

第三章 进程描述和控制

在进程执行时，任意给定一个时间，进程都可以唯一地表征为以下元素：

1. 标识符：跟这个进程相关的唯一标识符，用来区别其他进程。
2. 状态：如果进程正在执行，那么进程处于执行态。
3. 优先级：相对于其他进程的优先级。
4. 程序计数器：程序中即将被执行的下一条指令的地址。
5. 内存指针：包括程序代码和进程相关数据的指针，还有其他进程共享内存块的指针。
6. 上下文数据：进程执行的处理器的寄存器中的数据。
7. I/O 状态信息：包括显示的 I/O 请求、分配给进程的 I/O 设备（例如磁带驱动器）和被进程使用的文件列表等。
8. 审计信息：可包括处理器时间总和、使用的时钟数总和、时间限制、审计号等。

导致进程创建的原因：

1. 新的批处理作业：通常位于磁带或磁盘中的批处理作业控制流被提供给操作系统。当操作系统准备接纳新工作是，它将读取下一个作业控制命令。
2. 交互登录：终端用户登录到系统。
3. 操作系统因为提供一项服务而创建： 操作系统可以创建也个进程，代表用户程序执行一个功能，使用户无需等待（如控制打印的任务）。
4. 由现有的进程派生：基于模块化的考虑，或者为了开发并行性，用户程序可以指示创建多个进程。

进程派生

当操作系统为另一个进程的显示请求创建一个进程时，这个动作称为进程派生。

当一个进程派生另一个进程时，前一个进程称为父进程，被派生的进程称为子进程。在典型情况下，相关进程需要相互之间的通讯和合作。

导致进程终止的原因：

1. 正常完成：进程自行执行一个操作系统服务调用，表示它已经结束运行
2. 超过时限：进程运行时间超过规定的时限。
3. 无可用内存：系统无法满足系统需要的内存空间
4. 越界：进程试图访问不允许访问的内存单元。
5. 保护错误：进程试图使用不允许使用的资源或文件，或者试图以一种不正确的方式使用，如：往制度文件中写。
6. 算术错误：进程试图进行被禁止的计算，如除以零或者存储大于硬件可以接纳的数字
7. 时间超出：进程等待某一事件发生的时间超过了规定的最大值。
8. I/O 失败：在输入或输出期间发生错误，如找不到文件、在超出规定的最多努力次数后仍然读写失败。
9. 无效指令：进程试图执行一个不存在的指令。
10. 特权指令：进程试图使用为操作系统保留的指令。
11. 数据误用：错误类型或未初始化的一块数据。
12. 操作员或操作系统干涉：由于某些原因，操作员或操作系统终止进程（例如，如果存在死锁）。

13. 父进程终止：当一个父进程终止时，操作系统可自动终止该进程的所有后代进程。
14. 父进程请求：父进程通常具有终止其任何后代进程的权利。

进程状态转换图

挂起状态进程：

挂起状态进程的概念与不再主存中的进程概念是等价的。

挂起状态进程的特点如下所示：

1. 进程不能立即执行。
2. 进程可能是或不是正在等待一个事件。如果是，阻塞条件不依赖于挂起条件，阻塞事件的发生不会使进程立即被执行。
3. 为阻止进程执行，可以通过代理把这个进程至于挂起状态，代理可以是进程自己，也可以是父进程或操作系统。
4. 除非代理显示地命令系统进行状态转换，否则进程无法从这个状态中转移。

进程挂起的原因：

1. 交换：操作系统需要释放足够的主存空间，以调入并执行处于就绪态的进程。
2. 其他 OS 原因：操作系统可能挂起后台进程或工具程序进程，或者被怀疑导致问题的进程。
3. 交互式用户请求：用户可能希望挂起一个程序的执行，目的是为了调试或者与一个资源的使用进行连接。
4. 定时：一个进程可能会周期性地执行（例如审计或系统监视进程），而且可能在等待下一个时间间隔时被挂起。
5. 父进程请求：父进程可能会希望挂起后代进程的执行，以检查或修改挂起的进程，或者协调不同后代进程之间的行为。

进程控制块 PCB (Process Control Block)：

存放进程的管理和控制信息的数据结构称为进程控制块。它是进程管理和控制的最重要的数据结构，每一个进程均有一个 PCB，在创建进程时，建立 PCB，伴随进程运行的全过程，直到进程撤消而撤消。

在不同的操作系统中对进程的控制和管理机制不同，PCB 中的信息多少也不一样，通常 PCB 应包含如下一些信息。

1、进程标识符 name：

每个进程都必须有一个唯一的标识符，可以是字符串，也可以是一个数字。UNIX 系统中就是一个整型数。在进程创建时由系统赋予。

2、进程当前状态 status：

说明进程当前所处的状态。为了管理的方便，系统设计时会将相同的状态的进程组成一个队列，如就绪进程队列，等待进程则要根据等待的事件组成多个等待队列，如等待打印机队列、等待磁盘 I/O 完成队列等等。

3、进程相应的程序和数据地址，以便把 PCB 与其程序和数据联系起来。

4、进程资源清单。列出所拥有的除 CPU 外的资源记录，如拥有的 I/O 设备，打开的文件列表等。

5、进程优先级 priority：

进程的优先级反映进程的紧迫程序，通常由用户指定和系统设置。UNIX 系统采用用户设置和系统计算相结合的方式确定进程的优先级 。

6、CPU 现场保护区 cpustatus：

当进程因某种原因不能继续占用 CPU 时（等待打印机），释放 CPU，这时就要将 CPU 的各种状

态信息保护起来，为将来再次得到处理机恢复 CPU 的各种状态，继续运行。

7、进程同步与通信机制 用于实现进程间互斥、同步和通信所需的信号量等。

8、进程所在队列 PCB 的链接字 根据进程所处的现行状态，进程相应的 PCB 参加到不同队列中。PCB 链接字指出该进程所在队列中下一个进程 PCB 的首地址。

9、与进程有关的其他信息。 如进程记账信息，进程占用 CPU 的时间等。

进程执行模式：

非特权模式通常称为用户模式，这是因为用户程序通常在该模式下运行；特权模式可称为系统模式、控制模式或内核模式，内核模式指的是操作系统的内核，这是操作系统中包含重要系统功能的部分。

操作系统内核的典型功能：

进程管理：

1. 进程的创建和终止。
2. 进程的调度和分配。
3. 进程切换。
4. 进程同步以及对进程间通信的支持。
5. 进程控制块的管理。

内存管理：

1. 给进程分配地址空间。
2. 交换。
3. 页和段的管理。

I/O 管理：

1. 缓冲区管理。
2. 给进程分配 I/O 通道和设备。

支持功能：

1. 中断处理
2. 审计
3. 监视

进程的创建：

1. 给新进程分配一个唯一的进程标识号。
2. 给进程分配空间。
3. 初始化进程控制块。
4. 设置正确的连接。
5. 创建或扩充其他数据结构。

进程的切换：

进程切换的功能是很简单的。在某一时刻，一个正在运行的进程被中断，操作系统制定另一个进程为运行态，并把控制权交给这个进程。

进程切换的步骤：

1. 保存处理器上下文，包括程序计数器和其他寄存器。
2. 更新当前处于运行太的进程的进程控制块，包括进程状态改变为另一状态（就绪态、阻塞太、就绪/挂起态或退出态）。还必须更新其他相关域，包括离开运行态的原因和审计信息。
3. 把进程的进程控制块移到相应的队列（就绪、在事件 i 处阻塞、就绪/挂起）。
4. 选择另一个进程执行。
5. 更新所选择进程的进程控制块，包括把进程的状态变为运行态。
6. 更新内存管理的数据结构，这取决于如何管理地址转换。

7. 恢复处理器在被选择的进程的最后一次切换出运行态时的上下文，这可以通过载入程序设计数据和其他寄存器以前的值来实现。

进程切换涉及到状态变化，比模式切换需要做更多的工作，显然，模式切换与进程切换是不同的。发生模式切换可以不改变正处于运行态的进程状态，在这种情况 ixa，保存上下文和以后恢复上下文只需要很少的开销。但是如果当前正在运行的进程被切换到另一个状态（就绪、阻塞等），则操作系统必须使其环境产生实质性的变化。

第四章　线程、对称多处理和微内核

多线程：

多线程是指操作系统支持在一个进程中执行多个线程的能力，每个进程中只有一个线程在执行的传统方法（还没有明确线程的概念）称为单线程方法。

资源只能分配给进程（进程调用资源），而一个进程可以有一个或者多个线程。

在一个进程中的每个线程有：

1. 线程执行状态（运行，就绪）
2. 在未运行时保存的线程上下文；可以把线程看做进程中的一个独立的程序计数器在操作。
3. 一个执行的栈。
4. 用于每个线程局部变量静态存储空间。
5. 对所属进程的内存和资源的访问，并与该进程中的其它线程共享这些资源。

在多线程环境中，仍然有与进程相关联的进程控制块和用户地址空间，但是每个线程都有一个独立的栈，还有独立的控制块用于包含寄存器值、优先级和其他与线程相关的状态信息。

线程的重要优点：

1. 在一个已有的进程中创建一个新线程比创建一个进程所需的时间要少许多。
2. 终止一个线程比终止一个进程花费的时间少。
3. 同一进程内线程间切换比进程间切换花费的时间少。
4. 线程提高了不同的执行程序间通讯效率。在大多数操作系统中，独立进程间的通信需要内核的介入，以提供保护和通信所需要的机制。但是由于在同一个进程中的线程共享内存和文件，他们无需调用内核就可以互相通信。

在支持线程的操作系统中，调度和分派是在线程基础上完成的；因此大多数与执行相关的信息可以保存在线程级的数据结构中。

但是，有些活动影响着进程中的所有线程，操作系统必须在进程一级对他们进行管理。挂起涉及到把一个进程的地址空间换出主存并为其他进程的地址空间腾出位置。因为一个进程中的所有线程共享同一个地址空间，所以他们都会同时被挂起。类似的，进程终止也会导致其中的线程的终止。

线程同步：

一个进程中所有线程共享同一个地址空间和诸如打开的文件之类的其他资源。一个线程对资源的任何修改都会影响同一个进程中其他线程的环境。因此需要同步各种线程的活动，以便他们互不干涉且不破坏数据结构。

线程同步的方法主要有互斥锁和信号量:

1．互斥锁方法：

互斥锁是一种简单的加锁方法来控制对共享资源的存取。它只有两种状态：上锁和解锁。在同一时刻只能有一个线程掌握某个已经上锁的互斥锁，拥有上锁状态的线程能够对共享资源进行操作。若其他的线程希望上锁一个已经上锁了的互斥锁，则该线程会被挂起，直到上锁的线程释放互斥锁为止。

2．信号量方法：

信号量本质上是一个非负的整数计数器，它可以用来控制对公共资源的访问，如果信号量的值大于 0，则表示资源可用，否则表示资源不可用。信号量可以用于进程或者线程之间的同步和互斥两种

情况。如果用于互斥，一般只需要设置一个公共信号量 sem。如果用于同步，一般需要设置多个公共/私有信号量，并安排不同的初始值来实现他们之间的顺序执行。

用户级线程：

在一个纯粹的用户级线程中，有关线程管理的所有工作都由应用程序完成，内核没有意识到有线程的存在。

- 使用用户级线程代替内核级线程的优点：
  1. 由于所有线程管理数据结构都在一个进程的用户地址空间中，线程切换不需要内核模式特权，因此，进程不需要为了线程管理而切换到内核模式，者节省了在者两种模式间进行切换（从用户模式到内核模式；从内核模式返回到用户模式）的开销；
  2. 调用可以是应用程序装用的。一个应用程序可能倾向于简单的轮询调度算法，而另一个一个用程序可能倾向于基于优先级的调度算法。调度算法可以去适应应用程序，而不会扰乱底层操作系统的调度器；
  3. 用户级线程可以在任何操作系统中运行，不需要对底层内核进行修改以支持用户级线程。线程库是一组供所有应用程序共享的应用级软件包。
- 用户级线程相对于内核级线程的两个明显缺点：
  1. 在典型的操作系统中，许多系统调用都会引起阻塞。因此，当用户级线程执行一个系统调用时，不仅这个线程会被阻塞，进程中的所有线程都会被阻塞；
  2. 在纯粹的用户级线程策略中，一个多线程应用程序不能利用多处理技术。内核一次只能一个进程分配给一个处理器，因此一次进程中只有一个线程可以执行。实际上，在一个进程内有应用级的多道程序。虽然多道程序会使得应用程序的速度明显提高，但是同时执行部分代码更会使得应用程序受益。

内核级线程：

在一个纯粹的内核级线程软件中，有关线程管理的所有工作都是由内核完成的，应用程序部分没有进行线程管理的代码，只有一个到内核级线程实施的应用程序编程接口（API）。

- 使用内核级线程的优点（克服了用户级线程方法的两个基本缺陷）：
  1. 内核可以同时把同一个进程中的多个线程调度到多个处理器中；
  2. 如果进程中的一个线程被阻塞，内核可以调度同一个进程中的另一个线程。
  3. 内核级线程的另一个优点是内核例程自身也可以使用多线程。
- 相对与用户级线程，内核级线程方法的的主要缺点：
  同一个进程在把控制从一个线程时传送到另一个线程时需要到内核的模式切换。

因此，从表面上看，虽然使用内核级线程多线程技术会比使用单线程的进程有明显的速度上提高，但是使用用户级线程比内核级线程又有额外的提高。不过这个额外的提高是否真的能够实现则要取决与应用程序的性质。

如果应用程序中的大多书线程切换都需要内核模式的访问，那么基于用户级线程的方案不会比基于内核级线程的方案好多少。

第五章　并发性：互斥和同步

操作系统设计中的核心问题是关于进程和线程的管理：

1. 多道程序设计技术：管理但处理器系统中的多个进程；
2. 多处理技术：管理多处理器系统中的多个进程；
3. 分布式处理技术：管理多台分布式计算机系统中多个进程的执行。

并发是所有问题的基础，也是操作系统设计的基础。其产生的问题不仅会出现在多处理环境和分布式处理环境中，也会出现在单处理器的多道程序设计中。

- 并发包括的设计问题：
  1. 进程间通讯
  2. 资源共享与竞争
  3. 多个进程活动的同步以及分配给进程的处理器时间。
- 并发出现的上下文：
  1. 多个应用程序：多道程序设计技术允许在多个活动的应用程序间动态共享处理时间。
  2. 结构化程序设计：作为模块化设计和结构化程序设计的扩展，一些应用程序可以有效地设计成一组并发进程；
  3. 操作系统结构：同样的结构化程序设计有点可以用于系统程序员，且我们已经知道操作系统滋生常常作为一组进程或线程来实现。

和并发相关的关键术语：

临界区：是一段代码，在这段代码中进程将访问共享资源，当另一个进程已经在这段代码中运行时，这个进程就不能在这段代码中执行。

死锁： 两个或两个以上的进程因为其中的每个进程都在等待其他进程昨完某些事情而不能继续执行，这样的情形叫死锁。

活锁： 连个或两个以上的进程为了响应其他进程中的变化而继续改变自己的状态但不做有用的工作，这样的情形叫做活锁。

互斥： 当一个进程在临界区访问共享资源时，其他进程不能进入该临界区访问任何共享资源，这种情形叫互斥。

竞争条件： 多个线程或者进程在读写一个共享数据时结果依赖于他们执行的相对时间，这种情形叫做竞争条件。

饥饿： 是指一个可运行的进程尽管能继续执行，但被调度器无限期地忽视，而不能被调度执行的情况。

互斥的要求：

为了提供对互斥的支持，必须满足一下要求：

1. 必须强制实施互斥：在具有关于相同资源或共享对象的临界区的所有进程中，一次只允许一个进程进入临界区。
2. 一个在非临界区停止的进程必须不干涉其他的进程。
3. 决不允许出现一个需要访问临界区的进程被无限延时的情况，即不会出现死锁和饥饿。
4. 当没有进程在临界区中时，任何需要进入临界区的进程必须能够立即进入。
5. 当相关进程的速度和处理器的数目没有任何要求和限制。
6. 一个进程驻留在临界区中的时间必须是有限的。

生产者消费者：

有一个或多个生产者产生某种类型的数据（记录，字符），并放置在缓冲区中；有一个消费者从缓冲区中取数据，每次取一项；系统保证避免对缓冲区的重复操作，也就是说，在任何时候只有一个代理（生产者或消费者）可以访问缓冲区。

读者写者问题：

有一个许多进程共享的数据区，这个数据区的进程（reader）和一些只往数据区中写数据的进程（writer）；此外还需要如下条件：

1. 任意多的读进程可以同时读这个文件；
2. 一次只有一个进程可以往文件中写；
3. 如果一个写进程正在往文件中写时，则禁止任何进程读文件；

生产者-消费者&&读者-写者

读者-写者	生产者-消费者
1. 一次可以有多个读者同时读取	只有一个消费者去消费
2. 一次只允许一个写者进行写	可以有多个生产者

第六章 并发性：死锁和饥饿

第七章 内存管理

内存管理的五点需求：

重定位
保护
共享
逻辑组织
物理组织

内存管理技术：（具体见课本 P221 表 7.1）

动态分区的放置算法：

最佳适配：选择与要求的大小最接近的块。
首次适配：从开始扫描内存，选择大小足够的第一个可用块；
临近适配：从上一次放置的位置开始扫描内存，选择下一个大小足够的可用块。

三种算法的优劣：

分区式存储管理

早期的单用户、单任务的操作系统将内存空间简单地分为两个区域：系统区和用户区。操作系统使用系统区；应用程序则装入到用户区，并使用用户区全部空间。这种方式管理简单，但浪费内存空间。

为了支持多个程序并发执行，现代操作系统引入了分区式存储管理。内存被分为若干个区域，操作系统占用其中一个分区，其余的分区则提供给应用程序使用，每个应用程序占用其中一个或几个分区。

根据分区的大小是否固定，可以将分区式存储管理机制分为固定分区和动态分区两种类型。动态分区在进程申请内存空间时按其要求的容量分配内存，或根据进程的要求在其执行过程中动态改变分区大小。

分区存储管理的优点是易于实现，但缺点是容易造成空间浪费，产生碎片。

交换技术和分页技术

根据程序的局部性原理，在一个较短的时间间隔内，程序所访问的存储器地址在很大比例上集中在存储器地址空间的很小范围内。交换技术正是利用了程序的局部性原理实现多任务并发环境中的存储管理。 交换过程由换入和换出两个过程组成：换入过程将外存交换区的数据和程序代码换至内存，而换出过程将内存中的数据换到外存交换区中。

操作系统将暂时不执行的程序代码保存在外存中，并将这些进程排入进程请求的长期调度队列。队列中的一部分进程被调到主存中执行。当由于输入/输出操作等原因使得存储器中无进程处于就绪状态时，操作系统将部分进程换出至外存，并排入中期队列。腾出的内存空间则换入中期队列或长期队列中的一个可执行的进程。

交换技术的优点是增加了并发运行的进程数目。缺点是换入和换出操作增加了处理机的时间开销；而且交换的单位为整个进程的地址空间，没有考虑程序执行过程中地址访问的统计特性。

交换技术和早期采用的覆盖技术一样，虽然都是从逻辑上利用外存扩大主存空间，但并没有将主存和外存组成一个有机的整体。

分页技术引申出一种非常重要的存储管理策略——虚拟存储器(简称虚存)。在存储管理部件(MMU)的

支持下，虚拟存储器技术可以彻底解决存储器的调度与管理问题。

第八章　虚拟内存

分页和分段的特点：（见课本 P240 表 8.1）

关于替换策略的算法：（课本 P256->…… 图 8.16）

虚拟存储器的基本概念

实地址与虚地址

用户编制程序时使用的地址称为虚地址或逻辑地址，其对应的存储空间称为虚存空间或逻辑地址空间；而计算机物理内存的访问地址则称为实地址或物理地址，其对应的存储空间称为物理存储空间或主存空间。程序进行虚地址到实地址转换的过程称为程序的再定位。

虚存的访问过程

虚存空间的用户程序按照虚地址编程并存放在辅存中。程序运行时，由地址变换机构依据当时分配给该程序的实地址空间把程序的一部分调入实存。每次访存时，首先判断该虚地址所对应的部分是否在实存中：如果是，则进行地址转换并用实地址访问主存；否则，按照某种算法将辅存中的部分程序调度进内存，再按同样的方法访问主存。由此可见，每个程序的虚地址空间可以远大于实地址空间，也可以远小于实地址空间。前一种情况以提高存储容量为目的，后一种情况则以地址变换为目的。后者通常出现在多用户或多任务系统中：实存空间较大，而单个任务并不需要很大的地址空间，较小的虚存空间则可以缩短指令中地址字段的长度。

虚存机制要解决的关键问题

(1)调度问题：决定哪些程序和数据应被调入主存。

(2)地址映射问题：在访问主存时把虚地址变为主存物理地址（这一过程称为内地址变换）；在访问辅存时把虚地址变成辅存的物理地址（这一过程称为外地址变换），以便换页。此外还要解决主存分配、存储保护与程序再定位等问题。

(3)替换问题：决定哪些程序和数据应被调出主存。

(4)更新问题：确保主存与辅存的一致性。

在操作系统的控制下，硬件和系统软件为用户解决了上述问题，从而使应用程序的编程大大简化。

页式虚拟存储器

页式虚存地址映射

页式虚拟存储系统中，虚地址空间被分成等长大小的页，称为逻辑页；?鞔婵占湟脖环殖赏　　笮〉囊常　莆?物理页。相应地，虚地址分为两个字段：高字段为逻辑页号，低字段为页内地址（偏移量）；实存地址也分两个字段：高字段为物理页号，低字段为页内地址。通过页表可以把虚地址（逻辑地址）转换成物理地址。

在大多数系统中，每个进程对应一个页表。页表中对应每一个虚存页面有一个表项，表项的内容包含该虚存页面所在的主存页面的地址（物理页号），以及指示该逻辑页是否已调入主存的有效位。地址变换时，用逻辑页号作为页表内的偏移地址索引页表（将虚页号看作页表数组下标）并找到相应物理页号，用物理页号作为实存地址的高字段，再与虚地址的页内偏移量拼接，就构成完整的物理地址。现代的中央处理机通常有专门的硬件支持地址变换。

内页表和外页表

页表是虚地址到主存物理地址的变换表，通常称为内页表。与内页表对应的还有外页表，用于虚地址与辅存地址之间的变换。当主存缺页时，调页操作首先要定位辅存，而外页表的结构与辅存的寻址机制密切相关。例如对磁盘而言，辅存地址包括磁盘机号、磁头号、磁道号和扇区号等。

段式虚拟存储器和段页式虚拟存储器

段式虚拟存储器

段是按照程序的自然分界划分的长度可以动态改变的区域。通常，程序员把子程序、操作数和常数等不同类型的数据划分到不同的段中，并且每个程序可以有多个相同类型的段。在段式虚拟存储系统中，虚地址由段号和段内地址（偏移量）组成。虚地址到实主存地址的变换通过段表实现。

每个程序设置一个段表，段表的每一个表项对应一个段。每个表项至少包含下面三个字段：

1. 有效位：指明该段是否已经调入实存。
2. 段起址：指明在该段已经调入实存的情况下，该段在实存中的首地址。
3. 段长：记录该段的实际长度。设置段长字段的目的是为了保证访问某段的地址空间时，段内地址不会超出该段长度导致地址越界而破坏其他段。

段表本身也是一个段，可以存在辅存中，但一般驻留在主存中。

段式虚拟存储器有许多优点：

1. 的逻辑独立性使其易于编译、管理、修改和保护，也便于多道程序共享。
2. 段长可以根据需要动态改变，允许自由调度，以便有效利用主存空间。

段式虚拟存储器也有一些缺点：

1. 为段的长度不固定，主存空间分配比较麻烦
2. 容易在段间留下许多外碎片，造成存储空间利用率降低。
3. 由于段长不一定是 2 的整数次幂，因而不能简单地像分页方式那样用虚地址和实地址的最低若干二进制位作为段内偏移量，并与段号进行直接拼接，必须用加法操作通过段起址与段内偏移量的求和运算求得物理地址。

因此，段式存储管理比页式存储管理方式需要更多的硬件支持。

段页式虚拟存储器

段页式虚拟存储器是段式虚拟存储器和页式虚拟存储器的结合。

实存被等分成页。每个程序则先按逻辑结构分段，每段再按照实存的页大小分页，程序按页进行调入和调出操作，但可按段进行编程、保护和共享。

第九章　单处理器调度

处理器调度的类型：

处理器调度的目标是以满足系统个目标（如响应时间，吞吐率，处理器效率）的方式，把进程指定到一个或多个处理器中执行。在许多系统中，这个调度互动分成是哪个独立的功能：长程调度、中程调度、短程调度。

1. 长程调度：
2. 中程调度：
3. 短程调度：