第一章

★**1.**操作系统的概念： 通常把操作系统定义为用以控制和管理计算机系统资源方便用户使用的程序和数据结构的集合。

★**2.**操作系统的基本类型：批处理操作系统、分时操作系统、实时操作系统、个人计算机操作系统、网络操作系统、分布式操作系统。

①批处理操作系统

特点：

用户脱机使用计算机

成批处理

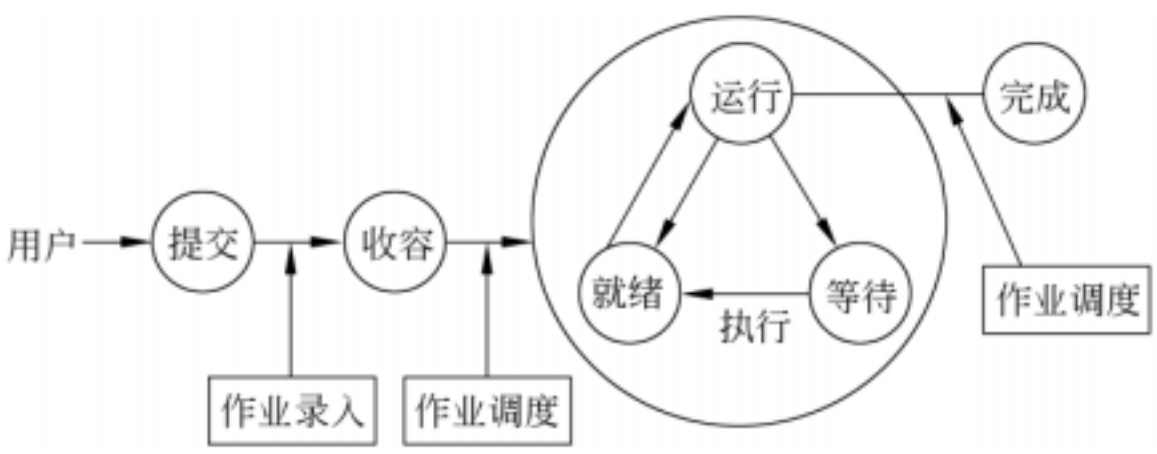
多道程序运行

优点：

由于系统资源为多个作业所共享，其工作方式是作业之间自动调度执行。并在运行过程中用户不干预自己的作业，从而大大提高了系统资源的利用率和作业吞吐量。

缺点：

无交互性，用户一旦提交作业就失去了对其运行的控制能力；而且是批处理的，作业周转时间长，用户使用不方便。



批处理系统中作业处理及状态

②分时操作系统 **(Time Sharing OS)**

分时操作系统是一个联机的多用户交互式的操作系统，如 **UNIX** 是多用户分时操作系统。

分时计算机系统：由于中断技术的使用，使得一台计算机能连接多个用户终端，用户可通过各自的终端使用和控制计算机，我们把一台计算机连接多个终端的计算机系统称为分时计算机系统，或称分时系统。

分时技术：把处理机的响应时间分成若干个大小相等（或不相等）的时间单位，称为时间片（如 **100** 毫秒），每个终端用户获得 **CPU**，就等于获得一个时间片，该用户程序开始运行，当时间片到（用完） ，用户程序暂停运行，等待下一次运行。

特点：

人机交互性好：在调试和运行程序时由用户自己操作。

共享主机：多个用户同时使用。

用户独立性：对每个用户而言好象独占主机。

③实时操作系统 **(real-time OS)**

实时操作系统是一种联机的操作系统，对外部的请求，实时操作系统能够在规定的时间内处理完毕。

特点：

有限等待时间

有限响应时间

用户控制

可靠性高

系统出错处理能力强

设计实时操作系统要考虑的一些因素：

（**1**）实时时钟管理

（**2**）连续的人 — 机对话

（**3**）过载

(4) 高度可靠性和安全性需要采取冗余措施。

④通用操作系统

同时兼有多道批处理、分时、实时处理的功能，或其中两种以上的功能。

⑤个人计算机上的操作系统

个人计算机上的操作系统是联机的交互式单用户操作系统，目前在个人计算机上使用的操作系统以 **windows** 系列和 **linux** 系统为主。

⑥网络操作系统

特征：

（**1**）计算机网络是一个互连的计算机系统群体。这些计算机在物理上是分散的。

（**2**）这些计算机是自治的，每台计算机有自己的操作系统，各自独立工作，它们在网络协议控制下协同工作。

（**3**）系统互连要通过通信设施（硬件、软件）来实现。

（**4**）系统通过通信设施执行信息交换、资源共享、互操作和协作处理。

⑦分布式系统 **(Distributed System)**

特征：

（**1**）功能的分布

（**2**）坚强性

（**3**）高可靠性

★3. 操作系统的功能

处理机管理、存储管理（内存分配、存储保护、内存扩充） 、设备管理（通道、控制器、输入输出设备的分配与管理，设备独立性）、信息管理（文件系统管理） 、用户接口（程序一级的接口、作业一级的接口） 。

4.通道和中断技术

通道：用于控制 **I/O** 设备与内存间的数据传输。启动后可独立于 **CPU** 运行，实现 **CPU** 与 **I/O** 的并行。

通道有专用的 **I/O** 处理器，可与 **CPU** 并行工作

可实现 **I/O** 联机处理

中断是指 **CPU** 在收到外部中断信号后，停止原来工作，转去处理该中断事件，完毕后回到原来断点继续工作。

中断处理过程：中断请求，中断响应，中断点（暂停当前任务并保存现场） ，中断处理例程，中断返回（恢复中断点的现场并继续原有任务

监督程序发展为执行系统 **(executive system)**，常驻内存

★5.多道批处理系统

特点

多道：内存中同时存放几个作业；

宏观上并行运行：都处于运行状态，但都未运行完；

微观上串行运行：各作业交替使用 **CPU** ；

优点：

资源利用率高： **CPU** 和内存利用率较高；

作业吞吐量大：单位时间内完成的工作总量大；

缺点：

用户交互性差：整个作业完成后或中间出错时，才与用户交互，不利于调试和修改；

作业平均周转时间长：短作业的周转时间显著增长；

多道程序系统中，要解决的问题：同步互斥、内存不够、使用效率、内存保护

6.计算机硬件：

构成计算机的基本硬件元素：处理器、存储器、输入输出控制与总线、外部设备。

与操作系统相关的几种主要的寄存器

数据寄存器

地址寄存器

条件码寄存器

程序计数器

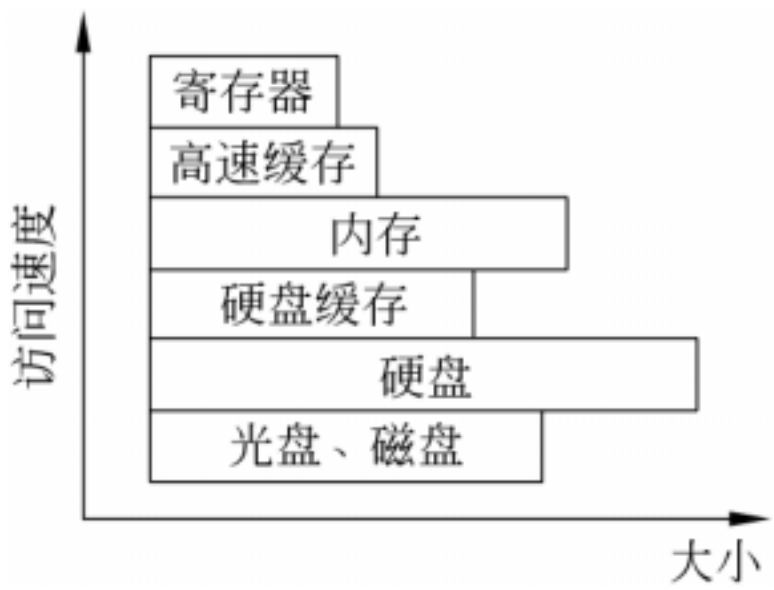
指令计数器

程序状态字 **PSW**

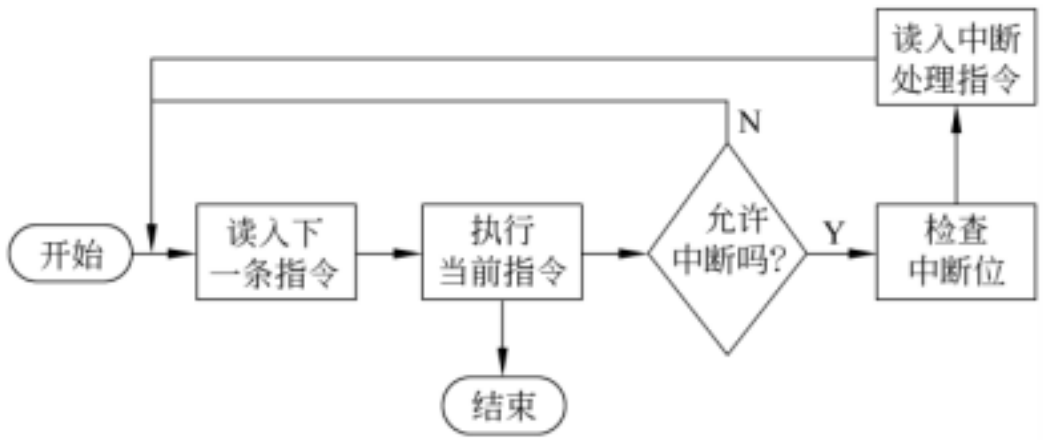
中断现场保护寄存器

过程调用用堆栈

存储器的访问速度



指令的执行和中断



操作系统的启动

启动电源 —— 产生中断信号 —— 触发 **CPU** 中的一段指令发现操作系统引导区位置 —— 导入内存执行 —— 操作系统程序加载到内存制定区域 —— 初始化硬件 ……

7.算法

begin …**:end** 算法的开始于结束

repeat 操作 … **..until** 条件 当“ 条件 ”未被满足时重复所描述的“ 操作 ”

while 条件 **do** 操作 …… **.od** 当“ 条件 ”满足时，进行相应的“ 操作 ”

if 条件 **then** 操作 **else** 操作 **fi** 满足“ **if** ”所指的“ 条件 ”时，进行“ **then** ”后的相关“ 操作 ”，否则完成“ **else**”后的相关操作。

第二章

★**1.**作业：在一次应用业务处理过程中，从输入开始到输出结束，用户要求计算机所做的有关该次业务处理的全部工作称为一个作业。

作业由不同的顺序相连的作业步组成，作业步是一个作业的处理过程中计算机所做的相对独立的工作。

2.作业的组织：

作业由三部分组成，即程序、数据和作业说明书。作业中包含的程序和数据完成用户所要求的业务处理工作，作业说明书则体现用户的控制意图。

★由作业说明书在系统中生成一个称为作业控制块（ **JCB**）的表格， **JCB** 包括：作业名、估计执行时间、优先数（用于调度）、作业说明书文件名、程序类型、资源要求 **(**静态申请和动态申请 **)**、作业状态（提交后各执行完成） 。

作业说明书包括：作业基本情况描述（用户名、作业名、使用语言名、允许最大处理时间等） 、作业控制描述（控制方式、操作顺序、出错处理等） 、作业资源要求描述（要求处理时间、内存空间、外设类型和数量、处理及优先级、库函数或实用程序等） 。

★**3.**如何控制作业

①联机输入输出方式

联机输入输出方式大多用在交互式系统中，用户与系统通过交互式会话输入输出作业。在联机输入输出方式中，外围设备直接与主机相连接。

②脱机输入输出方式

脱机输入又称为预输入方式，利用低档个人计算机作为外围处理机进行输入输出处理。

③直接耦合方式

把主机与低档外围通过一个公用的大容量外存直接耦合起来。

④ **SPOOLING** 系统（外围设备同时联机操作）

多台外围设备通过通道或 **DMA** 器件和主机与外存连接起来。

⑤网络联机方式

网络联机方式以上述几种输入输出方式为基础。当用户通过计算机网络中的某一台设备对计算机网络中的另一台主机

进行输入输出操作时，就构成了网络联机方式。

4.系统调用

系统调用大致可分为　**6** 类：

（**1**）设备管理：该类系统调用被用来请求和释放有关设备以及启动设备操作等。

（**2**）文件管理：包括对文件的读、写、创建和删除等。

（**3**）进程控制：包括进程创建、进程执行、进程撤销、进程等待和执行优先级控制等。

（**4**）进程通信：该系统调用被用在进程之间传递消息或符号。

（**5**）存储管理：包括调查作业占据内存区的大小、获取作业占据内存区的始址等。

（**6**）线程管理：包括线程的创建、调度、执行、撤销等。

系统调用的实现：当用户使用系统调用时，产生一条相应的指令，处理机在执行到该指令时发生相应的中断，并发出有关信号给该处理机制。该处理机制在收到了处理机发来的信号后，启动相关的处理程序去完成该系统调用所要求的功能。

陷进处理机构：在系统中为控制系统调用服务的机构称为陷进处理机构。

陷进指令：把由于系统调用引起处理机中断的指令称为陷进指令。

第三章

1. 程序的并发执行

程序用来描述计算机所完成的独立功能，并在时间上严格地按前后次序相继地进行计算机操作序列集合，是一个静态概念。

个程序由若干个程序段组成，而这些程序段的执行必须是顺序的，这种程序执行的方式就称为程序的顺序执行。

程序顺序执行的特点：

1. 顺序性

处理机严格按照程序所规定的顺序执行，即每个操作必须在下一个操作开始之前结束。

2. 封闭性

程序一旦开始执行，其计算结果不受外界的影响，当程序的初始条件给定之后，其后的状态只能由程序本身确定，即只有本程序才能改变它。

3. 可再现性

程序执行的结果与初始条件有关，而与执行时间无关。即只要程序的初始条件相同，它的执行结果是相同的，不论它在什么时间执行，也不管计算机的运行速度。

多道程序系统中程序执行环境的变化

执行环境的特点：

（1）独立性

在多道环境下执行的每道程序都是逻辑上独立的。

（2）随机性

程序和数据的输入和执行开始时间都是随机的。

（3）资源共享

软硬件资源的有限性导致资源共享。

程序并发执行：若干个程序段同时在系统中运行，这些程序的执行在时间上是重迭的，一个程序段的执行尚未结束，另一个程序段的执行已经开始，即使这种重迭是很小的，也称这几个程序段是并发执行的。

2.★ .进程：进程是一个程序对某个数据集的执行过程，是分配资源的基本单位。

进程和程序的区别与联系：

①程序是指令的集合，是静态的概念。　　进程是程序在处理机上的一次执行的过程，是动态的概念。程序可以作为软件资料长期保存。进程是有生命周期的。

②进程是一个独立的运行单位，能与其它进程并行（并发）活动。而程序则不是。

③进程是竞争计算机系统有限资源的基本单位，也是进行处理机调度的基本单位。

④不同的进程可以包含同一程序，只要该程序所对应的数据集不同。

作业和进程的关系

作业是用户需要计算机完成某项任务时要求计算机所做工作的集合。而进程则是已提交完毕程序的执行过程的描述，是资源分配的基本单位。

其主要区别如下：

作业是用户向计算机提交任务的任务实体。

一个作业可由多个进程组成。

作业的概念主要用于批处理系统中。

进程描述

在系统中一个进程存在：进程控制块 **PCB**、有关程序段、数据结构集

① 进程控制块 **PCB (Process Control Block)**

包含一个进程的描述信息、控制信息及资源信息，有些系统还有进程调度等待所使用的现场保护区。 **PCB** 集中反映一个进程的动态特征。在创建时，建立 **PCB**，并伴随进程运行的全过程，当进程完成其功能后，系统释放 **PCB**，进程也随之消亡

（**1**）描述信息

1、进程名或进程标识号 **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、用户名或用户标识号

每个进程都隶属于某个用户，用户名或用户标识号有利于资源共享和保护

3、家族关系 **process family**

有的系统允许一个进程可创建自已的子进程， 子进程还可以创建， 一个进程往往处在一个家族之中， 就需要记录进程在家族中位置的信息。

（**2**）控制信息

1、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。

2、进程优先级 **priority**

进程的优先级反映进程的紧迫程度，通常由用户指定和系统设置。

3、执行程序开始地址 **start-addr**

4、各种计时信息

进程占用系统资源的情况，不同的系统的处理差别很大。

5、通信信息 **communication information**

是指某个进程在运行的过程中要与其它进程进行通信，该区记录有关进程通信方面的信息。

（**3**）资源管理信息

包括有关存储器的信息、使用输入、输出设备的信息、有关文件系统的信息：

1、占用内存大小及管理用数据结构指针。

2、在某些复杂系统中，还有对换或覆盖用的有关信息。

3、共享程序段大小及起始地址。

4、输入输出设备的设备号，所要传送的数据长度、缓冲区地址、缓冲区长度及使用设备的有关数据结构指针等。

5、指向文件系统的指针及有关标识等。

（**4**）、**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。

②进程上下文实际上是进程执行活动全过程的静态描述。

进程上下文是一个抽象的概念，它包含了每个进程执行过的、执行时的以及待执行的指令和数据，在指令寄存器、堆栈（存放个调用子程序的返回点和参数等） ，状态字寄存器等中的内容。

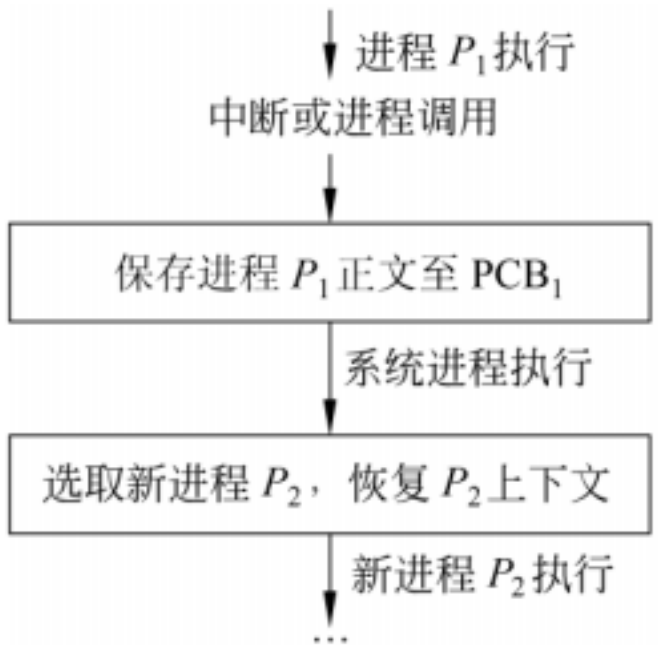
上文：已执行过的进程指令和数据在相关寄存器与堆栈中的内容。

正文：正在执行的指令和数据在相关寄存器与堆栈中的内容。

下文：待执行的指令和数据在相关寄存器与堆栈中的内容。

③进程上下文切换

进程上下文切换发生在不同的进程之间而不是同一个进程内。包含 **3** 个部分，第一部分为保存被切换进程的正文部分（或当前状态）至有关存储区。第二部分操作系统进程中有关调度和资源分配程序执行，并选取新的进程。第三部分则是将被选中进程的原来被保存的正文部分从有关存储区中选出， 并送至有关寄存器或堆栈中， 激活被选中进程执行。



④进程空间和大小

任一进程都有自己的地址空间，把该空间称为进程空间或虚空间。进程空间的大小只与处理机的位数有关。程序的执行都在进程空间内进行。用户程序、进程的各种控制表格等都按一定的结构排列在进程空间中。

在有的系统中进程空间被划分为两部分：用户空间和系统空间。

为了防止用户程序访问系统空间，　造成访问出错，　计算机通过程序状态寄存器等设置不同的执行模式，　即用户模式（用户态）和系统模式（系统态）来进行保护。

3.进程状态及其转换

★进程的三种基本状态：执行状态、就绪状态、等待状态（又称阻塞、挂起、睡眠）

就绪状态（ **Ready)**

存在于处理机调度队列中的那些进程，它们已经准备就绪，一旦得到 **CPU**，就立即可以运行，这些进程所取的状态为就绪状态。（有多个进程处于此状态）

执行状态 **(Running**）

当进程由调度 **/**分派程序分派后，得到 **CPU** 控制权，它的程序正在运行，该进程所处的状态为执行状态。（在系统中，总只有一个进程处于此状态）

等待状态（ **Wait)**

若一个进程正在等待某个事件的发生（如等待 **I/O** 的完成），而暂停执行，这时，即使给它 **CPU** 时间，它也无法执行，则称该进程处于等待状态。

★进程状态转换

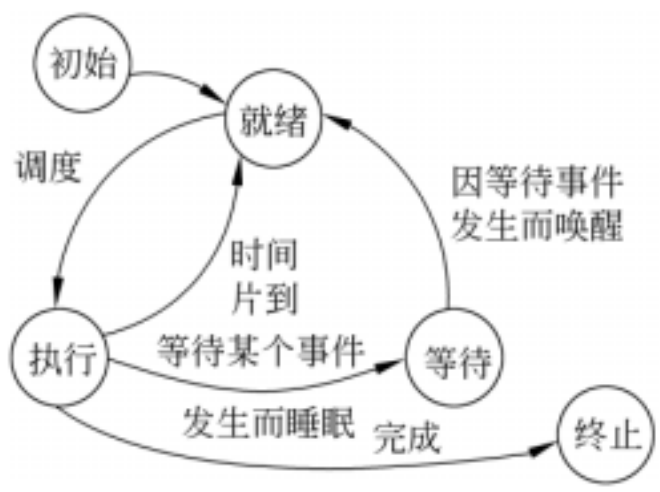
运行到等待　等待某事件的发生（如等待 **I/O** 完成）

等待到就绪　事件已经发生（如 **I/O** 完成）

运行到就绪　时间片到（例如，两节课时间到，下课）

新建进程到就绪　新创建的进程进入就绪状态

就绪到运行　当处理机空闭时，由调度（分派）程序从就绪进程队列中选择一个进程占用 **CPU**。



进程控制：就是系统使用一些具有特定功能的程序段来创建、撤销进程以及完成进程各状态的转换，从而达到多进程高效率并发执行和协调、实现资源共享的目的。

原语：把系统态下执行的某些具有特定功能的程序段称为原语。

用于进程控制的原语有：创建原语、撤销原语、阻塞原语、唤醒原语。

进程创建方式：由系统程序模块统一创建；由父进程创建。进程创建系统调用：　**create(name,priority,start-addr)**

UNIX　系统：　**fork()**

进程撤销：（**1**）该进程已完成所要求的功能而正常终止（　**2**）由于某种错误导致非正常终止（　**3**）祖先进程要求撤销某个子进程。在一般操作系统中进程撤消的系统调用是：　**kill**　**UNIX**　系统中是　**exit()**　如果撤销进程有自己的子进程，则撤销原语先撤销其子进程的　**PCB** 结构并释放子进程所释放的资源后，再撤销当前进程的　**PCB** 结构和释放其资源。

进程的阻塞与唤醒

当一个处在运行状态的进程，因等待某个事件的发生（如等待打印机）而不能继续运行时，将调用进程挂起系统调用，把进程的状态置为阻塞状态，并调用进程调度程序（等于让出处理机）。

进程从运行状态转换成阻塞状态是由进程挂起原语实现的，因此，调用进程挂起操作是在进程处于运行状态下执行的。它的执行将引起等待某事件的队列的改变.

一个正在运行的进程会因等待某事件（例如，等待打印机）的发生，由运行状态转换成阻塞状态，当它等待的事件发生后，这个进程将由阻塞状态转换成就绪状态。这种转换由进程唤醒操作完成。

唤醒一个进程有两种方式：系统进程唤醒、事件发生进程唤醒。

调用进程唤醒操作一般在中断处理、进程通信等过程中。例如，打印机完成中断处理程序，在完成了打印完成的操作后，就去检查等待打印机的队列，若不为空，则调用进程唤醒操作，唤醒一个（或多个）等待打印机的进程。

4.进程互斥

产生互斥的原因：资源共享、进程合作

★临界资源：一次仅允许一个进程使用的资源称为临界资源。

★临界区：每个进程中访问临界资源的那段程序段称为临界区（临界段）。

间接制约：由于共享某公有资源而引起的在临界区内不允许并发进程交叉执行的现象称为有共享公有资源而造成的对并发进程执行速度的间接制约，简称间接制约。

★互斥：在操作系统中，当某一进程正在访问某临界区时，就不允许其它进程进入，否则就会发生**(**后果**)**无法估计的错误。我们把进程之间的这种相互制约的关系称为互斥。

进入临界区的准则：

(1)不能假设各并发进程的相对执行速度；

（**2**）并发进程中的某个进程不在临界区时，它不能阻止其他进程进入临界区；

（**3**）并发进程中的若干个进程申请进入界区时，只能允许一个进程进入；

(4)当有若干个进程欲进入临界区时，应在有限的时间内使其进入。

解决进程互斥的最简单的办法是加锁。

在系统中为每个临界资源设置一个锁位，

1 表示资源可用，

0 表示资源已被占用（不可用）。

这样当一个进程使用某个临界资源之前必须完成下列操作：

1、考察锁位的值；

2、若原来的值是为 “**1**”，将锁位置为 “**0**”（占用该资源）；

3、若原来值是为 “**0**”，（该资源已被别人占用），则转到 **1**。

当进程使用完资源后，将锁位置为 “**1**“，称为开锁操作。

5.信号量与 **P**、**V** 原语

★信号量 **sem**：是一个整数，在 **sem** 大于等于零时，代表可供并发资源使用的资源实体数，但 **sem** 小于零时则表示正在等待使用临界区的进程数。**sem** 代表资源的实体。在实际应用中应准确地说明 **sem** 的意义和初值。

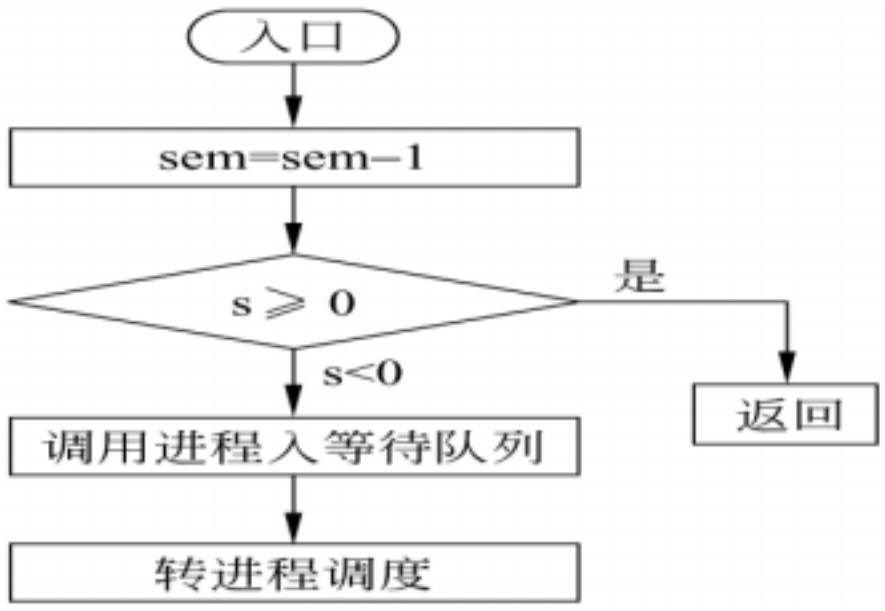
★**P** 操作：

（**1**） **sem** 减 **1**；

（**2**）若 **sem** 减 **1** 后仍大于等于 **0**，则进程继续执行；

（**3**）若结果小于 **0**，则该进程挂起。

注：挂起该进程包括：保留调用进程 **CPU** 现场；置 “等待 ”状态；入等待队列；转进程调度；



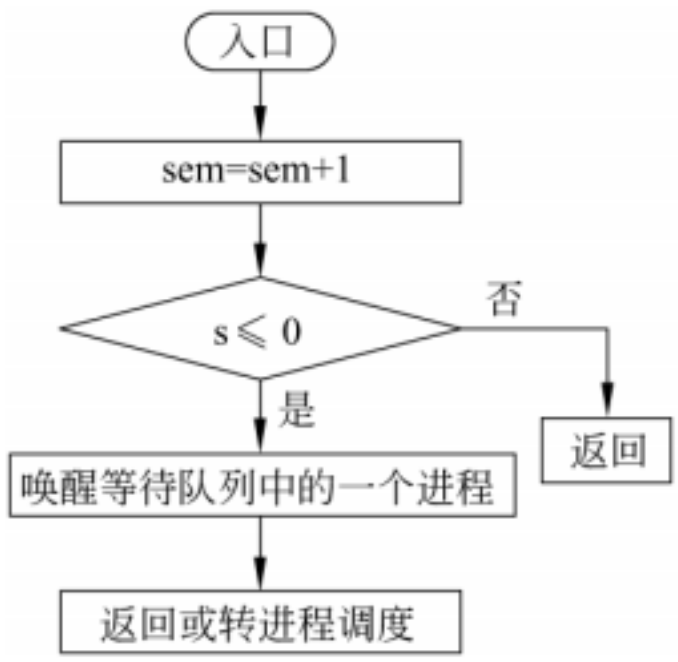

```
算法：V
输入：s
输出：无
{
  s++;
  if(s <= 0)
    唤醒等待S的进程;
  }
```

V 操作：

（**1**） **s** 值加 **1**；

（**2**）若相加结果大于 **0**，进程继续执行；

（**3**）否则，唤醒一个（或多个）等待该信号灯的进程，然后本进程继续执行或转进程调度。




```
算法：V
输入：s
输出：无
{
  s++;
  if(s <= 0)
    唤醒等待S的进程;
  }
```

★ **P**、**V** 原语实现互斥的原理

当一个进程想要进入临界区时，它必须先执行 **P** 原语操作以将信号量 **sem** 减 **1**。在一个进程完成对临界资源的操作后，它必须执行 **V** 原语操作以释放它占用的临界资源。由于信号量初始值为 **1**，所以，任一进程在执行 **P** 原语操作之后将 **sem** 的值变为 **0**，表示该进程可以进入临界区。在该进程未执行 **V** 原语操作之前如有另一进程想进入临界区的话，它也应先执行 **P** 原语操作，从而使 **sem** 的值变为 **-1**，因此，第二个进程将会被阻塞，直到第一个进程执行 **V** 原语操作之后， **sem** 的值变为 **0**，从而可唤醒第二个进程进入就绪队列，经调度后进入临界区。在第二个进程执行完 **V** 原语操作之后，如果没有其它进程申请进入临界区的话，则 **sem** 又恢复到初始值。

用信号量实现两并发进程 **Pa**，**Pb** 互斥的描述如下：

（**1**） 设 **sem** 为互斥信号量，其取值范围为（ **1**，**0**，**-1**）。

其中 **sem=1** 标志进程 **Pa**，**Pb** 都未进入类名为 **S** 的临界区， **sem=0** 表示进程 **Pa**，**Pb** 已进入类名为 **S** 的临界区， **sem=-1** 表示进程 **Pa**，**Pb** 中，一个进程已进入临界区，而另一进程等待进入临界区。

（**2**） 描述

Pa：

P（**sem**）

<S>

V(sem)：

…… .

Pb：

P（**sem**）

<S>

V(sem)：：

```
main()
  {
    int print=1;
    cobegin
      pa();
      pb();
    coend
```

两个进程并发执行时，print值1、0、-1，

若print=1，表示没有进程使用打印机;

若print=0，表示有一个进程正在使用打印机;

若print=-1，表示有一进程正在使用打印机，

还有一个进程等待使用打印机。

```
pa()
  {
    ....;
    p(print);
    使用打印机;
    v(print);
    ...;
    }
```

```
pb()
  {
    ....;
    p(print);
    使用打印机;
    v(print);
    ...;
    }
```

.....

6.进程同步

★同步：把异步环境下的一组并发进程，因直接制约而互相发送消息而进行互相合作、互相等待，使得各进程按一定的速度执行的过程称为进程间的同步。

用 **wait**（消息名）表示进程等待合作进程发来的消息 **.**

功能：等待到消息名为 **true** 的进程继续执行。

用 **signal**（消息名）表示向合作进程发送消息

功能：发送消息名，并将其值置为 **true**。

利用过程 **wait** 和 **singnal** 描述计算进程 **Pc** 和打印进程 **Pp** 的同步关系

（**1**） 设消息名 **Bufempty** 表示 **buf** 为空，消息名 **Buffull** 表示 **Buf** 中装满了数据。

（**2**） 初始化 **Bufempty=true** ， **Buffull=false.** 。

（**3**） 描述：

Pc ：

A：**wait(Bufempty)**

计算

Buf ← 计算结果

Bufempty ← **false**

signal(Buffull)

Goto A

Pp ：

B：**wait(Bufful)**

打印 **Buf** 中的数据

清除 **Buf** 中的数据

Buffuf ← **false**

signal(Bufempty)

Goto B

★私有信号量（ **private Semaphore**）：进程同步的信号量只与制约进程及被制约进程有关而不是与整组并发进程有关。因此该信号量称为私有信号量。

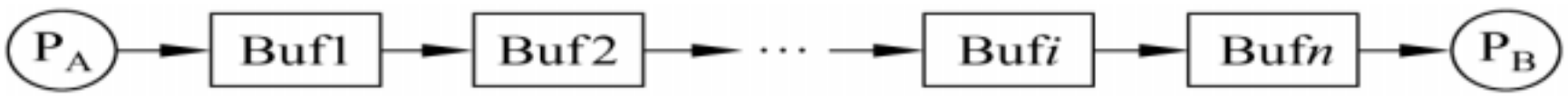
★用 **P,V** 原语操作实现同步

首先，为各并发进程设置私有信号量，

然后，为私有信号量赋初值，

最后，利用 **P**， **V** 原语和私有信号量规定各进程的执行顺序。

例：设进程 **Pa** 和 **Pb** 通过缓冲区队列传递数据。 **Pa** 为发送进程， **Pb** 为接收进程。 **Pa** 发送数据时调用发送过程 **deposit(data)**， **Pb** 接受数据时调用过程 **remove(data)**，且数据的发送和接受过程满足如下条件：



（**1**）在

```
P_A:  deposit(data):
      begin local x
          P(Bufempty);
          按 FIFO 方式选择一个空缓冲区 Buf(x);
          Buf(x)← dat
          Buf(x)置满标记
          V(Buffull)
      end
P_B:  remove(data):
      Begin local x
          P (Buffull);
          按 FIFO 方式选择一个装满数据的缓冲区 Buf(x)
          data ← Buf(x)
          Buf(x)置空标记
          V (Bufempty)
      End
```

★7.生产者与消费者问题

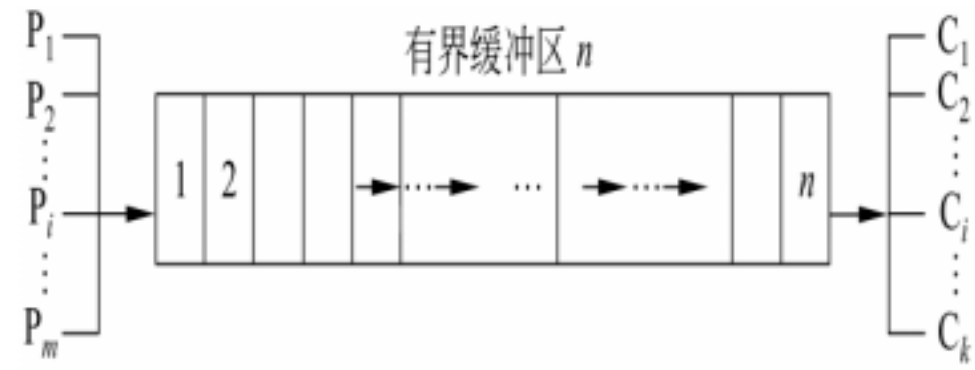
对于生产者进程：产生一个数据，当要送入缓冲区时，要检查缓冲区是否已满，若未满，则可将数据送入缓冲区，并通知消费者进程；否则，等待；

对于消费者进程：当它去取数据时，要看缓冲区中是否有数据可取，若有则取走一个数据，并通知生产者进程，否则，等待。

这种相互等待，并互通信息就是典型的进程同步。

同时，缓冲区是个临界资源，因此，诸进程对缓冲区的操作程序是一个共享临界区，因此，还有个互斥的问题。

```
deposit(data):
      begin
          P(avail)
          P(mutex)
          送数据入缓冲区某单元
          V(full)
          V(mutex)
      end
remove(data):
      begin
          P(full)
          P(mutex)
          取缓冲区中某单元数据
          V(avail)
          V(mutex)
      End
```



full:缓冲区产品数目，初值为 0； empty:缓冲区可存放产品的空位，初值为 n;
mutex:对缓冲区互斥信号灯，初值为 1；

```
producer()
 {
   while(生产未完成)
    {
      …;
      生产一个产品;
      p(empty);
      p(mutex);
      将产品放入缓冲区;
      v(mutex);
      v(full);
      }
   }
```

```
consumer()
 {
   while(还要继续消费)
    {
      p(full);
      p(mutex);
      从缓冲区中取出一个产品;
      v(mutex);
      v(empty);
      消费一个产品;
        …;
      }
   }
```

8.进程通信

通信（**communication**）意味着进程间传递数据。 操作系统可以看作是各种进程组成的， 这些进程都具有各自独立的功能，且大多数都被外部需要而启动执行。

在单机系统中进程的通信有 **4** 种形式：

（**1**）主从式

（**2**）会话式

（**3**）消息或邮箱机制

（**4**）共享存储区方式

会话方式的特点：

(1)使用进程在使用服务进程所提供的服务之前，必须得到服务进程的许可。

(2)服务进程根据使用进程的要求提供服务，但对所提供服务的控制由服务进程自身完成。

（**3**）使用进程和服务进程在进行通信时有固定连接关系。

消息或邮箱机制的特点是：

（**1**）只要存在空缓冲区或邮箱，发送进程就可以发送消息。

（**2**）与会话系统不同，发送进程和接受进程之间无直接联接关系。

（**3**）发送进程和接受进程之间存在缓冲区或邮箱用来存放被传送消息。

邮箱通信就是由发送进程申请建立一与接受进程联接的邮箱。设置邮箱的最大好处是发送进程和接受进程之间没有时间上的限制。

共享存储区方式不要求数据移动，两个需要互相交换信息的进程通过共享数据区的操作达到互相通信的目的。

9. 死锁问题

死锁：指个并发进程彼此互相等待对方所拥有的资源，且这些并发进程在得到对方的资源之前不会释放自己所拥有的资源。从而造成大家都想得到资源而又得不到资源，个并发进程不能继续向前推进的状态。

★死锁的起因：根本原因在于系统提供的资源个数少于并发进程所要求的该类资源数。

★产生死锁有四个必要条件：

（**1**）互斥条件。 并发进程所要求和占有的资源是不能同时被两个以上进程使用或操作的， 进程对他所需要的资源进行排他性控制。

（**2**）不剥夺条件。进程所获得的资源在未使用完毕之前，不能被其它进程强行剥夺，而只能由获得该资源的进程自己释放。

（**3**）部分分配。进程每次申请它所需要的一部分资源，在等待新资源的同时，继续占用已分配的资源。

（**4**）环路等待条件。存在一种进程循环链，链中每一个进程已获得的资源同时被下一个进程所请求。

只要有一个条件不满足，死锁就可解除。

预防死锁

1. 破坏‘请求与保持条件 ”每个进程在运行之前，必须预先提出自己所要使用的全部资源，调度程序在该进程所需要的资源末得到满足之前，不让它们投入运行，并且当资源一旦分配给某个进程之后，那么在该进程的整个运行期间相应资源一直被它占有，这就破坏了产生死锁的部分分配条件。

2. 破坏环路条件 对系统提供的每一项资源，由系统设计者将它们按类型进行线性排队，并赋予不同的序号。

3. 资源受控动态分配 为了避免死锁发生， 操作系统必须根据预先掌握的关于资源用法的信息控制资源分配， 使得共同进展路径的下一步不致于进入危险区，即只要有产生死锁的可能性，就避免把一种资源分配给一个进程。

死锁的检测和恢复

1. 资源剥夺法

（**1**）还原算法。即恢复计算结果和状态。

（**2**）建立检查点主要是用来恢复分配前的状态。

2. 撤消进程法

按一定的顺序中止进程序列，直至已释放到有足够的资源来完成剩下的资源为止。

第四章

1.一个作业从提交给计算机系统到执行结束退出系统，一般都要经历提交、收容、执行和完成四个状态。

一个作业在其处于从输入设备进入外部存储设备的过程成为提交状态。处于提交状态的作业，因其信息尚未全部进入系统，所以不能被调用程序选取。

收容状态也称为后备状态，输入管理系统不断地将作业输入到外存中对应部分（或称输入井，即专门用来存放待处理作业信息的一组外存分区）　。若一个作业的全部信息已全部被输入进输入井，那么，在它还未被调度去执行之前，该作业处于收容状态。

作业调度程序从后备作业中选取若干作业到内存投入运行。它为被选中作业建立进程并分配必要的资源，这时，这些被选中的作业处于执行状态。

当作业运行完毕，但它所占用的资源尚未全部被系统收回时，该作业处于完成状态。

一般来说，处理机调度可分为　　**4** 级：作业调度、交换调度、进程调度、线程调度。

作业调度：又称宏观调度或高级调度，其主要任务是按一定的原则对外存输入井上的大量后备作业进行选择，给选出的作业分配内存、输入输出设备等必要的资源，并建立相应的根程序，以使该作业的进程获得竞争处理机的权利，另外，当该作业执行完毕时，还负责回收系统资源。

交换调度：又称中级调度，其主要任务是按照给定的原则和策略，将处于外存交换区中的就绪状态或就绪等待状态的进程调入内存，或把处于内存就绪状态或内存等待状态的进程交换到外存交换区。交换调度主要涉及内存的管理和扩充，一般将它归在存储管理之中。

进程调度：又称微观调度或低级调度，其主要任务是按照某种策略和方法选取一个处于就绪状态的进程占用处理机。

只有在多道批处理系统中才有作业调度，而在分时和实时系统中一般只有进程调度、交换调度和线程调度。

这是因为在分时和实时系统中，为了缩短响应时间或为了满足用户需求的截止时间，作业不是建立在外存中，而是直接建立在内存中。

2.作业调度

作业调度的功能：

（**1**）记录系统中各作业的状况，包括执行阶段的有关情况。通常，系统为每个作业建立一个作业控制表　　**JCB** 记录这些有关信息。

作业控制块　**JCB**：在作业调度的过程中记录作业各方面的信息。它随作业的创建而产生，随作业的撤消而被清除。

（**2**）从后备队列中选取一部分作业投入执行

（**3**）为被选中的作业做好执行前的准备工作。

（**4**）在作业执行结束时做好善后处理工作。

作业调度目标：

（**1**）　对所有作业应该是公平合理的。

（**2**）　应使设备有高的利用率。

（**3**）　　每天执行尽可能多的作业

（**4**）　有快的响应时间

对于批处理系统，作业的平均周转时间或平均带权周转时间，被作为衡量调度算法优劣的标准；对于分时系统和实时系统，外加平均响应时间作为衡量调度算法优劣的标准

★**(1)**周转时间：

作业 **i** 从提交时刻到完成时刻称为作业的周转时间。　　**Ti　=　Tei　-　Tsi**

Tei 为作业　**i** 的完成时间，　**Tsi** 为作业的提交时间

一个作业的周转时间说明了该作业在系统内停留的时间　　,包含两部分：一是等待时间；二为执行时间

Ti　=　Twi　+　Tri

Twi 主要是指作业　**i** 由后备状态到执行状态的等待时间，它不包括作业进入执行状态后的等待时间。

★一批作业的平均周转时间为：

$$T=\frac{1}{n}\sum_{i=1}^{n} Ti$$

★带权周转时间

Wi=Ti/Tri　　**Ti** 作业周转时间　　**Tri** 作业执行时间

★一批作业的平均带权周转时间为

$$W=\frac{1}{n}\sum_{i=1}^{n} Wi$$

3. 进程调度

进程调度的功能：

①用 PCB 块记录系统中所有进程的执行情况

②按照一定的调度算法，选择一个处于就绪状态的进程，给它分配处理机 (这是最重要的功能)

③实施进行进程上下文的切换

引起进程调度的原因：

（1） 正在执行的进程执行完毕。这时，如果不选择新的就绪进程执行，将浪费处理机资源。

（2） 执行中进程自己调用阻塞原语将自己阻塞起来进入睡眠等待状态。

（3） 执行中进程调用了 P 原语操作，从而因资源不足而被阻塞；或调用了 V 原语激活了等待资源的进程队列。

（4） 执行中进程提出了 I/O 请求后被阻塞。

（5） 在分时系统中时间片已经用完。

（6） 在执行完系统调用，在系统程序返回用户进程，可认为系统进程执行完毕，从而可调度选择一新的用户程序执行。

以上都是 CPU 执行不可剥夺方式下做引起的进程调度的原因，在 CPU 执行方式是可剥夺时，还有：

（7） 就绪队列中的某进程的优先级变得高于当前执行进程的优先级，从而也将发生进程调度。

可剥夺方式：即就绪队列中一旦有优先级高于当前进程优先级的进程存在时，便立即发生进程调度，转让处理机。

非剥夺方式（不可剥夺方式） ：即使在就绪队列存在有优先级高于当前执行进程时，当前进程仍将继续占有处理机，直到该进程因自己调度调用原语操作或、等待 I/O 进入阻塞状态或时间片用完时才重新发生调度让出处理机。

进程调度性能评价

（1）进程调度性能是衡量操作系统性能的一个重要指标

（2）在大多数情况下，利用测试或模拟系统响应时间的方法来评价进程调度的性能

★4.调度算法

①先来先服务 (FCFS) 算法

将用户作业和就绪进程按提交顺序或变成就绪状态的先后排成队列，并按照先来先服务的方式进行调度处理。

优点：在一般意义下是公平的，即每个作业或进程都按照它们在队列中等待时间长短来决定它们是否优先享受服务。

缺点：对于那些执行时间较短的作业或进程来说，如果它们在某些执行时间很长的作业或进程之后到达，则它们等待很长时间。

②(时间片)轮转法 (RR)

算法描述：就绪队列按进程到达的时间来排列。处理机的时间被分为固定大小的时间片。调度程序总是选择就绪队列中的第一个进程。一个执行进程如果在用完一个时间片后还没有完成其任务，它就自动释放处理机回到就绪队列的末尾重新排队，等待下一次被调度。

缺点：只能用来分配那些可抢占资源，而且这种算法只能用于进程调度，不能用于作业调度（作业调度包含了不可抢占资源）。

时间片的选取非常重要，时间片长度的选择会直接影响系统开销和响应时间。如果时间片长度过短，则调度程序剥夺处理机的次数增多，这将使进程上下文交换次数也大大增加，加重了系统开销。如果时间片长度选择过长 (大)，大到一个进程足以完成其全部运行工作所需的时间，那么时间片轮转法就退化为先来先服务策略了。最佳的时间片量值应能使分时用户得到好的响应时间。

时间片的确定

在轮转法中，时间片长度 q 根据系统对响应时间的要求 R 和就绪队列中所能容纳的最大进程数 Nmax 确定的。

q=R/Nmax

一种改进的方法就是每当一轮调度开始时，系统根据就绪队列中当前的进程数计算一次 q，作为新一轮调度的时间片。

③多级反馈轮转法 (进程调度)

（1）在时间片轮转法中设置三个就绪队列

a.时间片完成就绪队列

b.等待结束就绪队列

c.新进程就绪队列

（2）每个队列建立时按 FCFS 排列，同一队列中进程的优先级相同，不同队列具有不同的优先级

优先级高的队列中进程的时间片短，优先级低的队列中进程的时间片长。

（3）进程调度时， 先调度高优先级就绪队列中的进程， 当高优先级就绪队列为空时才调度优先级低的就绪队列中的进程

（4）一个进程在执行过程中要经历不同的就绪队列

④优先级法

算法描述： 按照某种原则给作业或进程确定一个优先级，　　进程的就绪队列或作业的后备队列按对象的优先级进行排列，高前低后。对象进入队列是插入。当调度发生时，排列在最前面的进程或作业被调度。

确定优先级的方法有两类：动态法和静态法

静态法是根据作业或进程的静态特性，　　在作业或进程开始执行之前就确定它们的优先级，　　一旦开始执行后就不能改变。

动态法：把作业或进程静态性和动态性结合起来确定作业或进程的优先级，随着作业或进程的执行过程，优先级不断变化。

作业调度中静态优先级确定原则：

（**1**）　由用户自己根据作业的紧急程度输入一个适当的优先级

（**2**）　由系统或操作员根据作业类型指定优先级。

（**3**）　系统根据作业要求资源情况确定优先级。

进程调度静态优先级确定原则：

（**1**）　按照进程的类型给与不同的优先级。

（**2**）　将作业的静态优先级作为它所属进程的优先级。

由于在进程调度中静态优先级确定方法的缺陷：系统效率低、调度性能不高，所以多采用动态的方法确定优先级。

进程调度动态优先级确定原则：

（**1**）　根据进程占有　**CPU** 时间的长短来决定。一个进程占有处理机时间越长，则在被阻塞后再次获得调度的优先级越低，反之，获得调度的可能性越大

（**2**）　根据就绪进程等待　**CPU** 的时间长短来决定。一个就绪进程在就绪队列中等待的时间越长，则它获得调度选中的优先级就越高。

⑤最短作业优先法　**SJF(**作业调度　**)**

选择那些估计需要执行时间最短的作业投入执行，为它们创建进程和分配资源。

优点：可使得系统在同一时间内处理的作业个数最多，从而吞吐量也就大于其他调度方式。

缺点：对于一个不断有作业进入的批处理系统来说，最短作业优先法有可能使得那些长作业永远得不到调度执行的机会。

⑥最高响应比优先法　**(**作业调度　**)**

综合平衡　**FCFS**　和 **SJF**，既考虑等待时间长的作业，也照顾执行时间短的作业。

响应比：　**R=(** 等待时间　**W+** 执行时间　**T)/** 执行时间　**T**

优点：长作业有机会获得调度执行

缺点：同一时间内处理的作业数少于最短作业优先法，吞吐量也小于最短作业优先法

　　调度前计算响应比，系统开销增加。

算法评价

FCFS 算法

λ　：作业到达率；

μ　：服务器　**(**主机 **)**的服务率；

只有当　λ **<**　μ 时系统才是稳定的。

n：系统中的平均作业个数；

R：系统响应时间；

ρ　：　λ **/** μ，是系统中存在作业的概率，　　**1-** ρ 是系统中没有作业的概率。

　n=　ρ **/(1-** ρ **)**

Little　结果：　**n=** λ **R**；**R=n/** λ

FCFS　算法的评价：

　R=n/ λ **=** ρ **/(1-** ρ **)*1/** λ

RR　算法

q：时间片；

k：每个进程平均需要的时间片数，即该进程到达等待队列的次数；

线性优先级法的调度性能

1/ μ　：平均服务时间，则：　　**1/** μ **=k**× **q**

RR　算法的评价：

已使用过　**k** 次时间片的进程的响应时间是：

R(k)= ρ /(λ (1- ρ))

=1/(μ (1- ρ))=k × q/(1- ρ)

FCFS 方式短作业驻留时间与长作业相同，对短作业不利。

轮转法所需服务时间短的顾客响应时间将会小于所需服务时间长的顾客响应时间。

实时调度算法分类：静态表格驱动类、静态优先级驱动抢先式调度算法类、动态计划调度算法类、尽力而为调度算法类。

具有代表性的实时调度算法

时限式调度法（静态表格驱动类代表）　　：是一种以满足用户要求时限为调度原则的算法。

算法描述：时限有两种：处理开始时限和处理结束时限，在实际中可以使用任一种时限。

频率单调调度　（静态优先级驱动抢先式调度算法类代表）　　：是一种被广泛用于多周期性实时处理的调度算法。　　其基本原理是频率低（周期越长）的任务优先级越低。

第五章

1.存储器：能接收数据和保存数据、而且能根据命令提供这些数据的装置。

存储器分成两类：内存储器（简称内存、主存、物理存储器）外存储器（简称外存、辅助存储器）

虚拟存储器：为用户提供一种不受物理存储器结构和容量限制的存储器的技术称为虚拟存储器，或称虚拟存储技术。

虚拟存储器需要大容量的外存储器的支持，或称物资基础。

程序地址： 用户编程序时所用的地址　（或称逻辑地址　、虚地址　），基本单位可与内存的基本单位相同，　　也可以不相同。

程序地址空间（逻辑地址空间、虚地址空间）　　**:**用户的程序地址的集合称为逻辑地址空间，它的编址总是从　　**0** 开始的，可以是一维线性空间，也可以是多维空间。

物理地址：把内存分成若干个大小相等的存储单元，每个单元给一个编号，这个编号称为内存地址（物理地址、绝对地址、实地址），存储单元占　**8** 位，称作字节（　**byte**）。

物理地址空间：物理地址的集合称为物理地址空间（主存地址空间）　　，它是一个一维的线性空间。

安排进程的地址方法：

（**1**）　按照物理存储器中的位置赋予实际物理地址。好处：　　**CPU**　执行目标代码时的执行速度高。坏处：由于物理存储器的容量限制，能装入内存并发执行的进程数将会大大减少，对于某些较大的进程来说，当其所要求的总内存容量超过内存容量时将会无法执行；　　由于编译程序必须知道内存的当前空闲部分及其地址，　　并且把一个进程的不同程序段连续的存放起来，因此编译程序将非常复杂。

（**2**）　编译链接程序把用户源程序编译后链接到一个以　　**0** 地址为始地址的线性或多维虚拟地址空间。

2.存储管理功能：

★地址映射　　将程序地址空间中使用的逻辑地址变换成主存中的地址的过程

主存分配　　按照一定的算法把某一空闲的主存区分配给作业或进程。

存储保护　　保证用户程序　**(**或进程映象　**)**在各自的存储区域内操作　　**,**互不干扰。

提供虚拟存储技术　　使用户程序的大小和结构不受主存容量和结构的限制，　　即使在用户程序比实际主存容量还要大的情况下，程序也能正确运行.

★实现地址映射有三种方式：

①**.**编程或编译时确定地址映射关系

②**.**静态地址映射

③**.**动态地址映射

（**1**）编程或编译时确定地址映射关系

编程时确定虚－实地址的关系是指在用机器指令编程时，程序员直接按物理内存地址编程，这种程序在系统中是不能做任何移动的，否则就会出错。

（**2**）静态地址映射

静态地址映射是在程序装入内存时完成从逻辑地址到物理地址的转换的。在一些早期的系统中都有一个装入程序（加载程序），它负责将用户程序装入系统，并将用户程序中使用的访问内存的逻辑地址转换成物理地址。

优点：实现简单，不要硬件的支持。

缺点：程序一旦装入内存，移动就比较困难。有时间上的浪费。在程序装入内存时要将所有访问内存的地址转换成物理地址。

必须占用连续的内存空间，很难做到程序和数据的共享。

（**3**）动态地址映射

动态地址映射是在程序执行时由系统硬件完成从逻辑地址到物理地址的转换的。动态地址映射是由硬件地执行时完成

的，程序中不执行的程序就不做地址映射的工作，　　这样节省了　CPU 的时间　。 重定位寄存器的内容由操作系统用特权指令来设置，比较灵活。实现动态地址映射必须有硬件的支持，并有一定的执行时间延迟。现代计算机系统中都采用动态地址映射技术。

优点：可以对内存进行非连续分配，动态重定位提供了实现虚拟存储器的基础，有利于程序段的共享。

动态地址映射技术能满足以下目标：

（1）具有给一个用户程序任意分配内存区的能力；

（2）可实现虚拟存储；

（3）具有重新分配的能力

（4）对于一个用户程序，可以分配到多个不同的存储区

3.内外存数据传输的控制

要实现内存扩充，在程序执行过程中，内存和外存之间必须经常地交换数据。内外存的数据流动控制方法有两种

一种是用户自己控制程序，例子：覆盖技术，一种早期的主存扩充技术，要求用户了解程序结构，指定各程序段调入内存的先后次序。

另一种是操作系统控制，　　A 交换方式：操作系统把等待状态的进程换出内存，而把等待事件已发生，处于就绪态的进程换入内存。　B 请求调入方式和预调入方式：请求调入方式：在程序执行时，如果所要访问的程序段或数据段不在内存中，则操作系统自动地从外存将有关程序段和数据段调入内存地一种操作系统控制方式。预调入方式：系统预测在不远的将来会访问到的哪些程序段和数据段，并在它们访问前调入。

4.内存的分配和回收

在多道程序设计的环境中，内存分配的功能包括：制定分配策略、构造分配用的数据结构、响应系统的内存分配的请求和回收系统释放的内存区。内存管理策略有　　5 种：

（1）分配结构　　登记内存使用情况，供分配程序使用的表格和链表。

（2）放置策略　　确定调入内存的程序和数据在内存中的位置。决定内存中放置信息的区域（或位置）　　，即如何在若干个空闲区中选择一个或几个空闲区的原则；

(3)交换策略　　当内存不足时，决定将某些信息调出内存的策略　　。

(4)调入策略　　外存中的程序段和数据段什么时间按照什么样的控制方式进入内存

(5)回收策略　　回收的时机，对所回收的内存空闲区和已存在的内存空闲区的整理。

5.内存信息的共享与保护

常用的存储保护有三种。硬件法、软件法、软硬件结合

（1）上下界保护（常用的硬件保护法）

上界寄存器　　存放程序装入内存后的开始地址（首址）

下界寄存器　　存放程序装入内存后的末地址

判别式：上界寄存器　　≤　物理地址≤下界寄存器

（2）保护键法：为每一个被保护存储块分配一个单独的保护键。在程序状态字中则设置相应的保护键开关字段。

（3）界限寄存器与　CPU 的用户态或核心态工作方式相结合的保护方式。用户态进程只能访问那些在界限寄存器所规定范围内的内存部分，而核心态进程则可以访问整个内存地址空间。

6.分区存储管理

分区管理：把内存划分成若干个大小不等的区域，除操作系统占用一个区域之外，其余由多道环境下的各并发进程共享。

分区管理基本原理：给每一个内存中的进程划分一块适当大小的存储区，以连续存储各进程的数据和程序，使各进程得以并发执行。

按分区的时机，分区管理可以分为固定分区、动态分区。

（1）固定分区

把内存空间分成若干个大小不等的区域，称为分区。每个用户程序（作业、进程）调入内存后，占用其中一个分区，程序运行完成后释放该分区。

（2）动态分区

系统生成后，操作系统占用内存的一部分，剩下的部分作为一个空闲区，当一个用户程序（作业、进程）调入内存时，把这个空闲区的低地址部分的区域分配给它，　　当有作业完成后释放所占用的存储区。　　在系统运行的过程中，　系统中形成多个空闲的不连续的存储区，称主空闲。

分区的分配与回收

（1）固定分区时的分配和回收

当用户程序要装入执行时，通过请求表提出内存分配要求和所要求的内存空间大小。存储管理程序根据请求表查询分区说明表，从中找出一个满足要求的空闲分区，并将其分配给申请者。当进程执行完毕，不再需要内存资源时，管理程序将对应的分区状态置为未使用即可。

（**2**）动态分区时的分配和回收

动态分区时的分配与回收主要解决三个问题：分配空闲区、更新可用表、合并空闲区

动态分区时的分配方法从可用表或自由链中寻找空闲区的方法：首次适应算法、最佳适应算法、最坏适应算法

①首次适应算法

首次适应算法的表是按空闲区首址升序的（即空闲区表是按空闲区首址从小到大）方法组织的。

分配时从表首开始，以请求内存区的大小逐个与空闲区进行比较，找到第一个满足要求的空闲后，若空闲区大小与请求区的大小相等，则将该空闲区分配给请求者，并撤消该空闲区所在表目；若大于请求区，就将该空闲区的一部分分配给请求者，然后，修改空闲区的大小和首址。

②最佳适应算法

最佳适应算法是将申请者放入与其大小最接近的空闲区中。切割后的空闲区最小，若系统中有与申请区大小相等的空闲区，这种算法肯定能将这种空闲区分配给申请者。　　（首次适应法则不一定）　　这种算法最大的缺点是分割后的空闲区将会很小，直至无法使用，而造成浪费。

③最坏适应算法

为了克服最佳适应算法把空闲区切割得大小的缺点，人们提出了一种最坏适应算法，即每次分配时，总是将最大的空闲区切去一部分分配给请求者，其依据是当一个很大的空闲区被切割了一部分后可能仍是一个较大的空闲区。避免了空闲区越分越小的问题。

（**3**）动态分区的分配与回收

分配算法中切割空闲区是从低地址开始的，剩下的部分仍作为一个空闲区，门限值是切割空闲区后剩下的区域若小于门限值，就不切割该空闲区，统统分给申请者。

这三种放置算法的优劣很难区分，要具体情况具体分析。

例如：某时刻系统中有三个空闲区，其大小和首址为：　　**(35KB，100KB)**、**(12KB，156KB)**、**(28KB，200KB)**。有一作业系列：**(JOB1，12KB)**、**(JOB2，30KB)**、**(JOB3，28KB)**

（单位：KB）

大小	首址
35	100
12	156
28	200
0	

首次适应算法

（单位：KB）

大小	首址
23	**112**
12	156
28	200
0	

job1

不能继续进行分配

（单位：KB）

大小	首址
12	156
28	200
35	100
0	

最佳适应算法

（单位：KB）

大小	首址
28	200
35	100
0	

job 1

（单位：KB）

大小	首址
28	200
5	**130**
0	

job 2

（单位：KB）

大小	首址
5	**130**
0	

job3

（单位：KB）

大小	首址
35	100
28	200
12	156
0	

最坏适应算法

（单位：KB）

大小	首址
23	**112**
28	200
12	156
0	

job 1

不能继续进行分配

从搜索速度上看，最先适应算法具有最佳性能。从回收过程来看，最先适应算法也是最佳的。最先适应法尽可能地利用了地地址空间，从而保证高地址有较大的空闲区来放置要求内存较多的作业或进程。

最佳适应法找到的空闲区是最佳的，最坏适应法是基于不留下碎片空闲区这一点出发的，它选择最大的空闲区来满足用户的需求，以期分配后的剩余部分仍能进行再分配。

分区存储管理的优缺点：

优点：

（**1**）　实现了多个作业或进程对内存的共享，有助于多道程序设计，从而提高了系统的资源利用率

（**2**）　该方法要求的硬件支持少，管理算法简单，因而容易实现

缺点：

（1） 内存利用率仍然不高

（2） 作业或进程的大小受分区大小控制，除非配合采用覆盖技术和交换技术

（3） 无法实现各分区之间的信息共享

覆盖与交换技术

7.覆盖与交换技术是在多道环境下用来扩充内存的两种方法。

覆盖技术要求程序员提供一个清楚地覆盖结构。即程序员必须完成把一个程序划分成不同的程序段，并规定好它们的执行和覆盖顺序的工作。操作系统根据程序员提供的覆盖结构来完成程序段之间的覆盖。

交换技术是指先将内存某部分的程序或数据写入外存交换区，再从外存交换区中调入指定的程序或数据到内存中来，并让其执行的一种内存扩充技术。

交换技术不要求程序员给出程序段之间的覆盖结构，交换主要是在进程或作业之间进行，覆盖则主要是在同一个作业或进程内执行，覆盖只能覆盖那些与覆盖程序段无关的程序段。

交换进程由换入和换出两个过程组成。

8.页式管理

页式管理的基本原理

首先，进程虚拟地址空间分成大小相等的页面，进程的虚拟地址变为页号 **P** 与页内地址 **W** 组成。内存空间也按页的大小划分称片或页面，这些页面为系统中的任一进程所共享（除去操作系统以外） ，分页管理时，用户进程在内存空间内除了在每个页面内地址连续之外， 每个页面之间不再连续） 。采用请求调页或预调页技术实现内外存存储器的统一管理。

页式虚拟地址变为内存页面物理地址：页式管理把页式虚拟地址与内存页面物理地址建立一一对应页表，并用相应的硬件地址变换机构，来解决离散地址变换问题。

页式存储管理要解决如下问题：

（1）页式存储管理系统的地址映射；

（2）调入策略；

（3）淘汰策略；

（4）放置策略。

静态页面管理

静态页面管理方法是在作业或进程开始执行之前，把该作业或进程的程序段或数据全部装入内存的各个页面中，并通过页表和硬件变换地址机构实现虚拟地址到内存物理地址的地址映射。

①内存页面分配和回收

静态页面管理的第一步是为要求内存的作业或进程分配足够的页面。系统依靠存储页面表、请求表以及页表来完成内存的分配工作。

页表是页式存储管理的数据结构，它包括用户程序空间的页面与内存块的对应关系、页面的存储保护和存取控制方面的信息。

最简单的页表是由页号和页面号组成，页表在内存中占有一块固定的存储区，大小由进程或作业的长度来决定。页式管理时每个进程至少拥有一个页表。

请求表用来确定作业或进程的虚拟空间的各页在内存中的实际对应位置。系统应该知道每个作业或进程的页表起始地址和长度，以进行内存分配和地址变换。请求表中还应该包括每个进程或作业所请求的页面数。

存储页表指出内存各页面是否已被分配出去，以及未分配页面的总数。通常有两种记录空闲存储块的方法：位图法和链表法。

位图法：在内存中划分一块固定区域，每个单元的每个比特代表一个页面，如果该页面已被分配，则对应比特位置 **1**，否则置 **0**。

链表法： 在空闲页面链中， 对首页面的第一个单元和第二个单元分别放入空闲页面总数与指向下一个空闲页面的指针。其他页面的第一个单元中则分别放入指向下一个页面的指针。链表法由于使用了空闲页面本身的单元存放指针，因此不占据额外的内存空间。

分配算法：请求表给出进程或作业要求的页面数，然后，由存储页面数表检查是否有足够的空闲页面，如果没有，则本次无法分配，如果有则首先分配设置页表，并填写请求表中的相应表项后，按一定的查找算法，搜索出所要求的空闲页面，并将对应的页面号填入页表中。

静态页式管理的页面回收方法：当进程执行完毕时拆除对应的页表，并把页表中的各页面插入存储页面表即可。

动态页式管理

动态页式管理分为请求页式管理和预调入页式管理。

请求式分页存储管理与静态页式管理在内存块的分配与回收，存储保护某方面都十分相似，不同之处在于地址重定位问题。在请求式分页存储管理的地址重定位时，可能会出现所需页面不在主存的情况，此时，系统必须解决以下两个问题：

（1）当程序要访问的某页不在内存时，如何发现这种缺页情况？发现后应如何处理？

（2）当需要把外存上的某个页面调入内存时，此时内存中没有空闲块应怎么办？

怎样发现不在内存中虚页的问题可以用扩充页表的方法解决。增设缺页中断位和该页在外存的首址。缺页中断位：该位为“1”，表示此页已在内存；为“0”，表示该页不在内存。当此位为 0 时，会发出“缺页”中断信号，以求得系统的处理。

抖动现象：置换算法选择不当，有可能产生刚被调出内存的页又马上被调回内存，调回内存不久又马上被调出内存，如此反复的局面。这使得整个系统的页面调度非常频繁，以致大部分时间花费在主存和辅存之间的来回调入调出上的现象。

改变位：该位为“0”时，表示此页面在内存时数据未被修改过；为“1”时，表示被修改过。当此页面被选中为淘汰对象时，根据此位的取值来确定是否要将该页的内容进行磁盘回写操作。

页号	页面号	中断位	外存首址	改变位

请求页式管理中的置换算法

置换算法在内存中没有空闲页面时被调用。它的目的是选出一个被淘汰的页面。

把内存和外存统一管理的真正目的是把那些被访问概率非常高的页存放在内存中。因此，置换算法应该置换那些被访问概率最低的页，将它们移出内存。

比较常用的置换算法有：随机淘汰算法（在系统设计人员无法确定哪些页被访问的概率较低时，随机地选择某个用户的页面并将其换出）、轮转法 RR（轮转法循回换出内存可用去内一个可以被换出的页，无论该页是刚被换进或已换进内存很长时间）和先进先出法 FIFO（选择内存驻留时间最长的一页将其淘汰）。最近最久未用页面淘汰算法（最近最久未用（LRU）页面淘汰算法的着眼点是在要进行页面淘汰时，检查这些淘汰对象的被访问时间，总是把最长时间未被访问过的页面淘汰出去。这是一种基于程序局部性原理的淘汰算法。也就是说，该算法认为如果一个页面刚被访问过，那么不久的将来被访问的可能性就大；否则被访问的可能性就小。）最近最少用页面淘汰算法（最近最少用（LFU）页面淘汰算法的着眼点是考虑内存块中页面的使用频率，它认为在一段时间里使用得最多的页面，将来用到的可能性就大。因此，当要进行页面淘汰时，总是把当前使用得最少的页面淘汰出去。要实现 LFU 页面淘汰算法，应该为每个内存中的页面设置一个计数器。对某个页面访问一次，它的计数器就加 1。经过一个时间间隔，把所有计数器都清 0。产生缺页中断时，比较每个页面计数器的值，把计数器取值最小的那个页面淘汰出去。）最优页面淘汰算法（如果已知一个作业的页面走向，那么要进行页面淘汰时，应该把以后不再使用的或在最长时间内不会用到的页面淘汰出去，这样所引起的缺页中断次数肯定最小，这就是所谓的“最优（OPT）页面淘汰算法”。遗憾的是，OPT 的前提是要已知作业运行时的页面走向，这是根本不可能做到的，所以 OPT 页面淘汰算法没有实用价值，它只能用来做为一个标杆（或尺度），与别的淘汰算法进行比较。如果在相同页面走向的前提下，某个淘汰算法产生的缺页中断次数是否接近它。）

Belady 现象：一般来说，对于任一作业或进程，如果给它的页面数越接近于它所要求的页面数，则发生缺页的次数会越小。但是，使用 FIFO 算法时，有时会出现分配的页面数增多，缺页次数反而增加的奇怪现象。这种现象称为 Belady 现象。

存储保护

页式管理可以为内存提供两种方式的保护。一种是地址越界保护，另一种是通过页表控制对内存信息的存取操作方式以提供保护。

地址越界保护可由地址变换机构中的控制寄存器的值——页表长度和所要访问的虚地址相比较来完成。

存取控制保护的实现则是在页表中增加相应的保护位即可。

★页式管理的优缺点

优点

（1）由于它不要求作业或进程的程序段和数据在内存中连续存放，从而有效地解决了碎片问题；

（2）动态页式管理提供了内存和外存统一管理的虚存实现方式，使用户可以利用的存储空间大大增加。这既提高了主存的利用率，又有利于组织多道程序执行。

缺点

（1）要求有相应的硬件支持。例如地址变换机构，缺页中断的产生和选择淘汰页面等都要求有相应的硬件支持。这增加了机器成本。

（2）增加了系统开销，例如缺页中断处理。

（3）请求调页的算法如选择不当，有可能产生抖动现象。

（4）虽然消除了碎片，但每个作业和进程的最后一页总有一部分空间得不到利用。如果页面较大，则这一部分的损失仍然较大。

9.段式管理

段式存储管理的基本思想：把程序按内容或过程（函数）关系分成段，每段有自己的名字。一个用户作业或进程所包含的段对应于一个二维线性虚拟空间，也就是一个二维虚拟存储器。段式管理程序以段为单位分配内存，然后通过地址映射机构把段式虚拟存储地址转化为内存中的实际地址。和页式管理一样，段式管理也采用只把那些经常访问的段驻留内存，而把那些在将来一段时间内不被访问的段放在外存，待需要时自动调入内存的方法实现二维虚拟存储器。

段式与页式的比较

段　式	页　式
分段由用户设计自己划分，每段对应的程序模块，有完整的逻辑意义 段面是信息的逻辑单位 便于段的共享，执行时按需动态链接装入 段长不等，可动态装入，有利于新数据的增长 二维地址空间：段名、段中地址；段号、段内单元号 管理形式上象页式，但概念不同	分页用户看不见，由操作系统为内存管理划分 页面是信息的物理单位 页一般不能共享 页面大小相同，位置不能动态增加 一维地址空间 往往需要多次缺页中断才能把所需的信息完整地调入内存

段式管理的实现原理

段式管理把一个进程的虚地址空间设计成二维结构，即段号 **S** 与段内相对地址 **W**。段号与段号之间无顺序关系，段的长度是不固定的。每个段定义一组逻辑上完整的程序或数据。例如，一个进程中的程序和数据可被划分为主程序段、子程序段、数据段与工作区段。每个段是一个首地址为零、连续的一维线性空间。

段式管理的内存分配与释放

段式管理中以段为单位分配内存，每段分配一个连续的内存区。由于各段长度不等，所以这些存储区的大小不一。而且，同一进程所包含的各段之间不要求连续。段式管理的内存分配和释放是动态进行的，与分区式管理一样可以采用最先适应法、最佳适应法、最坏适应法等进行空闲区分配。内存回收法也同分区式管理。当内存中没有足够的空闲区时，需要淘汰算法。

★段式管理的地址变换

由于段式管理只存放部分信息副本在内存，而大部分信息在外存中，这必然引起 **CPU** 访问时发生所要访问的段不在内存现象。 那么 **CPU** 如何感知到所要访问的段不在内存而启动中断处理程序呢？还有，段式虚拟地址属于一个二维的虚拟空间，怎样变换到一个一维线性物理地址呢？这些都由段式地址变换机构解决。

段式管理程序在进行初始内存分配之前，首先根据用户要求的内存大小为一个作业或进程建立一个段表，以实现动态地址变换和缺段中断处理及存储保护等。

段式管理的地址变换：一般在内存中给出一块固定的区域放置段表。当某进程开始执行时，管理程序首先把该进程的段表始地址放入段表地址寄存器中。通过访问段表寄存器，管理程序得到该进程的段表始地址从而可开始访问段表。

然后，由虚拟地址中的段号 **s** 为索引，查段表。若该段在内存，则判断其存取控制方式是否有错。如果存取控制方式正确，则从段表相应表目中查出该段在内存的起始地址，并将其和段内相对应地址 **w** 相加，从而得到实际内存地址。

若该段不存在，则产生缺段中断将 **CPU** 控制权交给内存分配程序。内存分配程序首先检查空闲区链，以找到足够长度的空闲区来装入所需的段。如果内存中的可用空闲区总数小于所要求的段长时，则检查段表中访问位，以淘汰那些访问概率低的段并将需要段调入。

段的共享与保护

段式存储管理可以方便地实现内存信息共享和进行有效地内存保护。这是因为段是按逻辑意义来划分的，可以按名访问的缘故。

段的共享：在多道环境下，常常有许多子程序和应用程序是被多个用户所使用的。特别是在多窗口系统、支持工具等广泛流行的今天，被共享的程序和数据的个数和体积都在急剧增加，有时往往超过用户程序长度的许多倍。

内存只保留一个副本，供多个用户使用，称为共享。
在多道环境下，由于进程的并发执行，一段程序为多个进程共享时，有可能出现多次同时重复执行该段程序的情况。
这就要求它在执行过程中，该段程序的指令和数据不能被修改。
共享段进行内外存交换时，应该设置一个共享位。显然，一个正在被某进程使用或即将被某进程使用的共享段是不应该调出内存的。
段的保护：（**1**）地址越界保护法（ **2**）存取方式控制保护法
★段式管理的优缺点
优点
提供了内外存统一管理的虚存实现。
段长可根据需要动态增长。
便于对具有完整逻辑功能的信息段进行共享。
便于实现动态链接。
缺点
需要更多的硬件支持。
处理碎片比较麻烦。
给系统管理带来一定的难度和开销。
每个段的长度受内存可用区大小的限制。
选择不恰当的淘汰算法，可能会产生抖动现象。
10.段页式管理
段页式管理的基本思想
段式管理为用户提供了一个二维的虚地址空间，反映了程序的逻辑结构，有利于段的动态增长以及共享和内存保护等，这大大方便了用户。而分页管理系统则有效地克服了碎片，提高了存储器的利用率。从存储管理的目的来讲，主要是方便用户的程序设计和提高内存的利用率。那么把段式管理和页式管理结合起来让其互取长补短不是更好吗？于是，段页式管理方式便被提了出来。一般仅用于大型机。
段页式管理的实现原理
段页式管理时的进程的虚拟地址空间中的虚拟地址由三部分组成：即段号 **S**，页号 **P** 和页内相对地址 **D**。由于虚拟空间的最小单位是页而不是段，从而内存可用区也就被划分成为若干个大小相等的页面，且每段所拥有的程序和数据在内存中可以分开存放。分段的大小也不再受内存可用区的限制。
为了实现段页式管理，系统必须为每个作业或进程建立一张段表，管理内存分配与释放、缺段处理、存储保护和地址变换等。另外，由于一个段又被划分成了若干页，每个又必须建立一张页表，把段中的虚页变换成内存中实际页面。显然，与页式管理时相同，页表中也要有实现缺页中断处理和页面保护等功能的表项。
另外，由于在段页式管理中，页表不再属于进程而属于段，因此，段表中应有页表首址和长度的项。
★动态地址变换过程
在一般使用段页式存储管理的计算机系统中，都在内存中开辟出一块固定的区域存放进程的段表和页表。因此，在段页式管理系统中，要对内存中指令或数据进行一次存取的话，至少需要访问三次以上的内存。显然，**CPU** 的执行指令速度大大降低。
为了提高地址转换速度，设置快速联想寄存器。它用于存放当前最常用的段号、页号和对应的内存页面与其它控制用栏目。
★局部性原理和抖动问题
程序设计常识告诉我们，一个作业往往含有许多循环和子程序的结构。因此，在作业运行期间，在一小段时间内，访问的地址空间往往只涉及整个程序的一小部分。在另一小段数据内，又只涉及另外的一小部分。这种现象称为局部性特征。
反映在页面综迹里，这种特征表现为，在任何一小段时间里，作业只集中于访问某几页。
所谓工作集，就是一个作业在某一小段时间内访问页面的集合。
如用 **W(t,△t)**表示在 **(t-△t)**到 **t** 之间所访问的不同的页面，那么，这个 **W** 就称之为作业在时间 **t** 的工作集。工作集长度是 **W (t,△t)**中的页面数。工作集长度越短，局部性越突出。
一般来说，一个作业的工作集，在运行的不同时刻是不同的，工作集大小亦不相等。而且，工作集大小与△ **t** 有关。△ **t** 大小很难确定，过小，就不能体现一个工作集过渡到另一个工作集一般是缓慢的这一局部性特征。
一个进程执行过程中缺页的发生有两种可能。一种是并发进程所要求的工作集总和大于内存可提供的可用区。这时，系统将无法正常工作，因为缺乏足够的空间装入需要的程序和数据。

另一种可能性是，虽然存储管理程序为每个并发进程分配了足够的工作集，但系统无法在开始执行前选择适当的程序和数据进入内存。这种情况下，只能依靠执行过程中，当 **CPU** 发现所要访问的指令或数据不在内存时，由硬件中断后转入中断处理程序，将需要的程序和数据调入。

系统抖动：当给进程分配的内存小于所要求的工作集时，由于内存外存之间交换频繁，访问外存时间和输入输出处理时间大大增加，反而造成 **CPU** 因等待数据空转，使得整个系统性能大大下降，这就造成了系统抖动。

解决抖动问题

1、增加工作集大小；

2、选择不同的淘汰算法，尽量保持工作集页面在内存中。

实际上，为了使系统获得高效率，暂停一个作业，当其有足够数量的页面在主存时才恢复运行；而在调度一个新作业时，必须有足够多的空闲存储块，才让其进入主存。