第 3 章　操作系统知识

3.1　操作系统基础知识

计算机系统中的软件极为丰富，通常分为应用软件和系统软件两大类。应用软件是指计算机用户利用计算机的软件、硬件资源为某一专门的应用目的而开发的软件。例如，科学计算、工程设计、数据处理、事务处理和过程控制等方面的程序，以及文字处理软件、表格处理软件、辅助设计软件（CAD）和实时处理软件等。系统软件是计算机系统的一部分，由它支持应用软件的运行。常用的系统软件有操作系统、语言处理程序、链接程序、诊断程序和数据库管理系统等。操作系统是计算机系统中必不可少的核心系统软件，其他软件是建立在操作系统的基础上，并在操作系统的统一管理和支持下运行，是用户与计算机之间的接口。

3.1.1　操作系统的定义与作用

计算机系统的硬件资源包括中央处理机、存储器（包括主存与外存）和输入输出设备等物理设备；计算机系统的软件资源是以文件形式保存在存储器上的程序和数据等信息。操作系统能有效地组织和管理系统中的各种软、硬件资源，合理地组织计算机系统工作流程，控制程序的执行，并且向用户提供一个良好的工作环境和友好的接口。

操作系统有如下两个重要的作用。

（1）通过资源管理，提高计算机系统的效率。操作系统是计算机系统的资源管理者，它含有对系统软、硬件资源实施管理的一组程序。其首要作用就是通过 CPU 管理、存储管理、设备管理和文件管理，对各种资源进行合理地分配，改善资源的共享和利用程度，最大限度地发挥计算机系统的工作效率，提高计算机系统在单位时间内处理工作的能力（称为系统的“吞吐量（throughput）”。

（2）改善人机界面，向用户提供友好的工作环境。如果不安装操作系统，用户将要面对的是 01 代码和一些难懂的机器指令，通过按钮或按键来操作计算机，这样即笨拙又费时。一旦安装操作系统后，用户面对的不再是笨拙的裸机，而是操作便利、服务周到的操作系统，从而明显改善了用户界面，提高了用户的工作效率。

3.1.2　操作系统的特征与功能

操作系统的 4 个特征是并发行、共享性、虚拟性和不确定性。从资源管理的观点来看，操

作系统的功能可分为处理机管理、文件管理、存储管理、设备管理和作业管理5大部分。操作系统的 5 大部分通过相互配合、协调工作，以实现对计算机系统中资源的管理，控制任务的运行。

（1）处理机管理。实质上是对处理机执行“ 时间 ”进行管理，采用多道程序等技术将 CPU 的时间真正合理地分配给每个任务。主要包括进程控制、进程同步、进程通信和进程调度。

（2）文件管理。这又称为信息管理。主要包括文件存储空间管理、目录管理、文件的读写管理和存取控制。

（3）存储管理。这是对主存储器“ 空间 ”进行管理。主要包括存储分配与回收、存储保护、地址映射（变换）和主存扩充。

（4）设备管理。这实质是对硬件设备的管理，其中包括对输入输出设备的分配、启动、完成和回收。

（5）作业管理。这包括任务、人机交互和用户界面管理等。

操作系统提供系统命令一级的接口，供用户用于组织和控制自己的作业运行，如命令行、菜单式或 GUI“ 联机 ”、命令脚本“ 脱机 ”。操作系统还提供编程一级接口，供用户程序和系统程序调用操作系统功能，如系统调用和高级语言库函数。

3.1.3 操作系统的类型

操作系统分为批处理操作系统（简称批处理）、分时操作系统、实时操作系统、网络操作系统、分布式操作系统、微机操作系统和嵌入式操作系统。

1．批处理操作系统

批处理操作系统分为单道批处理和多道批处理两种。

（1）单道批处理操作系统。单道批处理操作系统是一种早期的操作系统，该系统可以提交多个作业，“ 单道 ”的含义是指一次只有一个作业装入内存执行。作业由用户程序、数据和作业说明书（作业控制语言）三部分组成。当一个作业运行结束后，随即自动调入同批的下一个作业运行，从而节省了作业之间的人工干预时间，提高了资源的利用率。

（2）多道批处理操作系统。多道批处理操作系统允许多个作业装入内存执行，在任意一个时刻，作业都处于开始点和终止点之间。每当运行中的一个作业因输入/输出操作需要调用外部设备时，就把 CPU 及时交给另一道等待运行的作业，从而将主机与外部设备的工作由串行改变为并行，进一步避免了因主机等待外设完成任务而白白浪费宝贵的 CPU 时间。多道批处理系统主要有三个特点：多道、宏观上并行运行、微观上串行运行。

2．分时系统

分时操作系统将 CPU 的工作时间划分为许多很短的时间片，轮流为各个终端的用户服务。

分时系统的特点如下。

（1）多路性。允许在一台主机上同时连接多台联机终端，系统按分时原则为每个用户服务。宏观上是多个用户同时工作，共享系统资源，而微观上则是每个用户作业轮流运行一个时间片。多路性即同时性，它提高了资源利用率，从而促进了计算机更广泛的应用。

（2）独立性。每个用户各占一个终端，彼此独立操作，互不干扰。因此，用户会感觉到就像他一人独占主机。

（3）交互性。用户可通过终端与系统进行人机对话。用户可以请求系统提供多方面服务，如文件编辑、数据处理和资源共享等。

（4）及时性。用户的请求能在很短时间内获得响应，此时间间隔是以人们所能接受的等待时间来确定的，通常为 1～2s。响应时间是分时系统的重要指标，它是用户发生终端命令到系统做出响应间的时间间隔。系统的响应时间主要是根据用户所能接受的等待时间确定的。分时系统中时间片的选择是一个复杂和关键的任务，如时间片选得过大，造成响应时间不变时用户数减少，或造成响应时间过长，当时间片过小时，在一个时间片内切换开销相对增加，一个进程相对要花费更多的时间片才能运行结束，一个进程在系统中的周转时间大大增长。最佳的时间片值应既能使分时用户得到好的响应时间，同时又要使在一个时间片内切换开销相对较小可忽略。

UNIX 系统是典型多用户、多任务的分时操作系统。

3．实时系统

实时是指计算机对于外来信息能够以足够快的速度进行处理，并在被控对象允许的时间范围内做出快速响应。实时系统对交互能力要求不高，但可靠性要求高。为了提高系统的响应时间，对随机发生的外部事件做出及时响应并对其进行处理。

实时系统通常分为如下两类。

（1）实时控制系统。主要用于生产过程的自动控制，实验数据自动采集，武器的控制，包括火炮自动控制、飞机自动驾驶、导弹的制导系统。

（2）实时信息处理系统。主要用于实时信息处理，如飞机订票系统、情报检索系统。

实时系统的主要特点如下。

（1）快速的响应时间。实时系统是为了提高系统响应时间而设计的操作系统，特别是实时控制系统，对外部事件的响应要十分及时迅速。外部事件往往以中断方式通知系统，系统有较强的中断处理能力，实时系统的设计也以“事件驱动”方式来设计。

（2）有限的交互能力。实时系统（如实时信息处息系统）一般是专用系统，它能提供人机交互方式，但用户只能访问系统中某些特定的专用服务程序，不能像分时系统一样向终端用户提供多方面服务。

（3）高可靠性。批处理系统和分时系统虽也要求系统可靠，相比之下，实时系统则要求系统高度可靠。因此，实时系统中往往都采用双机系统，多级容错措施来保证系统和数据的安全。

实时系统与分时系统的主要区别如下。

（1）系统的设计目标不同。分时系统是设计成一个多用户的通用系统，交互能力强；而实时系统大都是专用系统。

（2）交互性的强弱不同。分时系统是多用户的通用系统，交互能力强；实时系统是专用系统，仅允许操作并访问有限量的专用程序，不能随便修改其中的程序，且交互能力差。

（3）响应时间的敏感程度不同。分时系统是以人能接受的等待时间为系统的设计依据，而实时系统是以被测物体所能接受的延迟为系统设计依据，因此实时系统对响应时间的敏感程度强，而分时系统的敏感程度弱。

4．网络操作系统

网络操作系统是使联网计算机能方便而有效地共享网络资源，为网络用户提供所需各种服务的软件和有关协议的集合。因此，网络操作系统的功能主要包括高效、可靠的网络通信；对网络中共享资源（在 LAN 中有硬盘、打印机等）有效的管理；提供电子邮件、文件传输、共享硬盘和打印机等服务；网络安全管理；提供互操作能力。

5．分布式操作系统

分布式计算机系统是由多个分散的计算机经连接而成的计算机系统，系统中的计算机无主、次之分，任意两台计算机可以通过通信交换信息。为分布式计算机配置的操作系统称为“ 分布式操作系统 ”。

分布式操作系统能直接对系统中各类资源进行动态分配和调度、任务划分、信息传输协调工作，并为用户提供一个统一的界面，标准的接口，用户通过这一界面实现所需要的操作和使用系统资源，使系统中若干台计算机相互协作完成共同的任务，有效控制和协调诸任务的并行执行，并向系统提供统一、有效的接口的软件集合。

分布式操作系统是网络操作系统的更高级形式，它保持网络系统所拥有的全部功能，同时又有透明性、可靠性和高性能等。网络操作系统与分布式操作系统虽然都属于管理分布在不同地理位置的计算机，但最大的差别是：网络操作系统的工作，用户必须知道网址，而分布式系统用户则不必知道计算机的确切地址；分布式操作系统负责全系统的资源分配，分布式 OS 通常很好地隐藏系统内部的实现细节，如对象的物理位置、并发控制和系统故障等对用户都是透明的。

6．微机操作系统

微型计算机（简称微机）的出现犹如一颗重磅炸弹，导致了计算机产业革命，拥有巨大的

使用量和广泛的用户。将配置在微机上的操作系统称为微机操作系统。常用的微机操作系统有MS-DOS、MS Windows、OS/2、SCO UNIX 和 Linux 等。其中，单用户单任务操作系统 MS-DOS 是 Microsoft 公司开发的首先在 IBM-PC 上使用的微机 OS，现在成了事实上的 16 位微机单用户单任务操作系统的标准。

多任务操作系统 Windows 98/NT/2000/XP 是 Microsoft 公司开发的一个图形用户界面的多任务、多线程、全 32 位的操作系统。多用户多任务操作系统 SCO UNIX，是 SCO 公司将运行于大、中、小型机上 UNIX 操作系统移植到微机上的。

Linux 操作系统是一个遵循标准操作系统界面的免费操作系统，具有 UNIX BSD 和 UNIX SYS Ⅴ的扩展特性。它的版权所有者是芬兰籍的 Linus B.Toroalds 和其他开发人员，并且遵循通用公共许可证（General Public License，GPL），保证用户有使用上的自由、获得源程序的自由、可以自己修改的自由、可以复制和推广的自由。

7．嵌入式操作系统

嵌入式操作系统是运行在嵌入式智能芯片环境中，对整个智能芯片以及它所操作、控制的各种部件装置等资源进行统一协调、处理、指挥和控制的系统软件。

3.2 处理机管理

处理机管理也称进程管理。在多道程序批处理系统和分时系统中，有多个并发执行的程序，为了描述系统中程序动态变化的过程引入了进程。进程是资源分配和独立运行的基本单位。处理机管理重点需要研究诸进程之间的并发特性，以及进程之间相互合作与资源竞争产生的问题。

3.2.1 基本概念

1．程序与进程

1）程序顺序执行的特征

前趋图是一个有向无循环图，图由节点和节点间的有向边组成，节点代表各程序段的操作，而节点间的有向边表示两程序段操作之间存在的前趋关系（“→”）。两程序段 P_i 和 P_j 的前趋关系表示成 $P_i \rightarrow P_j$，其中 P_i 是 P_j 的前趋，P_j 是 P_i 的后继，其含义是 P_i 执行完毕才能由 P_j 执行。例如，图 3-1 为三个程序段，第一个程序段输入是第二个程序段计算的前驱，第二个程序段计算是第三个程序段输出的前驱。

图 3-1　三个节点的前驱图

程序顺序执行时的特征包括顺序性、封闭性和可再现性。

2）程序并发执行的特征

若在计算机系统中采用多道程序设计技术，则主存中的多道程序可处于并发执行状态。对于上述有三个程序段的作业类，虽然每个作业有前趋关系的各程序段不能在 CPU 和输入输出各部件并行执行，但是同一个作业内没有前趋关系的程序段或不同作业的程序段可以分别在 CPU 和各输入输出部件上并行执行。例如，某系统中有一个 CPU、一台输入设备和一台输出设备，每个作业具有三个程序段输入 I_i，计算 C_i 和输出 $P_i(i=1,2,3)$。图 3-2 为三个作业的各程序段并发执行的前驱图。

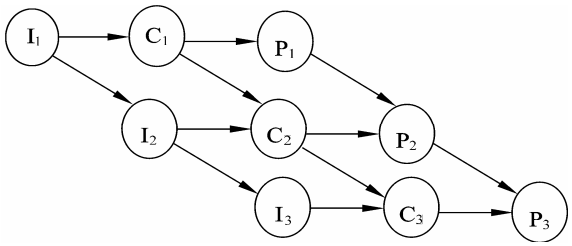


图 3-2　程序并发执行的前驱图

从图 3-2 中可以看出，I_2 与 C_1 并行执行；I_2、C_2 与 P_1 并行执行；C_3 与 P_2 并行执行。其中，I_2、I_3 受到 I_1 的间接制约，C_2、C_3 受到 C_1 的间接制约，P_2、P_3 受到 P_1 的间接制约；而 C_1、P_1 受到 I_1 的直接制约，等等。

程序并发执行时的特征如下。

（1）失去了程序的封闭性。

（2）程序和机器的执行程序的活动不再一一对应。

（3）并发程序间的相互制约性。

例如，两个并发执行的程序段完成交通流量的统计，其中“观察者”P_1 识别通过的车辆数，“报告者”P_2 定时将观察者的计数值清 0。程序实现如下。

P_1	P_2
L1: if 有车通过 then	L2: PRINT COUNT;
COUNT:=COUNT+1;	COUNT:=0;
GOTO L1;	GOTO L2;

对于上例，由于程序可并发执行，所以可能有以下三种执行序列。

① COUNT:=COUNT+1；PRINT COUNT；COUNT:=0

② PRINT COUNT；COUNT:=0；COUNT:=COUNT+1

③ PRINT COUNT；COUNT:=COUNT+1；COUNT:=0

假定 COUNT 的某个循环的初值为 *n*，那么对这三种执行序列得到的 COUNT 结果不同，如表 3-1 所示。

表 3-1　程序并发执行的结果

执 行 序 列	①	②	③
COUNT 打印的值	*n*+1	*n*	*n*
COUNT 执行后的值	0	1	0

这种不正确结果的发生是因为两个程序 P_1 和 P_2 共享变量 COUNT 引起的，即程序并发执行破坏了程序的封闭性和可再现性，使得程序和执行程序的活动不再一一对应。为了解决这一问题，需要研究进程间的同步与互斥问题。

2．进程的组成

进程是程序的一次执行，该程序可以和其他程序并发执行。进程通常是由程序、数据和进程控制块（PCB）组成的。

（1）进程控制块。进程控制块是进程存在的唯一标志，其主要内容如表 3-2 所示。

表 3-2　PCB 的内容

信　　息	含　　义
进程标识符	标明系统中的各个进程
状态	说明进程当前的状态
位置信息	指明程序及数据在主存或外存的物理位置
控制信息	参数、信号量、消息等
队列指针	链接同一状态的进程
优先级	进程调度的依据
现场保护区	将处理机的现场保护到该区域以便再次调度时能继续正确运行
其他	因不同的系统而异

（2）程序。程序部分描述了进程需要完成的功能。假如一个程序能被多个进程同时共享执行，那么这一部分就应该以可再入（纯）码的形式编制，它是程序执行时不可修改的部分。

（3）数据。数据部分包括程序执行时所需的数据及工作区。这部分只能为一个进程所专用，

是进程的可修改部分。

3．进程的状态及其状态间的切换

1）三态模型

在多道程序系统中，进程在处理器上交替运行，状态也不断地发生变化，因此进程一般有三种基本状态：运行、就绪和阻塞。图 3-3 显示了进程基本状态及其转换，也称三态模型。

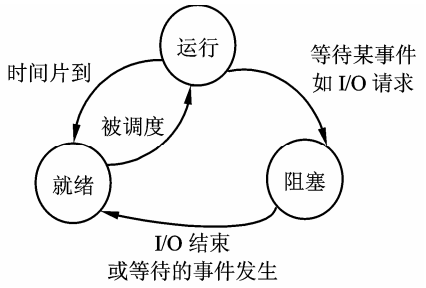


图 3-3　进程的三态模型

（1）运行。当一个进程在处理机上运行时，则称该进程处于运行状态。显然，对于单处理机系统，处于运行状态的进程只有一个。

（2）就绪。一个进程获得了除处理机外的一切所需资源，一旦得到处理机即可运行，则称此进程处于就绪状态。

（3）阻塞。阻塞也称等待或睡眠状态，一个进程正在等待某一事件发生（例如请求 I/O 而等待 I/O 完成等）而暂时停止运行，这时即使把处理机分配给进程也无法运行，故称该进程处于阻塞状态。

2）五态模型

事实上，对于一个实际的系统，进程的状态及其转换将更复杂。例如，引入新建态和终止态构成了进程的五态模型。如图 3-4 所示。

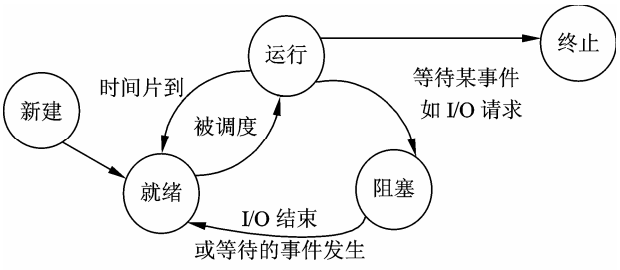


图 3-4　进程的五态模型

其中，新建态对应于进程刚刚被创建时没有被提交的状态，并等待系统完成创建进程的所有必要信息。因为创建进程时分为两个阶段，第一个阶段为一个新进程创建必要的管理信息，第二个阶段让该进程进入就绪状态。由于有了新建态操作系统，往往可以根据系统的性能和主存容量的限制推迟新建态进程的提交。

类似地，进程的终止也可分为两个阶段，第一个阶段等待操作系统进行善后处理，第二个阶段释放主存。

3）具有挂起状态的进程状态及其转换

由于进程的不断创建，系统资源特别是主存资源已不能满足进程运行的要求。这时，就必须将某些进程挂起，放到磁盘对换区，暂时不参加调度，以平衡系统负载。或者是系统出现故障，或者是用户调试程序，也可能需要将进程挂起检查问题。图 3-5 是具有挂起状态的进程状态及其转换。

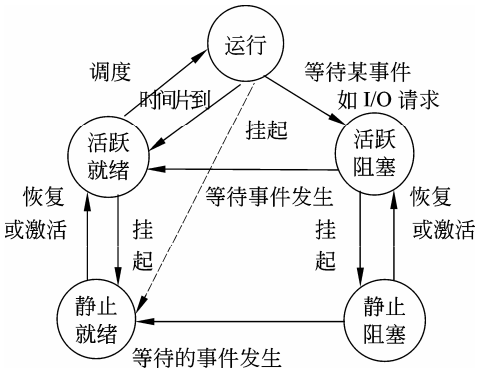


图 3-5　细分进程状态及其转换

（1）活跃就绪。活跃就绪是指进程在主存并且可被调度的状态。

（2）静止就绪。静止就绪是指就绪进程被对换到辅存时的状态，是不能被直接调度的状态，只有当主存中没有活跃就绪态进程，或者是挂起态进程具有更高的优先级，系统将把挂起就绪态进程调回主存并转换为活跃就绪。

（3）活跃阻塞。活跃阻塞是指进程在主存，一旦等待的事件产生便进入活跃就绪状态。

（4）静止阻塞。静止阻塞是指阻塞进程对换到辅存时的状态，一旦等待的事件产生便进入静止就绪状态。

3.2.2　进程的控制

进程控制就是对系统中所有进程从创建到消亡的全过程实施有效的控制。为此，操作系统设置了一套控制机构，该机构的主要功能包括创建一个新进程，撤销一个已经运行完的进程，改变进程的状态，实现进程间的通信。进程控制是由操作系统内核（Kernel）中的原语实现的。内核是计算机系统硬件的首次延伸，是基于硬件的第一层软件扩充，它为系统对进程进行控制和管理提供了良好的环境。

原语（Primitive）是指由若干条机器指令组成的，用于完成特定功能的程序段。原语的特点是在执行时不能被分割，即原子操作要么都做，要么都不做。内核中所包含的原语主要有进程控制原语、进程通信原语、资源管理原语以及其他方面的原语。属于进程控制方面的原语有进程创建原语、进程撤销原语、进程挂起原语、进程激活原语、进程阻塞原语以及进程唤醒原语等。不同的操作系统内核所包含的功能不同，但大多数操作系统的内核都包含支撑功能和资源管理的功能。

3.2.3 进程间的通信

在多道程序设计的系统中，由于多个进程可以并发执行，故进程间必然存在资源共享和相互合作的问题。进程通信是指各个进程交换信息的过程。

1．同步与互斥

同步是合作进程间的直接制约问题，互斥是申请临界资源进程间的间接制约问题。

1）进程间的同步

在计算机系统中，多个进程可以并发执行，每个进程都以各自独立的、不可预知的速度向前推进，但是需要在某些确定点上协调相互合作进程间的工作。例如，进程 A 向缓冲区送数据，进程 B 从缓冲区取数据加工，当进程 B 要取数据加工时，必须是进程 A 完成了向缓冲区送数据的操作，否则进程 B 必须停下来等待进程 A 的操作结束。可见，进程间的同步是指进程间完成一项任务时直接发生相互作用的关系。

2）进程间的互斥

在多道程序系统环境中，各进程可以共享各类资源，但有些资源一次只能供一个进程使用，称为临界资源（Critical resource，CR），如打印机、公享变量和表格等。进程间的互斥是指系统中各进程互斥使用临界资源。

3）临界区管理的原则

临界区（Critical section，CS）是进程中对临界资源实施操作的那段程序。对互斥临界区管理的 4 条原则如下。

（1）有空即进。当无进程处于临界区时，允许进程进入临界区，并且只能在临界区运行有限的时间。

（2）无空则等。当有一个进程在临界区时，其他欲进入临界区的进程必须等待，以保证进程互斥地访问临界资源。

（3）有限等待。对要求访问临界资源的进程，应保证进程能在有限时间进入临界区，以免陷入“饥饿”状态。

（4）让权等待。当进程不能进入自己的临界区时，应立即释放处理机，以免进程陷入忙等

状态。

2．信号量机制

荷兰学者 Dijkstra 与 1965 年提出的信号量机制是一种有效的进程同步与互斥工具。目前信号量机制有了很大的发展，主要有整型信号量、记录型信号量和信号量集机制。

1）整型信号量与 PV 操作

信号量是一个整型变量，根据控制对象的不同被赋予不同的值。信号量分为如下两类：

（1）公用信号量。实现进程间的互斥，初值为 1 或资源的数目。

（2）私用信号量。实现进程间的同步，初值为 0 或某个正整数。

信号量 S 的物理意义：S≥0 表示某资源的可用数，若 S<0，则其绝对值表示阻塞队列中等待该资源的进程数。

对系统中的每个进程，其工作的正确与否不仅取决于它自身的正确性，而且与它在执行中能否与其他相关进程正确地实施同步互斥有关。PV 操作是实现进程同步与互斥的常用方法。P 操作和 V 操作是低级通信原语，在执行期间不可分割。其中，P 操作表示申请一个资源，V 操作表示释放一个资源。

P 操作的定义：S:=S–1，若 S≥0，则执行 P 操作的进程继续执行；若 S<0，则置该进程为阻塞状态（因为无可用资源），并将其插入阻塞队列。P 操作可用如下过程表示，其中 Semaphore 表示所定义的变量是信号量。

```
Procedure P(Var S:Semaphore);
  Begin
    S:=S–1;
    If S<0 then W(S)          {执行 P 操作的进程插入等待队列}
  End;
```

V 操作定义：S: =S+1，若 S>0，则执行 V 操作的进程继续执行；若 S≤0，则从阻塞状态唤醒一个进程，并将其插入就绪队列，然后执行 V 操作的进程继续。

V 操作可用如下过程表示。

```
Procedure V(Var S:Semaphore);
  Begin
    S:=S+1;
    If S≤0 then R(S)          {从阻塞队列中唤醒一个进程}
  End;
```

2）利用 PV 操作实现进程的互斥

令信号量 mutex 的初值为 1，当进入临界区时执行 P 操作，退出临界区时执行 V 操作。这

样，利用 PV 操作实现进程互斥的代码段如下：

```
P(mutex)
  临界区
V(mutex)
```

【例 3.1】 将交通流量统计程序改写如下。

```
                P1                                    P2
L1: if 有车通过 then                    L2: begin
    begin                                     P(mutex)
      P(mutex)                                PRINT COUNT;
      COUNT:=COUNT+1;                         COUNT:=0;
      V(mutex)                                V(mutex)
end                                       end
GOTO L1;                                  GOTO L2;
```

3）利用 PV 操作实现进程的同步

进程的同步是由于进程间合作引起的相互制约的问题，要实现进程的同步可用一个信号量与消息联系起来，当信号量的值为 0 时表示希望的消息未产生，当信号量的值为非 0 时表示希望的消息已经存在。假定用信号量 S 表示某条消息，进程可以通过调用 P 操作测试消息是否到达，调用 V 操作通知消息已准备好。最典型的是单缓冲区的生产者和消费者的同步问题。

【例 3.2】 生产者进程 P_1 不断地生产产品送入缓冲区，消费者进程 P_2 不断地从缓冲区中取产品消费。为了实现 P_1 与 P_2 进程间的同步问题，需要设置一个信号量 S_1，且初值为 1，表示缓冲区空，可以将产品送入缓冲区；设置另一个信号量 S_2，且初值为 0，表示缓冲区有产品。其同步过程如图 3-6 所示。若信号量 S_1 的初值可以设为 0，此时同步过程如图 3-7 所示。

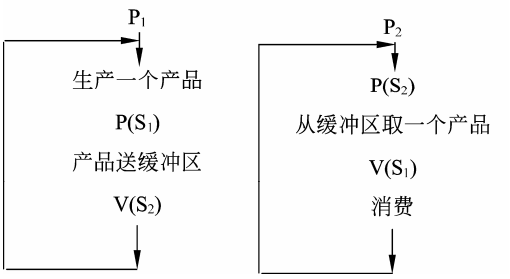


图 3-6 单缓冲区的同步举例方法 1

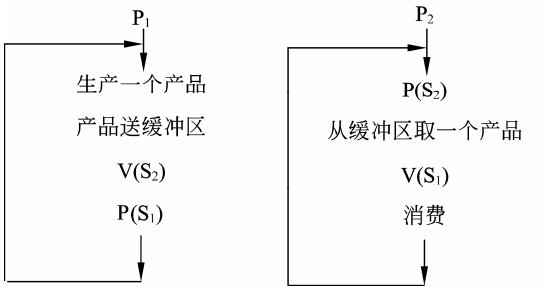


图 3-7 单缓冲区的同步举例方法 2

【例 3.3】 一个生产者和一个消费者，缓冲区可存放 *n* 件物品。生产者不断地生产产品，

消费者不断地消费产品。如何用 PV 操作实现生产者和消费者的同步。可以通过设置三个信号量 S、S_1 和 S_2，其中，S 是一个互斥信号量，初值为 1，因为缓冲区是一个互斥资源，所以需要进行互斥控制；S_1 表示是否可以将物品放入缓冲区，初值为 n；S_2 表示缓冲区是否存有物品，初值为 0。同步过程如图 3-8 所示。

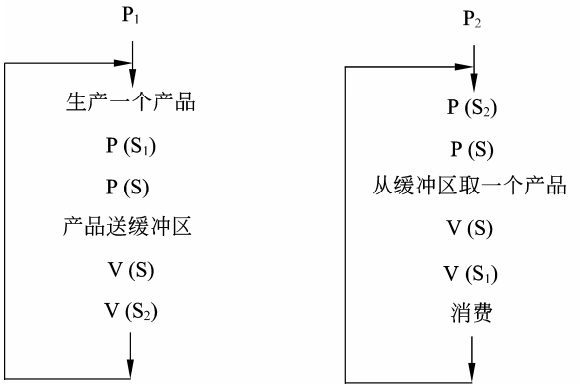


图 3-8　n 个缓冲区的同步举例

3．高级通信原语

进程间通信是指进程之间的信息交换，少则一个状态，多则成千上万个信息。根据交换信息量的多少和效率的高低，进程通信的方式分为低级方式和高级方式。PV 操作属于低级通信方式，若用 PV 操作实现进程间通信，则存在如下问题。

（1）编程难度大，通信对用户不透明，即要用户利用低级通信工具实现进程间的同步与互斥。但是，PV 操作使用不当容易引起死锁。

（2）效率低，生产者每次只能向缓冲区放一个消息，消费者只能从缓冲区取一个消息。

为了提高信号通信的效率，传递大量数据，减轻程序编制的复杂度，系统引入了高级通信方式。高级通信方式主要分为共享存储模式、消息传递模式和管道通信。

（1）共享存储模式。相互通信的进程共享某些数据结构（或存储区）实现进程之间的通信。

（2）消息传递模式。进程间的数据交换以消息为单位，程序员直接利用系统提供的一组通信命令（原语）来实现通信。如 Send(A)、Receive(A)。

（3）管道通信。所谓管道，是指用于连接一个读进程和一个写进程，以实现它们之间通信的共享文件（pipe 文件）。向管道（共享文件）提供输入的发送进程（即写进程），以字符流的形式将大量的数据送入管道；而接收进程可从管道接收大量的数据。由于它们通信时采用管道，所以叫管道通信。

3.2.4 管程

1．管程的引入

用信号量和 P、V 操作来解决进程的同步与互斥问题，需在程序中适当位置安排 P、V 操作，否则会造成死锁错误。为了解决分散编程带来的困难，1974 年和 1975 年汉森（Brinsh Hansen）和霍尔（Hoare）提出了另一种同步机制——管程（monitor）。其基本思路是采用资源集中管理的方法，将系统中的资源用某种数据结构抽象地表示出来。由于临界区是访问共享资源的代码段，建立一个管程管理进程提出的访问请求。

采用这种方式对共享资源的管理就可以借助数据结构及在其上实施操作的若干过程来进行，对共享资源的申请和释放可以通过过程在数据结构上的操作来实现。

管程是由一些共享数据、一组能为并发进程所执行的作用在共享数据上的操作的集合、初始代码以及存取权组成。管程提供了一种可以允许多进程安全有效地共享抽象数据类型的机制，管程实现同步机制由“条件结构（condition construct）”所提供。为实现进程互斥同步，必须定义一些条件变量，例如 var notempty、notfull：condition，这些条件变量只能被 wait 和 signal 操作所访问。Notfull.wait 操作意味着调用该操作的进程将被挂起，使另一个进程执行；而 notfull.signal 操作仅仅是启动一个被挂起的进程，如无挂起进程则 notfull.signal 操作相当于空操作，不改变 notfull 状态，这不同于 V 操作。

2．管程的结构

每一个管程都要有一个名字以供标识，用语言来写一个管程的形式如下：

```
Type <管程名>=monitor
     <管程变量说明>;
     define < (能被其他模块引用的) 过程名列表>;
     procedure   <过程名> (<形式参数表>)
       begin
         <过程体>;
       end;
       …
     procedure   <过程名> (<形式参数表>)
       begin
         <过程体>;
       end;
begin
     <管程的局部数据初始化语句>;
end.
```

3．利用管程解决生产者—消费者问题

【例 3.4】 建立一个管程 Producer-Consumer，它包括两个过程 put（item）和 get（item），分别执行将生产的消息放入缓冲池和从缓冲池取出消息的操作，设置变量 count 表示缓冲池已存消息的数目。管程描述如下：

```
Type Producer-Consumer=monitor
        var    buffer : array [ 0 , … ,n-1] of item ; {定义缓冲区}
        in , out , count : integer ;                  {in out 为存取指针, count 为缓冲区产品个数}
        notfull ,notempty :condition ;                {条件变量}
   procedure    entry    put (item)
      begin
        if    count >= n    then    notfull.wait ;          {缓冲区已满}
           buffer ( in ) : = nextp ;
           in := (in+1) mod n ;
           count = count + 1 ;
        if notempty.queue    then notempty.signal ;    {唤醒等待者}
      end
   procedure entry    get ( item)
      begin
        if count <= 0 then    notempty.wait ;          {缓冲区已空}
           nextc := buffer ( out ) ;
           out := (out+1) mod n ;
           count := count - 1 ;                        {减少一个产品}
        if    notfull.queue then notfull.signal;     {唤醒等待者}
      end
   begin    in := out := 0 ;    count := 0;    end        {初始化}
```

利用管程解决生产者—消费者问题，其中生产者和消费者程序如下：

```
   cobegin
        producer : begin
                        repeat
                        produce an    item    in nextp ;
                        Producer-Consumer.put ( item) ;
                        until false;
                      end
        consumer : begin
```

```
                repeat
                Producer-Consumer.get (item) ;
                consume the item in nextc ;
                until false ;
              end;
coend
```

3.2.5 进程调度

进程调度方式是指当有更高优先级的进程到来时如何分配 CPU。调度方式分为可剥夺和不可剥夺两种。可剥夺式是指当有更高优先级的进程到来时，强行将正在运行进程的 CPU 分配给高优先级的进程；不可剥夺式是指当有更高优先级的进程到来时，必须等待正在运行进程自动释放占用的 CPU，然后将 CPU 分配给高优先级的进程。

1．三级调度

在某些操作系统中，一个作业从提交到完成需要经历高、中、低三级调度。

（1）高级调度。又称“长调度”或“作业调度”或“接纳调度”，它决定处于输入池中的哪个后备作业可以调入主系统做好运行的准备，成为一个或一组就绪进程。系统中一个作业只需经过一次高级调度。

（2）中级调度。又称“中程调度”或“对换调度”，它决定处于交换区中的就绪进程哪个可以调入内存，以便直接参与对 CPU 的竞争。在内存资源紧张时，为了将进程调入内存，必须将内存中处于阻塞状态的进程调出至交换区，以便为调入进程腾出空间。这相当于使处于内存的进程和处于盘交换区的进程交换了位置。

（3）低级调度。又称“短程调度”或“进程调度”，它决定处于内存中的就绪进程哪个可以占用 CPU。低级调度是操作系统中最活跃、最重要的调度程序，对系统的影响很大。

2．调度算法

常用的进程调度算法有先来先服务、时间片轮转、优先级调度和多级反馈调度算法。

（1）先来先服务（FCFS）。FCFS 按照作业提交或进程成为就绪状态的先后次序分配 CPU，即进程调度总是将就绪队列队首的进程投入运行。FCFS 的特点是比较有利于长作业，而不利于短作业；有利于 CPU 繁忙的作业，而不利于 I/O 繁忙的作业。FCFS 算法主要用于宏观调度。

（2）时间片轮转。时间片轮转算法主要用于微观调度，其设计目标是提高资源利用率。通过时间片轮转，提高进程并发性和响应时间特性，从而提高资源利用率。时间片的长度可从几个 ms 到几百 ms，选择的方法一般有如下几种。

① 固定时间片。分配给每个进程相等的时间片，使所有进程都公平执行，是一种实现简

单又有效的方法。

② 可变时间片。根据进程不同的要求对时间片的大小实时修改，可以更好地提高效率。

（3）优先级调度。优先级调度算法是让每一个进程都拥有一个优先数，数值大的表示优先级高，系统在调度时总选择优先数大的占用 CPU。优先级调度分为静态优先级和动态优先级两种。

- 静态优先级。进程的优先级在创建时确定，直到进程终止都不会改变。根据以下因素确定优先级：进程类型（系统进程优先级较高）、对资源的需求（对 CPU 和内存需求较少的进程优先级较高）、用户要求（紧迫程度和付费多少）。
- 动态优先级。在创建进程时赋予一个优先级，在进程运行过程中还可以改变，以便获得更好的调度性能。例如，在就绪队列中，随着等待时间增长，优先级将提高。这样，对于优先级较低的进程在等待足够的时间后，其优先级提高到可被调度执行。进程每执行一个时间片，就降低其优先级，从而当一个进程持续执行时，其优先级会降低到让出 CPU。

（4）多级反馈调度。多级反馈队列算法是时间片轮转算法和优先级算法的综合与发展。其优点是照顾了短进程以提高系统吞吐量、缩短了平均周转时间；照顾 I/O 型进程以获得较好的 I/O 设备利用率和缩短响应时间；不必估计进程的执行时间，动态调节优先级。该算法如图 3-9 所示，其实现思路如下。

① 设置多个就绪队列。队列 1，队列 2，…，队列 *n* 分别赋予不同的优先级，队列 1 的优先级≥队列 2 的优先级≥…≥队列 *n* 的优先级。每个队列执行时间片的长度也不同，规定优先级越低则时间片越长，如逐级加倍。

② 新进程进入内存后，先投入队列 1 的末尾，按 FCFS 算法调度；若某进程在队列 1 的一个时间片内未能执行完，则降低投入到队列 2 的末尾，同样按 FCFS 算法调度；如此下去，当进程降低到最后的队列，则按“时间片轮转”算法调度直到完成。

③ 仅当较高优先级的队列为空，才调度较低优先级队列中的进程执行。如果进程执行时有新进程进入较高优先级的队列，则抢先执行新进程，并把被抢先的进程投入原队列的末尾。

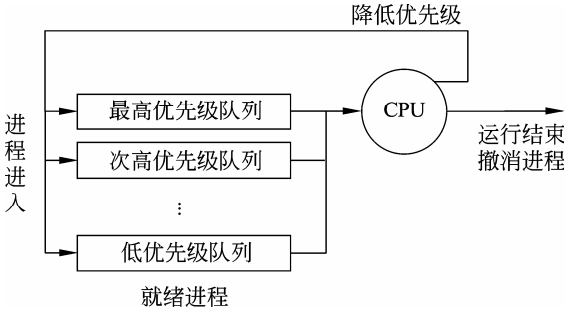


图 3-9　多级队列算法

3．进程优先级确定

优先级确定需要考虑如下情况。

（1）对 I/O 型进程，让其进入最高优先级队列，以及时响应需要 I/O 交互的进程。通常执行一个小的时间片，在该时间片内要求可处理完一次 I/O 请求的数据，然后转入到阻塞队列。

（2）对计算型进程，每次都执行完时间片后进入更低级队列。最终采用最大时间片来执行，减少调度次数。

（3）对 I/O 次数不多，而主要是 CPU 处理的进程。在 I/O 完成后，放回优先 I/O 请求时离开的队列，以免每次都回到最高优先级队列后再逐次下降。

（4）为适应一个进程在不同时间段的运行特点，I/O 完成时，提高优先级；时间片用完时，降低优先级。

3.2.6 死锁

在计算机系统中有许多互斥资源（如磁带机、打印机和绘图仪等）或软件资源（如进程表、临界区等），若两个进程同时使用打印机，或同时进入临界区必然会出现问题。所谓死锁，是指两个以上的进程互相都要求对方已经占有的资源导致无法继续运行下去的现象。

1．死锁举例

【例 3.5】 进程推进顺序不当引起的死锁。设系统中有一台读卡机 A，一台打印机 B，它们被进程 P_1 和 P_2 共享，两个进程并发执行，它们按下列顺序请求和释放资源。

P_1		P_2	
...		...	
Request(A);	<a>	Request(B);	<a>
Request(B);	<b>	Request(A);	<b>
Release(B);		Release(A);	
Release(A);		Release(B);	

假如按 P_1<a> P_2<a> P_1<b> P_2<b>的次序执行，则系统会发生死锁。因为进程 P_1<a>时，由于读卡机未被占用，所以请求可以得到满足；进程 P_2<a>时，由于打印机未被占用，所以请求也可以得到满足。接着进程 P_1<b>时，由于打印机被占用，所以请求得不到满足，P_1 等待；进程 P_2<b>时，由于读卡机被占用，所以请求得不到满足，P_2 也等待。导致互相在请求对方已占有的资源，系统发生死锁。

【例 3.6】 同类资源分配不当引起死锁。若系统中有 m 个资源被 n 个进程共享，当每个进程都要求 k 个资源，而 $m<nk$ 时，即资源数小于进程所要求的总数时，可能会引起死锁。例如，

m=5，*n*=3，*k*=3，若系统采用的分配策略是轮流地为每个进程分配，则第一轮系统先为每个进程分配 1 台，还剩下 2 台；第二轮系统再为两个进程各分配 1 台，此时，系统中已无可供分配的资源，使得各个进程都处于等待状态导致系统发生死锁。

【例 3.7】 PV 操作使用不当引起的死锁。对于图 3-10，当信号量 $S_1=S_2=0$ 时，将发生死锁。

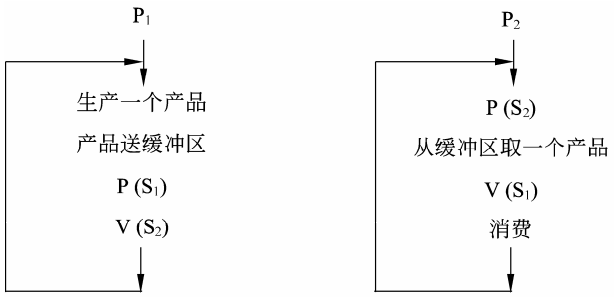


图 3-10　PV 操作引起的死锁

从图 3-10 可见，P_2 进程从缓冲区取产品前，先执行 $P(S_2)$，由于 S_2=–1 故 P_2 等待；P_1 进程将产品送到缓冲区后，执行 $P(S_1)$，由于 S_1=–1，故 P_1 等待。这样，P_1、P_2 进程都无法继续运行下去，导致系统死锁。

2．死锁产生的原因及 4 个必要条件

可以看出，产生死锁的原因为竞争资源及进程推进顺序非法。当系统中有多个进程所共享的资源，不足以同时满足它们的需求时，引起它们对资源的竞争导致死锁。进程推进顺序非法，进程在运行的过程中，请求和释放资源的顺序不当，导致进程死锁。产生死锁的 4 个必要条件是互斥条件、请求保持条件、不可剥夺条件和环路条件。

当发生死锁时，在进程资源有向图中必构成环路，其中每个进程占有了下一个进程申请的一个或多个资源，如图 3-11 所示。

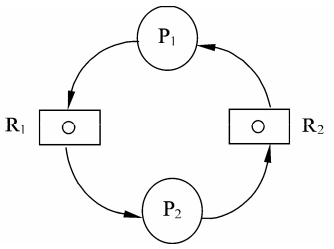


图 3-11　进程资源有向图

进程资源有向图由方框、圆圈和有向边三部分组成。其中方框表示资源，圆圈表示进程。请求资源：○→□，箭头由进程指向资源；分配资源：○←□，箭头由资源指向进程。

3．死锁的处理

死锁的处理策略主要有 4 种：鸵鸟策略（即不理睬策略）、预防策略、避免策略和检测与

解除死锁。

1）死锁预防

死锁预防是采用某种策略，限制并发进程对资源的请求，破坏死锁产生的 4 个必要条件之一，使系统在任何时刻都不满足死锁的必要条件。预防死锁的两种策略如下。

（1）预先静态分配法。破坏了“ 不可剥夺条件 ”。预先分配所需资源，保证不等待资源。该方法的问题是降低了对资源的利用率，降低进程的并发程度；有时可能无法预先知道所需资源。

（2）资源有序分配法。破坏了“ 环路条件 ”。把资源分类按顺序排列，保证不形成环路。该方法存在的问题是限制进程对资源的请求；由于资源的排序占用系统开销。

2）死锁避免

死锁预防是设法破坏产生死锁的 4 个必要条件之一，严格防止死锁的产生。死锁避免则不那么严格地限制产生死锁的必要条件。最著名的是 Dijkstra 提出的银行家算法。死锁避免算法需要很大的系统开销。

银行家算法对于进程发出的每一个系统可以满足的资源请求命令加以检测，如果发现分配资源后，系统进入不安全状态，则不予分配；若分配资源后系统仍处于安全状态，则实施分配。与死锁预防策略相比提高了资源的利用率，但增加了系统开销。

所谓安全状态，是指系统能按某种顺序如 $< P_1, P_2, \cdots, P_n >$ 来为每个进程分配其所需资源，直至最大需求，使每个进程都可顺序完成。通常称 $< P_1, P_2, \cdots, P_n >$ 序列为安全序列。若系统不存在这样一个安全序列，则称系统处于不安全状态。

【例 3.8】 假设系统中有三类互斥资源 R_1、R_2 和 R_3，可用资源数分别为 8，7 和 4。在 T_0 时刻系统中有 P_1、P_2、P_3、P_4 和 P_5 这 5 个进程，这些进程对资源的最大需求量和已分配资源数如图 3-12 所示。

资源 / 进程	最大需求量			已分配资源数		
	R_1	R_2	R_3	R_1	R_2	R_3
P_1	6	4	2	1	1	1
P_2	2	2	2	2	1	1
P_3	8	1	1	2	1	0
P_4	2	2	1	1	2	1
P_5	3	4	2	1	1	1

图 3-12　进程已分配资源数

若有如下 4 个执行序列，那么进程按什么序列执行，系统状态是安全的。

① $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_3$　　② $P_2 \rightarrow P_1 \rightarrow P_4 \rightarrow P_5 \rightarrow P_3$

③ $P_4 \rightarrow P_2 \rightarrow P_1 \rightarrow P_5 \rightarrow P_3$　　④ $P_4 \rightarrow P_2 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

解：初始时系统的可用资源数分别为 8，7 和 4，在 T_0 时刻已分配资源数分别为 7，6 和 4，因此系统剩余的可用资源数分别为 1，1 和 0。

由于 R_3 资源为 0，系统不能在分配 R_3 资源了，所以不能一开始就运行需要分配 R_3 资源的进程 P_1 和 P_2，故①和②显然是不安全的。

分析序列③的 $P_4 \rightarrow P_2 \rightarrow P_1 \rightarrow P_5 \rightarrow P_3$ 是否安全。进程 P_4 可以设置能完成标志 True，如图 3-13 所示。

资源 / 进程	可用			需求			已分			可用+已分			能否完成标志
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	
P_4	1	1	0	1	0	0	1	2	1	2	3	1	True
P_2	2	3	1	0	1	1	2	1	1	4	4	2	True
P_1	4	4	2	5	3	1	1	1	1				

图 3-13　进程按序列③的 $P_4 \rightarrow P_2 \rightarrow P_1 \rightarrow P_5 \rightarrow P_3$ 执行

因为系统的可用资源数为（1，1，0），而进程 P_4 只需要一台 R_1 资源；进程 P_2 可以设置能完成标志 True，因为进程 P_4 运行完毕将释放所有资源，此时系统的可用资源数应为（2，3，1），而进程 P_2 只需要（0，1，1），进程 P_2 运行完毕将释放所有资源，此时系统的可用资源数应为（4，4，2）；进程 P_1 不能设置能完成标志 True，因为进程 P_1 需要 R_1 资源为 5，系统能提供的 R_1 资源为 4，所以序列无法进行下去，因此，$P_4 \rightarrow P_2 \rightarrow P_1 \rightarrow P_5 \rightarrow P_3$ 为不安全序列。

序列④的 $P_4 \rightarrow P_2 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$ 是安全的，因为所有的进程都能设置完成标志 True，如图 3-14 所示。

资源 / 进程	可用			需求			已分			可用+已分			能否完成标志
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	
P_4	1	1	0	1	0	0	1	2	1	2	3	1	True
P_2	2	3	1	0	1	1	2	1	1	4	4	2	True
P_5	4	4	2	2	3	1	1	1	1	5	5	3	True
P_1	5	5	3	5	3	1	1	1	1	6	6	4	True
P_3	6	6	4	6	0	1	2	1	0	8	7	4	True

图 3-14　进程按序列④的 $P_4 \rightarrow P_2 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$ 执行

3）死锁检测

解决死锁的另一条途径是死锁检测方法，这种方法对资源的分配不加限制，即允许死锁产生。但系统定时地运行一个死锁检测程序，判断系统是否发生死锁，若检测到有死锁，则设法加以

解除。

4）死锁解除

死锁解除通常采用如下方法。

（1）资源剥夺法。从一些进程那里强行剥夺足够数量的资源分配给死锁进程。

（2）撤销进程法。根据某种策略逐个地撤销死锁进程，直到解除死锁为止。

3.2.7 线程

传统的进程有两个基本属性：可拥有资源的独立单位；可独立调度和分配的基本单位。由于在进程的创建、撤销和切换中，系统必须为之付出较大的时空开销，因此在系统中设置的进程数目不宜过多，进程切换的频率不宜太高，这就限制了并发程度的提高。引入线程后，将传统进程的两个基本属性分开，线程作为调度和分配的基本单位，进程作为独立分配资源的单位。用户可以通过创建线程来完成任务，以减少程序并发执行时付出的时空开销。

例如，在文件服务进程中可设置多个服务线程，当一个线程受阻时，第二个线程可以继续运行，当第二个线程受阻时，第三个线程可以继续运行……从而显著地提高了文件系统的服务质量及系统的吞吐量。

这样，对拥有资源的基本单位，不用频繁对其切换，进一步提高了系统中各程序的并发程度。需要说明的是，线程是进程中的一个实体，是被系统独立分配和调度的基本单位。线程基本上不拥有资源，只拥有一点运行中必不可少的资源（如程序计数器、一组寄存器和栈），它可与同属一个进程的其他线程共享进程所拥有的全部资源。

线程也具有就绪、运行和阻塞三种基本状态。由于线程具有许多传统进程所具有的特性，故称为“轻型进程（Light-Weight Process）”；传统进程称之为“重型进程（Heavy-Weight Process）”。线程可创建另一个线程，同一个进程中的多个线程可并发执行。

线程分为用户级线程（User-Level Threads）和内核支持线程（Kernel-Supported Threads）两类。用户级线程不依赖于内核，该类线程的创建、撤销和切换都不利用系统调用来实现；内核支持线程依赖于内核，即无论是在用户进程中的线程，还是在系统中的线程，它们的创建、撤销和切换都利用系统调用来实现。某些系统同时实现了两种类型的线程。

与线程不同的是，不论是系统进程还是用户进程，在进行切换时，都要依赖于内核中的进程调度。因此，不论是什么进程都是与内核有关的，是在内核支持下进行切换的。尽管程序和进程表面上看起来相似，但它们在本质上是不同的。

3.3 存储管理

存储器管理的对象是主存，也称内存。存储器是计算机系统中的关键性资源，是存放各种

信息的主要场所。特别是近几年来，系统软件、应用软件在功能及其所需存储空间等方面都在急剧膨胀，如何对存储器实施有效的管理，不仅直接影响到存储器的利用率，而且还对系统性能有重大的影响。其主要功能包括分配和回收主存空间、提高主存的利用率、扩充主存、对主存信息实现有效保护。

3.3.1 基本概念

1．存储器的结构

存储器的功能是保存数据，存储器的发展方向是高速、大容量和小体积。一般存储器的结构有“寄存器—主存—外存”结构和“寄存器—缓存—主存—外存”结构(如图3-15所示)。存储组织的功能是在存储技术和 CPU 寻址技术许可的范围内组织合理的存储结构，使得各层次的存储器都处于均衡的繁忙状态。

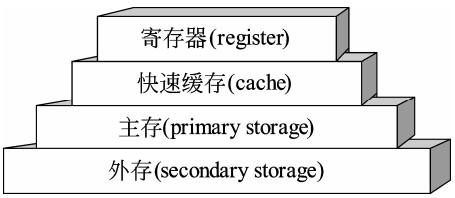


图 3-15　存储器的层次结构

（1）虚拟地址。对程序员来说，数据的存放地址是由符号决定的，故称符号名地址，或者称为名地址，而把源程序的地址空间叫做符号名地址空间或者名空间。它是从 0 号单元开始编址，并顺序分配所有的符号名所对应的地址单元，所以它不是主存中的真实地址，故称为相对地址、程序地址、逻辑地址或称虚拟地址。

（2）地址空间。把程序中由符号名组成的空间称为名空间。源程序经过汇编或编译后再经过链接编辑程序加工形成程序的装配模块，即转换为相对地址编址的模块，它是以 0 为基址顺序进行编址的。相对地址也称为逻辑地址或虚地址，把程序中由相对地址组成的空间叫做逻辑地址空间。相对地址空间通过地址再定位机构转换到绝对地址空间，绝对地址空间也叫物理地址空间。

（3）存储空间。简单来说，逻辑地址空间（简称地址空间）是逻辑地址的集合，物理地址空间（简称存储空间）是物理地址的集合。

2．地址重定位

地址重定位是指将逻辑地址变换成主存物理地址的过程。在可执行文件装入时，需要解决可执行文件中地址（指令和数据）与主存地址的对应关系，由操作系统中的装入程序 Loader 和地址重定位机构来完成。地址重定位分为静态地址重定位和动态地址重定位。

（1）静态重定位。静态重定位是指在程序装入主存时已经完成了逻辑地址到物理地址的变换，在程序的执行期间将不会再发生变化。静态地址重定位的优点是无须硬件地址变换机构的

支持，它只要求程序本身是可重定位的，只对那些要修改的地址部分具有某种标识，由专门设计的程序来完成。在早期的操作系统中多数都采用这种方法。静态重定位的缺点是必须给作业分配一个连续的存储区域，在作业的执行期间不能扩充存储空间，也不能在主存中移动，多个作业也难以共享主存中的同一程序副本和数据。

（2）动态重定位。动态重定位是指在程序运行期间完成逻辑地址到物理地址的变换。其实现机制要依赖硬件地址变化机构，如基地址寄存器（BR）。动态地址重定位的优点是程序在执行期间可以换入和换出主存，以解决主存紧张；可以在主存中移动，把主存中的碎片集中起来，以充分利用空间；不必给程序分配连续的主存空间，可以较好地利用较小的主存块；可以实现共享。

3.3.2 存储管理方案

存储管理的主要目的是解决多个用户使用主存的问题，其存储管理方案主要包括分区存储管理、分页存储管理、分段存储管理、段页式存储管理以及虚拟存储管理。本小节介绍分区存储管理方案，其他存储管理方案在后续章节中介绍。

分区存储管理是早期的存储管理方案，其基本思想是把主存的用户区划分成若干个区域，每个区域分配给一个用户作业使用，并限定它们只能在自己的区域中运行，这种主存分配方案就是分区存储管理方式。按分区的划分方式不同可分为固定分区、可变分区和可重定位分区。

（1）固定分区。固定分区是一种静态分区方式，在系统生成时已将主存划分为若干个分区，每个分区的大小可不等。操作系统通过主存分配情况表管理主存。这种方法的突出问题是已分配区中存在未用空间，原因是程序或作业的大小不可能刚好等于分区的大小，故造成了空间的浪费。通常将已分配分区内的未用空间叫做零头或内碎片。

（2）可变分区。可变分区是一种动态分区方式，存储空间的划分是在作业装入时进行的，故分区的个数是可变的，分区的大小刚好等于作业的大小。可变分区分配需要两种管理表格：已分配表，记录已分配分区的情况；未分配表，记录未分配分区的情况。

对于可变分区的请求和释放分区主要有如下 4 种算法。

① 最佳适应算法。假设系统中有 n 个空白区（自由区），每当用户申请一个空间时，将从这 n 个空白区中找到一个最接近用户需求的分区。这种算法能保留较大的空白区。但缺点是空闲区不可能刚好等于用户要求的区，所以必然要将一个分区一分为二，可是随着系统不断地释放空间，可能会使产生的小分区小到无法再继续分配，将这样的无用小分区称之为外碎片。

② 最差适应算法。系统总是将用户作业装入最大的空白分区。这种算法将一个最大的分区一分为二，所以剩下的空白区通常也大，不容易产生外碎片。

③ 首次适应算法。每当用户作业申请一个空间时，系统总是从主存的低地址开始选择一个能装入作业的空白区。当用户释放空间时，该算法更易实现相邻的空白区合并。

④ 循环首次适应算法。与首次适应算法的不同之处是，每次分配都是从刚分配的空白区开始寻找一个能满足用户要求的空白区。

引入可变分区后虽然主存分配更灵活，也提高了主存利用率，但是由于系统在不断地分配和回收中，必定会出现一些不连续的小的空闲区，尽管这些小的空闲区的总和超过某一个作业要求的空间，但是由于不连续而无法分配，产生了未分配区的无用空间，通常称之为外碎片。解决碎片的方法是拼接（或称紧凑），即向一个方向（例如向低地址端）移动已分配的作业，使那些零散的小空闲区在另一个方向连成一片。

（3）可重定位分区。可重定位分区是解决碎片问题的简单而又行之有效的方法。基本思想是移动所有已分配好的分区，使之成为连续区域。如同队列有一个队员出列，指挥员叫大家“靠拢”一样。分区“靠拢”的时机是当用户请求空间得不到满足时或某个作业执行完毕时。由于靠拢是要代价的，所以通常是在用户请求空间得不到满足时进行。需要注意的是当进行分区“靠拢”时会导致地址发生变化，所以有地址重定位问题。

分区保护的目的是防止未经核准的用户访问分区，常用如下两种方式。

（1）采用上界/下界寄存器保护。上界寄存器中存放的是作业的装入地址，下界寄存器装入的是作业的结束地址，形成的物理地址必须满足如下条件：

上界寄存器≤物理地址≤下界寄存器

（2）采用基址/限长寄存器保护。基址寄存器中存放的是作业的装入地址，限长寄存器装入的是作业的长度，形成的物理地址必须满足如下条件：

基址寄存器≤物理地址< 基址寄存器+限长寄存器

3.3.3 分页存储管理

尽管分区管理方案是解决多道程序共享主存的可行方案，但是该方案的主要问题是用户程序必须装入连续的地址空间中，若无满足用户要求的连续空间时，需要进行分区靠拢操作，这是以耗费系统时间为代价的。为此，引入了分页存储管理方案。

1．纯分页存储管理

1）分页原理

将一个进程的地址空间划分成若干个大小相等的区域，称为页。相应地，将主存空间划分成与页相同大小的若干个物理块，称为块或页框。在为进程分配主存时，将进程中若干页分别装入多个不相邻接的块中。

2）地址结构

分页系统的地址结构如图 3-16 所示，它由两部分组成：前一部分为页号 P；后一部分为偏移量 W，即页内地址。图中的地址长度为 32 位，其中 0～11 位为页内地址（每页的大小为 4KB），

12～31 位为页号，所以允许地址空间的大小最多为 1MB 个页。

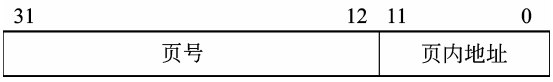


图 3-16　分页地址结构

3）页表

在将进程的每一页离散地分配到主存的多个物理块中后，系统应能保证在主存中找到每个页面所对应的物理块。为此，系统为每个进程建立了一张页面映射表，简称页表（如图 3-17 所示）。每个页在页表中占一个表项，记录该页在主存中对应的物理块号。进程在执行时，通过查找页表就可以找到每页所对应的物理块号。图中逻辑页号为 4，查找的物理块号为 15，与页内地址 256 拼接得到物理地址。可见，页表的作用是实现从页号到物理块号的地址映射。

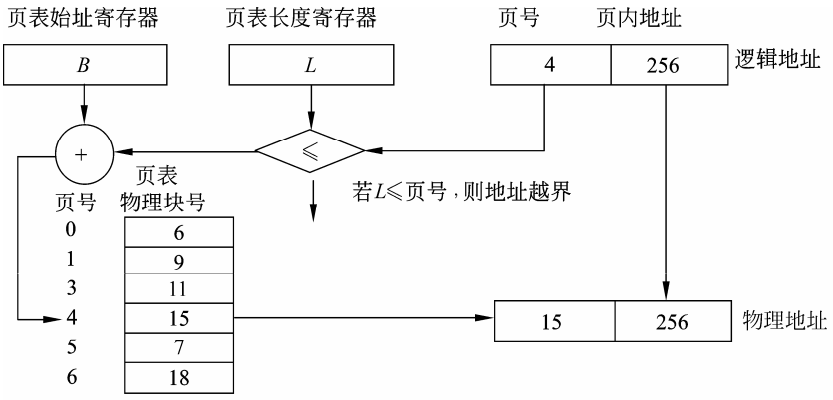


图 3-17　页式存储管理的地址映射

地址变换机构的基本任务是利用页表把用户程序中的逻辑地址变换成主存中的物理地址，实际上就是将用户程序中的页号变换成主存中的物理块号。为了实现地址变换功能，在系统中设置页表寄存器，用来存放页表的始址和页表的长度。在进程未执行时，每个进程对应的页表的始址和长度存放在进程的 PCB 中，当该进程被调度时，就将它们装入页表寄存器。在进行地址变换时，系统将页号与页表长度进行比较，如果页号大于等于页表寄存器中的页表长度 L（页号从 0 开始），则访问越界，产生越界中断。若未出现越界，则根据页表寄存器中的页表始址和页号计算出该页在页表项中的位置，得到该页的物理块号，将此物理块号装入物理地址寄存器中。与此同时，将有效地址（逻辑地址）寄存器中页内地址直接装入物理地址寄存器的块内地址字段中，这样便完成了从 p 逻辑地址到物理地址的变换。

2．快表

从地址映射的过程可以发现，页式存储管理至少需要两次访问主存。例如，第一次是访问页表，得到的是数据的物理地址；第二次是存取数据。若数据是间接地址，还需要再进行地址变换。为了提高访问主存的速度，可以在地址映射机构中增加一组高速寄存器，用来保存页表。这种方法需要大量的硬件开销，经济上是不可行的。另一种方法是在地址映射机构中增加一个小容量的联想存储器，联想存储器由一组高速存储器组成，称之为快表，用来保存当前访问频率高的少数活动页的页号及相关信息。

联想存储器存放的只是当前进程最活跃的少数几页，因此，用户程序要访问数据时，根据该数据所在逻辑页号在联想存储器中找出对应的物理页号，然后与页内地址拼接形成物理地址；若找不到相应的逻辑页号，则地址映射仍通过主存的页表进行，得到物理地址后，须将物理块号填入联想存储器的空闲单元中，若无空闲单元，则根据淘汰算法淘汰某一行，再填入新得到的页号。事实上，查找联想存储器和查找主存页表是并行进行的，一旦在联想存储器中找到相符的逻辑页号时，就停止查找主存页表。

3．两级页表机制

大家知道，80386 的逻辑地址有 2^{32} 个，若页面大小为 4KB（2^{12}B），则页表项达 1MB 个，每个页表项占用 4B，故每个进程的页表占用 4MB 主存空间，并且还要求是连续的，显然这是不现实的。为了减少页表所占用的连续的主存空间，在 80386 中采用了两级页表机制。基本方法是将页表进行分页，每个页面的大小与主存物理块的大小相同，并为它们进行编号，可以离散地将各个页面分别存放在不同的物理块中。为此需要建立一张页表，称为外层页表（页表目录），即第一级页表，其中的每个表目是存放某个页表的物理地址。第二级是页表，其中的每个表目所存放的是页的物理块号。

两级页表的逻辑地址结构和两级页表的地址变换机构如图 3-18 所示。

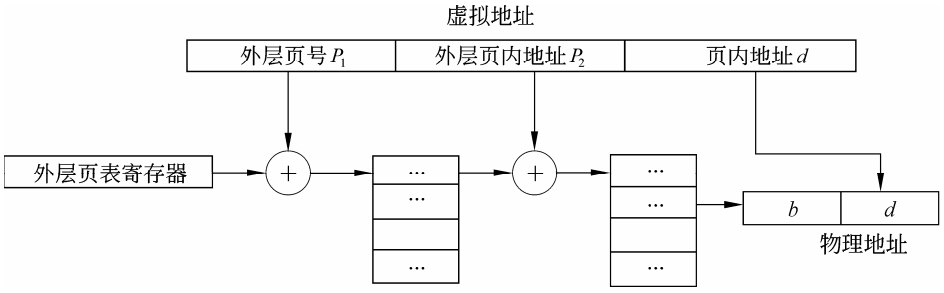


图 3-18　两级页表的地址变换机构

3.3.4 分段存储管理

在分段存储管理方式中，作业的地址空间被划分为若干个段，每个段是一组完整的逻辑信息，如有主程序段、子程序段、数据段及堆栈段等，每个段都有自己的名字，都是从 0 开始编址的一段连续的地址空间，各段长度是不等的。分段系统的地址结构如图 3-19 所示，逻辑地址由段号（名）和段内地址两部分组成。在该地址结构中，允许一个作业最多有 64 KB 个段，每个段的最大长度为 64 KB。

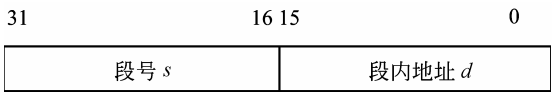


图 3-19 分段的地址结构

在分段式存储管理系统中，为每个段分配一个连续的分区，而进程中的各个段可以离散地分配到主存的不同分区中。在系统中为每个进程建立一张段映射表，简称为“段表”。每个段在表中占有一个表项，在其中记录了该段在主存中的起始地址（又称为“基址”）和段的长度，如图 3-20 所示。进程在执行时，通过查段表来找到每个段所对应的主存区。可见，段表实现了从逻辑段到物理主存区的映射。

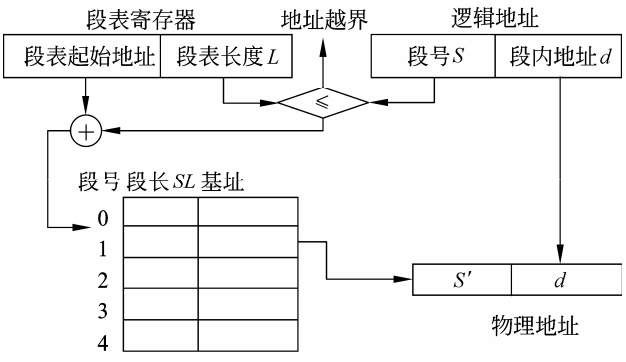


图 3-20 段式存储管理的地址变换机构

为了实现从逻辑地址到物理地址的变换功能，系统中设置了段表寄存器，用于存放段表始址和段表长度。在进行地址变换时，系统将逻辑地址中的段号 S 与段表长度 L 进行比较。若 $S \geq L$，表示段号太大，访问越界，于是产生越界中断信号；若未越界，则根据段表的始址和该段的段号，计算出该段对应段表项的位置，从中读出该段在主存中的起始地址，然后再检查段内地址 d 是否超过该段的段长 SL。若超过，即 $d \geq SL$，同样发出越界中断信号；若未越界，则

将该段的基址 *S'*与段内地址 *d* 相加，得到要访问的主存物理地址。

【例 3.9】 设某作业的段表如下：

段号	基地址	段长
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

（1）逻辑地址（0，128）、（1，30）、（2，88）、（3，290）和（4，100）能否转换为对应的物理地址？为什么？

（2）将问题（1）的逻辑地址分别转换成对应的物理地址。

解：

（1）逻辑地址（0，128）、（2，88）和（3，290）可以转换成对应的物理地址，而逻辑地址（1，30）和（4，100）不能转换为对应的物理地址，因为地址越界。

（2）逻辑地址（0，128）对应的物理地址是 219+128=347；

逻辑地址（2，88）对应的物理地址是 90+88=178；

逻辑地址（3，290）对应的物理地址是 1327+290=1617。

3.3.5 段页式存储管理

分页和分段存储管理方式都各有其优缺点。因为分页的过程是由操作系统完成的，对用户是透明的，所以用户不必关心分页的过程，其缺点是不易实现共享；段是信息的逻辑单位，其优点是易于实现段的共享，即允许若干个进程共享一个或多个段，而且对段的保护也十分简单。如果对两种存储管理方式“各取所长”，则可以形成一种新的存储管理方式的系统——段页式系统。这种新系统既具有分页系统能有效地提高主存利用率的优点，又具有分段系统能很好地满足用户需要的长处，显然是一种比较有效的存储管理方式。

段页式系统的基本原理是先将整个主存划分成大小相等的存储块（页架），将用户程序按程序的逻辑关系分为若干个段，并为每个段赋予一个段名，再将每个段划分成若干页，以页架为单位离散分配。在段页式系统中，其地址结构由段号、段内页号和页内地址三部分组成。作业地址空间的结构如图 3-21 所示。

段号 *s*	段内页号 *p*	页内地址 *w*

图 3-21　段页式管理的地址结构

在段页式系统中，为了实现从逻辑地址到物理地址的变换，系统中必须同时配置段表和页表。由于将段中的页进行离散地分配，段表中的内容不再是段的主存始址和段长，而是页表始址和页表长度。在段页式系统中，有一个段表寄存器，存放段表始址和段表长度 *TL*，其地址变换机构如图 3-22 所示。

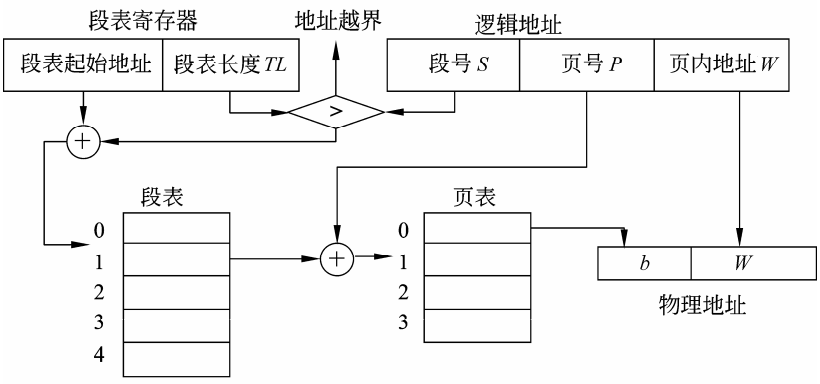


图 3-22　段页式存储管理的地址变换机构

3.3.6　虚拟存储管理

前面介绍的存储管理方案中，　必须为每个作业分配足够的空间，以便装入全部信息。当主存空间不能满足作业要求时，作业便无法装入主存执行。

如果一个作业只部分装入主存便可开始启动运行，其余部分暂时留在磁盘上，需要时再装入主存。这样可以有效地利用主存空间，从用户角度看，该系统所具有的主存容量将比实际主存容量大得多，人们把这样的存储器称为虚拟存储器。虚拟存储器是为了扩大主存容量而采用的一种设计方法，其容量是由计算机的地址结构决定的。

1．程序局部性原理

早在 1968 年 P. Denning 就指出过，程序在执行时将呈现出局部性规律，即在一段时间内，程序的执行仅局限于某个部分。相应地，它所访问的存储空间也局限于某个区域内。程序的局限性表现在时间局限性和空间局限性两个方面。

（1）时间局限性。如果程序中的某条指令一旦执行，则不久的将来该指令可能再次被执行；如果某个存储单元被访问，则不久以后该存储单元可能再次被访问。产生时间局限性的典型原因是在程序中存在着大量的循环操作。

（2）空间局限性。一旦程序访问了某个存储单元，则在不久的将来，其附近的存储单元也

最有可能被访问。即程序在一段时间内所访问的地址，可能集中在一定的范围内，其典型原因是程序是顺序执行的。

2．虚拟存储器的实现

虚拟存储器是具有请求调入功能和置换功能，能仅把作业的一部分装入主存便可运行作业的存储器系统，是能从逻辑上对主存容量进行扩充的一种虚拟的存储器系统。其逻辑容量由主存和外存容量之和以及 CPU 可寻址的范围来决定，其运行速度接近于主存速度，成本也下降。可见，虚拟存储技术是一种性能非常优越的存储器管理技术，故被广泛地应用于大、中、小型机器和微型机中。虚拟存储器实现主要有如下三种方式。

（1）请求分页系统。它是在分页系统的基础上，增加了请求调页功能和页面置换功能所形成的页式虚拟存储系统。它允许只装入若干页的用户程序和数据（而非全部程序）就可以启动运行，以后再通过调页功能和页面置换功能陆续把将要使用的页面调入主存，同时把暂不运行的页面置换到外存上，置换时以页面为单位。

（2）请求分段系统。它是在分段系统的基础上，增加了请求调段和分段置换功能所形成的段式虚拟存储系统。它允许只装入若干段（而非全部段）的用户程序和数据就可以启动运行，以后再通过调段功能和置换功能将不运行的段调出，同时调入将要运行的段，置换是以段为单位。

（3）请求段页式系统。它是在段页式系统的基础上，增加了请求调页和页面置换功能所形成的段页式虚拟存储系统。

3．请求分页管理的实现

请求分页是在纯分页系统的基础上，增加了请求调页功能、页面置换功能所形成的页式虚拟存储系统，是目前常用的一种虚拟存储器的方式。

请求分页的页表机制是在纯分页的页表机制上形成的，由于只将应用程序的一部分调入主存，还有一部分仍在磁盘上，故需在页表中再增加若干项，如状态位、访问字段和辅存地址等供程序（数据）在换进、换出时参考。

请求分页系统中的地址变换机构，是在分页系统的地址变换机构的基础上增加了某些功能，如产生和处理缺页中断、从主存中换出一页实现虚拟存储。

在请求分页系统中，每当所要访问的页面不在主存时，便要产生一个缺页中断，请求 OS 将所缺的页调入主存，这是由缺页中断机构完成的。缺页中断与一般中断的主要区别如下。

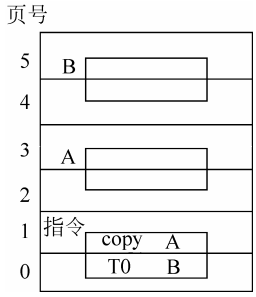
（1）缺页中断在指令执行期间产生和处理中断信号，而一般中断在一条指令执行完，下一条指令开始执行前检查和处理中断信号。

（2）发生缺页中断时，返回到被中断指令的开始重新执行该指令，而一般中断返回到下一

条指令执行。

（3）一条指令在执行期间，可能会产生多次缺页中断。

【例 3.10】 在某计算机中，假设某程序的 COPY 指令跨两个页面，且源地址 A 和目标地址 B 所涉及的区域也跨两个页面，如下图所示。



若地址为 A 和 B 的操作数均不在内存，计算机执行 COPY 指令时，系统将产生__（1）__缺页中断；若系统产生三次缺页中断，那么该程序有__（2）__个页面在内存。

（1）A．2　　B．3　　C．4　　D．5

（2）A．2　　B．3　　C．4　　D．5

例题分析

从例题的图中可见，程序的 COPY 指令跨两个页面，且源地址 A 和目标地址 B 所涉及的区域也跨两个页面页内地址，这时，如果 2，3，4 和 5 号页面不在内存，系统执行“ COPY A TO B ”指令时，取地址为 A 的操作数。由于该操作数不在内存且跨两个页面 2，3，需要将 2，3 页面装入内存，所以产生两次缺页中断。同理，取地址为 B 的操作数，由于该操作数不在内存且跨两个页面 4 和 5，需要将 4 和 5 页面装入内存，所以产生两次缺页中断，共产生 4 次缺页中断。故例题空（1）的正确答案为 C。

同理，如果 1，2，3 号页面不在内存，系统执行“ COPY A TO B ”指令时，由于程序的 COPY 指令跨两个页面，当取出指令分析是多字节的，那么系统将产生一次缺页中断取指令的后半部分；当取地址为 A 的操作数，由于该操作数不在内存，且跨两个页面 2 和 3，需要将 2 和 3 页面装入内存，所以产生两次缺页中断，共产生 3 次缺页中断。故例题空（2）的正确答案为 B。

4．页面置换算法

请求分页是在纯分页系统的基础上，增加了请求调页功能、页面置换功能所形成的页式虚拟存储系统，是目前常用的一种虚拟存储器的方式。在进程运行过程中，如果发生缺页，此时主存中又无空闲块时，为了保证进程能正常运行，就必须从主存中调出一页程序或数据送磁盘的对换区。但究竟将哪个页面调出，需要根据一定的页面置换算法来确定。置换算法的好坏将

直接影响系统的性能，不适当的算法可能会导致系统发生“ 抖动 ”（thrashing)。即刚被换出的页很快又被访问，需重新调入，导致系统频繁地更换页面，以致一个进程在运行中把大部分时间花费在完成页面置换的工作上，这种现象称之为系统发生了“抖动”（也称颠簸)。请求分页系统的核心问题是选择合适的页面置换算法。常用的页面置换算法如下所述。

1）最佳（Optimal）置换算法

这是一种理想化的算法，即选择哪些是永不使用的，或者是在最长时间内不再被访问的页面置换出去。这种方法性能最好，但实际上难于实现。要确定哪一个页面是未来最长时间内不再被访问的是很难的，所以该算法通常用来评价其他算法。

【例 3.11】 假定系统为某进程分配了三个物理块，进程访问页面的顺序为 0，7，6，5，7，4，7，3，5，4，7，4，5，6，5，7，6，0，7，6。如图 3-23 所示，进程运行时先将 0，7，6 三个页面装入主存。当进程访问页面 5 时，产生缺页中断，根据最佳置换算法，页面 0 将在第 18 次才被访问，是三页中将最久不被访问的页面，所以被淘汰。接着访问页面 7 时，发现已在主存中，而不会产生缺页中断，依此类推。从图中可以看出，采用最佳置换算法，产生了 9 次缺页中断，发生了 6 次页面置换。

访问页面	0	7	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
物	0	0	0	5	5	5	5	5	5	5	5	5	5	5	5	5	5	0	0	0
理		7	7	7	7	7	7	3	3	3	7	7	7	7	7	7	7	7	7	7
块			6	6	6	4	4	4	4	4	4	4	4	6	6	6	6	6	6	6
缺页	x	x	x	x		x		x			x			x				x		

图 3-23　最佳置换算法

2）先进先出（FIFO）置换算法

该算法总是淘汰最先进入主存的页面，即选择在主存中驻留时间最久的页面予以淘汰。该算法实现简单，只须把一个进程调入主存的页面，按先后次序链接成一个队列，并设置一个指针即可。它是一种最直观，性能最差的算法，有 BELADY 异常现象：对页面访问序列 A B C D A B E A B C D E，当分配的物理块从 3 块增加到 4 块时，缺页次数反而增加。

【例 3.12】 假定系统中某进程访问页面的顺序为 0，7，6，5，7，4，7，3，5，4，7，4，5，6，5，7，6，0，7，6。利用 FIFO 算法对上例进行页面置换的结果如图 3-24 所示。

访问页面	0	7	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
物	0	0	6	5	7	4	4	3	5	3	7	4	5	6	6	7	7	0	0	0
理		7	0	6	5	7	7	4	3	5	3	7	4	5	5	6	6	7	7	7
块			7	0	6	5	5	7	4	4	5	3	7	4	4	5	5	6	6	6
缺页	x	x	x	x	x	x		x	x		x	x	x	x		x		x		

图 3-24　先进先出置换算法

从图中可以看出，共发生了 11 次页面置换，缺页次数为 14 次。

3）最近最少未使用（Least Recently Used，LRU）置换算法

该算法是选择最近最少未使用的页面予以淘汰，系统在每个页面设置一个访问字段，用以记录这个页面自上次被访问以来所经历的时间 *T*，当要淘汰一个页面时，选择 *T* 最大的页面。但在实现时需要硬件的支持（寄存器或栈）。

【例 3.13】 假定系统中某进程访问页面的顺序为 0，7，6，5，7，4，7，3，5，4，7，4，5，6，5，7，6，0，7，6。利用 LRU 算法对上例进行页面置换的结果如图 3-25 所示。

访问页面	0	7	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
物	0	0	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
理		7	0	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7
块			7	0	6	5	5	4	7	3	5	5	7	4	4	6	5	7	6	0
缺页	x	x	x	x	x	x		x	x	x	x			x		x		x		

图 3-25　最近最少未使用置换算法

从图中可以看出，共发生了 10 次面置换，缺页次数 13 次。

4）最近未用（Not Used Recently，NUR）置换算法

将最近一段时间未引用过的页面换出。这是一种 LRU 的近似算法。该算法为每个页面设置一位访问位，将主存中的所有页面都通过链接指针链成一个循环队列。当某页被访问时，其访问位置 1。在选择一页淘汰时，检查其访问位，如果是 0，则选择该页换出；若为 1，则重新置为 0，暂不换出该页，在循环队列中检查下一个页面，直到访问位为 0 的页面为止。由于该算法只有一位访问位，只能用它表示该页是否已经使用过，而置换时是将未使用过的页面换出去，所以把该算法称为最近未用算法。

5．工作集

程序在运行中所产生的缺页情况，会影响程序的运行速度及系统性能，而缺页率的高低又与每个进程所占用的物理块数目有关。究竟应该为每个进程分配多少个物理块，才能把缺页率保持在一个合理的水平上？否则会因为进程频繁地从辅存请求页面，而出现“颠簸”（也称抖动）现象。所谓工作集，是指在某段时间间隔（Δ）里，进程实际要访问的页面的集合。工作集的理论是在 1968 年由 Denning 提出来的，他认为，虽然程序只需有少量的几页在主存就可以运行，但为了使程序能够有效地运行，较少地产生缺页，就必须使程序的工作集驻留在主存中。把某进程在时间 *t* 的工作集记为 *w*（*t*，Δ），把变量Δ称为工作集“窗口尺寸（Windows Size）”。正确选择工作集窗口（Δ）的大小，对存储器的有效利用和系统吞吐量的提高，都将产生重要的影响。

程序在运行时对页面的访问是不均匀的，即往往在某段时间内的访问仅局限于较少的若干个页面，如果能够预知程序在某段时间间隔内要访问哪些页面，并能将它们提前调入主存，将会大大地降低缺页率，从而减少置换工作，提高 CPU 的利用率。当每个工作集都已达到最小值时，虚存管理程序跟踪进程的缺页数量，根据主存中自由页面数量可以适当增加其工作集的大小。

3.4 设备管理

设备管理是操作系统中最繁杂而且与硬件紧密相关的部分。设备管理不但要管理实际 I/O 操作的设备（如磁盘机、打印机），还要管理诸如设备控制器、DMA 控制器、中断控制器和 I/O 处理机（通道）等支持设备。设备管理包括各种设备分配、缓冲区管理和实际物理 I/O 设备操作，通过管理达到提高设备利用率和方便用户的目的。

3.4.1 设备管理概述

设备是计算机系统与外界交互的工具，具体负责计算机与外部的输入输出工作，所以常称为外部设备（简称外设）。在计算机系统中，将负责管理和输入输出的机构称为 I/O 系统。因此，I/O 系统由设备、控制器、通道（具有通道的计算机系统）、总线和 I/O 软件组成。

1．设备的分类

现代计算机系统都配有各种各样的设备，如打印机、显示器、绘图仪、扫描仪、键盘和鼠标等。设备可以有各种不同的分类方式。

（1）按数据组织分类。分为块设备（Block Device）和字符设备（Character Device）。块设备是指以数据块为单位来组织和传送数据信息的设备，如磁盘。字符设备是指以单个字符为单位来传送数据信息的设备，如交互式终端、打印机等。

（2）从资源分配角度分类。分为独占设备、共享设备和虚拟设备。独占设备是指在一段时间内只允许一个用户（进程）访问的设备，大多数低速的 I/O 设备，如用户终端、打印机等属于这类设备。共享设备是指在一段时间内允许多个进程同时访问的设备。显然，共享设备必须是可寻址的和可随机访问的设备。典型的共享设备是磁盘。虚拟设备是指通过虚拟技术将一台独占设备变换为若干台供多个用户（进程）共享的逻辑设备。一般可以利用假脱机技术（Spooling 技术）实现虚拟设备。

（3）按数据传输率分类。分为低速设备、中速设备和高速设备。低速设备是指传输速率为每秒钟几个字节到数百个字节的设备，典型的设备有键盘、鼠标、语音的输入等。中速设备是指传输速率在每秒钟数千个字节至数十千个字节的设备，典型的设备有行式打印机、激光打印

机等。高速设备是指传输速率在数百千个字节至数兆字节的设备，典型的设备有磁带机、磁盘机和光盘机等。

2．设备管理的目标与任务

设备管理的目标主要是如何提高设备的利用率，为用户提供方便统一的界面。提高设备的利用率，就是提高 CPU 与 I/O 设备之间的并行操作程度，主要利用的技术有中断技术、DMA 技术、通道技术和缓冲技术。

设备管理的任务是保证在多道程序环境下，当多个进程竞争使用设备时，按一定策略分配和管理各种设备，控制设备的各种操作，完成 I/O 设备与主存之间的数据交换。

设备管理的主要功能是动态地掌握并记录设备的状态、设备分配和释放、缓冲区管理、实现物理 I/O 设备的操作、提供设备使用的用户接口及设备的访问和控制。

3.4.2 I/O 软件

设备管理软件的设计水平决定了设备管理的效率。从事 I/O 设备管理软件的结构，其基本思想是分层构造，也就是说把设备管理软件组织成为一系列的层次。其中低层与硬件相关，它把硬件与较高层次的软件隔离开来。而最高层的软件则向应用提供一个友好的、清晰而统一的接口。

设计 I/O 软件的主要目标是设备独立性和统一命名。I/O 软件独立于设备，就可以提高设备管理软件的设计效率。当输入输出设备更新时，没有必要重新编写全部设备驱动程序。在实际应用中也可以看到，在常用操作系统中，只要安装了相对应的设备驱动程序，就可以很方便地安装好新的输入输出设备，甚至不必重新编译就能将设备管理程序移到他处执行。

I/O 设备管理软件一般分为 4 层：中断处理程序、设备驱动程序、与设备无关的系统软件和用户级软件。至于一些具体分层时细节上的处理，是依赖于系统的，没有严格的划分，只要有利于设备独立这一目标，可以为了提高效率而设计不同的层次结构。

I/O 软件的所有层次及每一层的主要功能如图 3-26 所示。

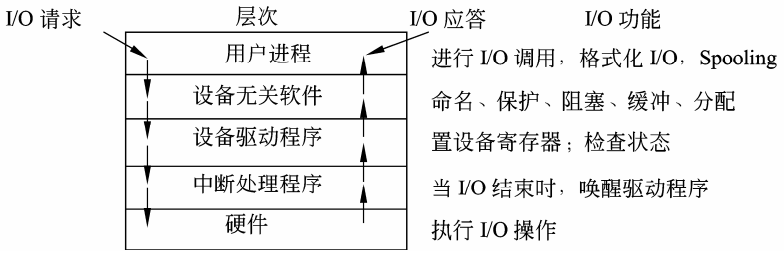


图 3-26　I/O 系统的层次结构与每层的主要功能

图中的箭头给出了 I/O 部分的控制流。这里举一个读硬盘文件的例子。当用户程序试图读一个硬盘文件时，需要通过操作系统实现这一操作。与设备无关软件检查高速缓存中有无要读的数据块。若没有，则调用设备驱动程序，向 I/O 硬件发出一个请求。然后，用户进程阻塞并等待磁盘操作的完成。当磁盘操作完成时，硬件产生一个中断，转入中断处理程序。中断处理程序检查中断的原因，认识到这时磁盘读取操作已经完成，于是唤醒用户进程取回从磁盘读取的信息，从而结束此次 I/O 请求。用户进程在得到了所需的硬盘文件内容之后，继续运行。

3.4.3 设备管理采用的相关缓冲技术

1．通道技术

引入通道的目的是使数据的传输独立于 CPU，使 CPU 从繁琐的 I/O 工作中解脱出来。设置通道后，CPU 只需向通道发出 I/O 命令，通道收到命令后，从主存中取出本次 I/O 要执行的通道程序并执行，仅当通道完成了 I/O 任务后，才向 CPU 发出中断信号。

根据信息交换方式的不同，将通道分为字节多路通道、数组选择通道和数组多路通道三类。由于通道价格昂贵，导致计算机系统中的通道数是有限的，这往往会成为输入输出的“瓶颈”问题。一个单通路的 I/O 系统中主存和设备之间只有一条通路。一旦某通道被设备占用，即使另一通道空闲，连接该通道的其他设备也只有等待。解决“瓶颈”问题的最有效方法是增加设备到主机之间的通路，使得主存和设备之间有两条以上的通路。

2．DMA 技术

直接主存存取（Direct Memory Access，DMA）是指数据在主存与 I/O 设备间的直接成块传送，即在主存与 I/O 设备间传送一个数据块的过程中，不需要 CPU 的任何干涉，只需要 CPU 在过程开始启动（即向设备发出“传送一块数据”的命令）与过程结束（CPU 通过轮询或中断得知过程是否结束和下次操作是否准备就绪）时的处理，实际操作由 DMA 硬件直接执行完成，CPU 在此传送过程中可做别的事情。例如，在非 DMA 时，打印 2048 字节至少需要执行 2048 次输出指令，加上 2048 次中断处理的代价。而在 DMA 情况下，若一次 DMA 可传送 512 个字节，则只需要执行 4 次输出指令和处理 4 次打印机中断。若一次 DMA 可传送字节数大于等于 2048 个字节，则只需要执行一次输出指令和处理一次打印机中断。

3．缓冲技术

缓冲技术可提高外设利用率，尽可能使外设处于忙状态。缓冲技术可以采用硬件缓冲和软件缓冲。硬件缓冲是利用专门的硬件寄存器作为缓冲，软件缓冲是通过操作系统来管理的。引入缓冲的主要原因有以下几个方面。

（1）缓和 CPU 与 I/O 设备间速度不匹配的矛盾。

（2）减少对 CPU 的中断频率，放宽对中断响应时间的限制。

（3）提高 CPU 和 I/O 设备之间的并行性。

在所有的 I/O 设备与处理机（主存）之间都使用了缓冲区来交换数据，所以操作系统必须组织和管理好这些缓冲区。缓冲可分为单缓冲、双缓冲、多缓冲和环形缓冲。

4．Spooling 技术

Spooling（Simultaneous Peripheral Operations On Line，外围设备联机操作），常简称为 Spooling 系统或假脱机系统。所谓 Spooling 技术，实际上是用一类物理设备模拟另一类物理设备的技术，是使独占使用的设备变成多台虚拟设备的一种技术，也是一种速度匹配技术。

Spooling 系统是由“预输入程序”、“缓输出程序”和“井管理程序”以及输入和输出井组成的。其中，输入井和输出井是为了存放从输入设备输入的信息以及作业执行的结果，系统在辅助存储器上开辟的存储区域。Spooling 系统的组成和结构如图 3-27 所示。

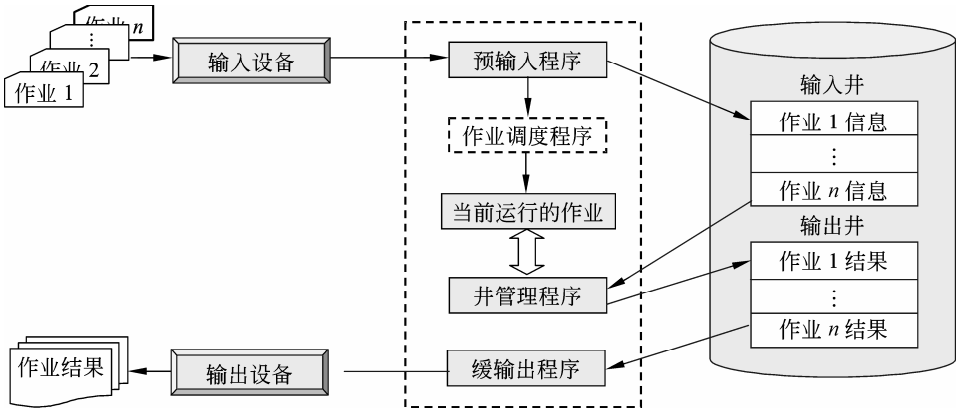


图 3-27　Spooling 系统的组成和结构

Spooling 系统的工作过程是操作系统初启后激活 Spooling 预输入程序使它处于捕获输入请求的状态，一旦有输入请求消息，Spooling 输入程序立即得到执行，把装在输入设备上的作业输入到硬盘的输入井中，并填写好作业表以便在作业执行中要求输入信息时，可以随时找到它们的存放位置。当作业需要输出数据时，可以先将数据送到输出井，当输出设备空闲时，由 Spooling 输出程序把硬盘上输出井的数据送到慢速的输出设备上。

Spooling 系统中拥有一张作业表用来登记进入系统的所有作业的作业名、状态和预输入表位置等信息。每个用户作业拥有一张预输入表来登记该作业的各个文件的情况，包括设备类、

信息长度及存放位置等。输入井中的作业有如下 4 种状态。

（1）提交状态。作业的信息正从输入设备上预输入。

（2）后备状态。作业预输入结束但未被选中执行。

（3）执行状态。作业已被选中运行，运行过程中，它可从输入井中读取数据信息，也可向输出井写信息。

（4）完成状态。作业已经撤离，该作业的执行结果等待缓输出。

3.4.4 磁盘调度

磁盘是可被多个进程共享的设备。当有多个进程请求访问磁盘时，为了保证信息的安全，系统每一时刻只允许一个进程启动磁盘进行 I/O 操作，其余的进程只能等待。因此，操作系统应采用一种适当的调度算法，使各进程对磁盘的平均访问（主要是寻道）时间最小。磁盘调度分为移臂调度和旋转调度两类，并且是先进行移臂调度，然后进行旋转调度。由于访问磁盘最耗时的是寻道时间，因此，磁盘调度的目标是使磁盘的平均寻道时间最少。

1．磁盘驱动调度

常用的磁盘调度算法如下。

（1）先来先服务（First-Come First-Served，FCFS）。这是最简单的磁盘调度算法。它根据进程请求访问磁盘的先后次序进行调度。此算法的优点是公平、简单，且每个进程的请求都能依次得到处理，不会出现某进程的请求长期得不到满足的情况。但此算法由于未对寻道进行优化，致使平均寻道时间可能较长。

（2）最短寻道时间优先（Shortest Seek Time First，SSTF）。该算法选择这样的进程。其要求访问的磁道与当前磁头所在的磁道距离最近，使得每次的寻道时间最短。但这种调度算法却不能保证平均寻道时间最短。

（3）扫描算法（SCAN）。扫描算法不仅考虑到欲访问的磁道与当前磁道的距离，更优先考虑的是磁头的当前移动方向。例如，当磁头正在由里向外移动时，SCAN 算法所选择的下一个访问对象应是其欲访问的磁道既在当前磁道之外，又是距离最近的。这样由里向外地访问，直至再无更外的磁道需要访问时，才将磁臂换向，由外向里移动。这时，同样也是每次选择在当前磁道之内，且距离最近的进程来调度。这样，磁头逐步地向里移动，直至再无更里面的磁道需要访问。显然，这种方式避免了饥饿现象的出现。这种算法中，磁头移动的规律颇似电梯的运行，故又常称为电梯调度算法。

（4）单向扫描调度算法（CSCAN）。SCAN 存在这样的问题：当磁头刚从里向外移动过某一磁道时，恰有一进程请求访问此磁道，这时该进程必须等待，待磁头从里向外，然后再从外向里扫描完所有要访问的磁道后，才处理该进程的请求，致使该进程的请求被严重地推迟。为

了减少这种延迟，算法规定磁头作单向移动。

2．旋转调度算法

当移动臂定位后，有多个进程等待访问该柱面时，应当如何决定这些进程的访问顺序？这就是旋转调度要考虑的问题。显然，系统应该选择延迟时间最短的进程对磁盘的扇区进行访问。当有若干等待进程请求访问磁盘上的信息时，旋转调度应考虑如下情况。

（1）进程请求访问的是同一磁道上不同编号的扇区。

（2）进程请求访问的是不同磁道上不同编号的扇区。

（3）进程请求访问的是不同磁道上具有相同编号的扇区。

对于（1）与（2），旋转调度总是让首先到达读写磁头位置下的扇区先进行传送操作；对于（3），旋转调度可以任选一个读写磁头位置下的扇区进行传送操作。

【例 3.14】 当进程请求读磁盘时，操作系统__（1）__。假设磁盘每磁道有 10 个扇区，移动臂位于 18 号柱面上，且进程的请求序列如表 3-3 所示。

表 3-3　进程的请求序列

请 求 序 列	柱　面　号	磁　头　号	扇　区　号
①	15	8	9
②	20	6	3
③	20	9	6
④	40	10	5
⑤	15	8	4
⑥	6	3	10
⑦	8	7	9
⑧	15	10	4

那么，按照最短寻道时间优先的响应序列为__（2）__。

（1）A. 只需要进行旋转调度，无须进行移臂调度

B. 旋转、移臂调度同时进行

C. 先进行移臂调度，再进行旋转调度

D. 先进行旋转调度，再进行移臂调度

（2）A. ②③⑤①⑧⑦⑥④　　B. ②③⑤⑧①⑦⑥④

C. ⑤⑧①⑦⑥②④③　　D. ⑥⑦⑧①⑤②③④

分析：空（1）的正确答案为 C；空（2）的正确答案为 A。

当进程请求读磁盘时，操作系统先进行移臂调度，再进行旋转调度。由于移动臂位于 18 号柱面上，按照最短寻道时间优先的响应柱面序列为 20→15→8→6→40。按照旋转调度的原则：

进程在 20 号柱面上的响应序列为②→③，因为进程访问的是不同磁道上的不同编号的扇区，旋转调度总是让首先到达读写磁头位置下的扇区先进行传送操作。进程在 15 号柱面上的响应序列为⑤→①→⑧或⑤→⑧→①。对于①和⑧可以任选一个进行读写，因为进程访问的是不同磁道上具有相同编号的扇区，旋转调度可以任选一个读写磁头位置下的扇区进行传送操作。④在 40 号柱面上。⑥在 6 号柱面上。⑦在 8 号柱面上。

【例 3.15】 数据存储在磁盘上的排列方式会影响 I/O 服务的总时间。假设每磁道划分成 10 个物理块，每块存放 1 个逻辑记录。逻辑记录 R1，R2，…，R10 存放在同一个磁道上，记录的安排顺序如下表所示。

物理块	1	2	3	4	5	6	7	8	9	10
逻辑记录	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10

假定磁盘的旋转速度为每周 20ms，磁头当前处在 R1 的开始处。若系统顺序处理这些记录，使用单缓冲区，每个记录处理时间为 4ms，则处理这 10 个记录的最长时间为__（1）__；对信息存储进行优化分布后，处理 10 个记录的最少时间为__（2）__。

（1）A. 180ms　　B. 200ms　　C. 204ms　　D. 220ms

（2）A. 40ms　　B. 60ms　　C. 100ms　　D. 160ms

空（1）分析：系统读记录的时间为 20/10＝2ms，对第一种情况，系统读出并处理记录 R1 之后，将转到记录 R4 的开始处，所以为了读出记录 R2，磁盘必须再转一圈，需要 2ms（读记录）加 20ms（转一圈）的时间。这样，处理 10 个记录的总时间应为处理前 9 个记录（即 R1，R2，…，R9）的总时间再加上读 R10 和处理时间，即 9×22ms+ 6ms=204ms。

空（2）分析：对于第二种情况，若对信息进行分布优化的结果如下表所示。

物理块	1	2	3	4	5	6	7	8	9	10
逻辑记录	R1	R8	R5	R2	R9	R6	R3	R10	R7	R4

可以看出，当读出记录 R1 并处理结束后，磁头刚好转至 R2 记录的开始处，立即就可以读出并处理，因此处理 10 个记录的总时间为 10×（2ms（读记录）+4ms（处理记录））＝10×6ms=60ms。

3.5　文件管理

随着计算机应用需求不断增长，快速、高效地处理大量的信息是计算机的首要任务之一，而这些信息通常是存储在大容量的外存储器上。然而在早期，用户要访问外存储器上的信息是很麻烦的，不仅要考虑信息在外存储器上的存放位置，而且要记住信息在外存储器的分布情况，构造 I/O 程序。稍不注意，就会破坏已存放的信息。特别是多道程序技术出现后，多个用户之

间根本无法预料各个不同程序间的信息在外存储器上是如何分配的。鉴于这些原因，引入文件系统专门负责管理外存储器上的信息，使用户可以“按名”高效、快速和方便地存储信息。

3.5.1 文件与文件系统

1．文件

文件（file）是具有符号名的、在逻辑上具有完整意义的一组相关信息项的集合。例如，一个源程序、一个目标程序、编译程序、一批待加工的数据和各种文档等都可以各自组成一个文件。

信息项是构成文件内容的基本单位，可以是一个字符，也可以是一个记录，记录可以等长，也可以不等长。一个文件包括文件体和文件说明。文件体是文件真实的内容；文件说明是操作系统为了管理文件所用到的信息，包括文件名、文件内部标识、文件的类型、文件存储地址、文件的长度、访问权限、建立时间和访问时间等。

文件是一种抽象机制，它隐蔽了硬件和实现细节，提供了将信息保存在磁盘上而且便于以后读取的手段，使用户不必了解信息存储的方法、位置以及存储设备实际运作方式便可存取信息。因此，在文件管理中的一个非常关键的问题在于文件的命名。文件名是在进程创建文件时确定的，以后这个文件将独立于进程存在直到它被显式删除。其他进程要使用文件时必须显式指出该文件名，操作系统根据文件名对其进行控制和管理。不同的操作系统文件命名规则有所不同，即文件名字的格式和长度因系统而异。

2．文件系统

由于计算机系统处理的信息量越来越大，所以不可能将所有的信息保存到主存中。特别是在多用户系统中，既要保证各用户文件存放的位置不冲突，又要防止任一用户对外存储器（简称外存）空间占而不用；既要保证各用户文件在未经许可的情况下不被窃取和破坏，又要允许在特定的条件下多个用户共享某些文件。因此，需要设立一个公共的信息管理机制来负责统一管理外存和外存上的文件。

所谓文件管理系统，就是操作系统中实现文件统一管理的一组软件和相关数据的集合，专门负责管理和存取文件信息的软件机构，简称文件系统。文件系统的功能包括按名存取，即用户可以“按名存取”，而不是“按地址存取”；统一的用户接口，在不同设备上提供同样的接口，方便用户操作和编程；并发访问和控制，在多道程序系统中支持对文件的并发访问和控制；安全性控制，在多用户系统中的不同用户对同一文件可有不同的访问权限；优化性能，采用相关技术提高系统对文件的存储效率、检索和读写性能；差错恢复，能够验证文件的正确性，并具有一定的差错恢复能力。

3．文件的类型

（1）按文件性质和用途可将文件分为系统文件、库文件和用户文件。

（2）按信息保存期限分类可将文件分为临时文件、档案文件和永久文件。

（3）按文件的保护方式分类可将文件分为只读文件、读写文件、可执行文件和不保护文件。

（4）UNIX 系统将文件分为普通文件、目录文件和设备文件（特殊文件）。

（5）目前常用的文件系统类型有 FAT、Vfat、NTFS、Ext2 和 HPFS 等。

文件分类的目的是对不同文件进行管理，提高系统效率；提高用户界面友好性。当然，根据文件的存取方法和物理结构不同还可以将文件分为不同的类型，这在文件的逻辑结构和文件的物理结构中介绍。

3.5.2 文件的结构和组织

文件的结构是指文件的组织形式。从用户角度看到的文件组织形式称为文件的逻辑结构，文件系统的用户只要知道所需文件的文件名，就可存取文件中的信息，而无须知道这些文件究竟存放在什么地方。从实现的角度看文件在文件存储器上的存放方式，称为文件的物理结构。

1．文件的逻辑结构

文件的逻辑结构可分为两大类：一是有结构的记录式文件，它是由一个以上的记录构成的文件，故又称为记录式文件；二是无结构的流式文件，它是由一串顺序字符流构成的文件。

1）有结构的记录式文件

在记录式文件中，所有的记录通常都是描述一个实体集的，有着相同或不同数目的数据项，记录的长度可分为定长和不定长两类。

（1）定长记录。指文件中所有记录的长度相同。所有记录中的各个数据项都处在记录中相同的位置，具有相同的顺序及相同的长度，文件的长度用记录数目表示。定长记录的特点是处理方便，开销小，是目前较常用的一种记录格式，被广泛用于数据处理中。

（2）变长记录。指文件中各记录的长度不相同。这是因为：一个记录中所包含的数据项数目可能不同，如书的著作者、论文中的关键词；数据项本身的长度不定，如病历记录中的病因、病史，科技情报记录中的摘要等。但是，不论是哪一种结构，在处理前每个记录的长度是可知的。

2）无结构的流式文件

文件体为字节流，不划分记录。无结构的流式文件通常采用顺序访问方式，并且每次读写访问可以指定任意数据长度，其长度以字节为单位。对流式文件访问，是利用读写指针指出下一个要访问的字符。可以把流式文件看作是记录式文件的一个特例。在 UNIX 系统中，所有的文件都被看作是流式文件，即使是有结构的文件，也被视为流式文件，系统不对文件进行格式处理。

2．文件的物理结构

文件的物理结构是指文件的内部组织形式，即文件在物理存储设备上的存放方法。由于文件的物理结构决定了文件在存储设备上的存放位置，所以文件的逻辑块号到物理块号的转换也是由文件的物理结构决定的。根据用户和系统管理上的需要，可采用多种方法来组织文件，下面介绍几种常见的文件物理结构。

（1）连续结构。连续结构也称顺序结构，它将逻辑上连续的文件信息（如记录）依次连续存放在连续编号的物理块上。只要知道文件的起始物理块号和文件的长度，就可以很方便地进行文件的存取。

对文件诸记录进行批量存取时，连续结构在所有逻辑文件中的存取效率是最高的。但在交互应用的场合，如果用户（程序）要求随机地查找或修改单个记录，此时系统需要逐个地查找各个记录，这样采用连续结构所表现出来的性能就可能很差，尤其是当文件较大时情况更为严重。连续结构的另一个缺点是不便于记录的增加或删除操作。为了解决这个问题，可以为采用连续结构的文件配置一个运行记录文件（Log File）或称为事务文件（Transactor File），规定每隔一定时间，例如 4 小时，将运行记录文件与原来的主文件进行合并，产生一个新文件。这样，不必每次对记录进行增加或删除操作，物理移动磁盘信息，使其成为连续结构。

（2）链接结构。链接结构也称串联结构，它是将逻辑上连续的文件信息（如记录）存放在不连续的物理块上，每个物理块设有一个指针指向下一个物理块。因此，只要知道文件的第一个物理块号，就可以按链指针查找整个文件。

（3）索引结构。采用索引结构时，将逻辑上连续的文件信息（如记录）存放在不连续的物理块中，系统为每个文件建立一张索引表。索引表记录了文件信息所在的逻辑块号对应的物理块号，并将索引表的起始地址放在与文件对应的文件目录项中。

（4）多个物理块的索引表。索引表是在文件建立时由系统自动建立的，并与文件一起存放在同一文件卷上。根据一个文件大小的不同，其索引表占用物理块的个数不等，一般占一个或几个物理块。多个物理块的索引表可有两种组织方式：链接文件和多重索引方式。

在 UNIX 文件系统中采用的是三级索引结构，文件系统中 inode 是基本的构件，它表示文件系统树型结构的节点。UNIX 文件索引表项分 4 种寻址方式：直接寻址、一级间接寻址、二级间接寻址和三级间接寻址。

3.5.3 文件目录

为了实现“按名存取”，系统必须为每个文件设置用于描述和控制文件的数据结构，它至少要包括文件名和存放文件的物理地址，这个数据结构称为文件控制块（FCB），文件控制块的有序集合称为文件目录。换句话说，文件目录是由文件控制块组成的，专门用于文件的检索。

文件控制块也称为文件的说明或文件目录项（简称目录项）。

1．文件控制块

文件控制块中包含以下三类信息：基本信息类、存取控制信息类和使用信息类。

（1）基本信息类。例如文件名、文件的物理地址、文件长度和文件块数等。

（2）存取控制信息类。文件的存取权限，像 UNIX 用户分成文件主、同组用户和一般用户三类，这三类用户的读写执行（RWX）权限。

（3）使用信息类。文件建立日期、最后一次修改日期、最后一次访问的日期；当前使用的信息如打开文件的进程数，在文件上的等待队列等。

2．目录结构

文件目录结构的组织方式直接影响到文件的存取速度，关系到文件共享性和安全性，因此组织好文件的目录是设计文件系统的重要环节。常见的目录结构有三种：一级目录结构、二级目录结构和多级目录结构。

（1）一级目录结构。一级目录的整个目录组织是一个线性结构，在整个系统中只需建立一张目录表，系统为每个文件分配一个目录项。一级目录结构简单，但缺点是查找速度慢，不允许重名和不便于实现文件共享等，因此它主要用在单用户环境中。

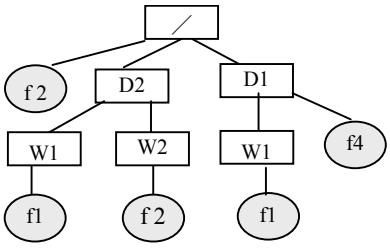
（2）二级目录结构。为了克服一级目录结构存在的缺点，引入了二级目录结构。二级目录结构是由主文件目录（Master File Directory，MFD）和用户目录（User File Directory，UFD）组成的。在主文件目录中，每个用户文件目录都占有一个目录项，其目录项中包括用户名和指向该用户目录文件的指针。用户目录是由用户所有文件的目录项组成。

二级目录结构基本上克服了单级目录的缺点，其优点是提高了检索目录的速度，较好地解决了重名问题。采用二级目录结构也存在一些问题。该结构虽然能有效地将多个用户隔离开，这种隔离在各个用户之间完全无关时是一个优点；但当多个用户之间要相互合作去共同完成一个大任务，且一个用户又需去访问其他用户的文件时，这种隔离便成为一个缺点，因为这种隔离使诸用户之间不便于共享文件。

（3）多级目录结构。为了解决以上问题，在多道程序设计系统中常采用多级目录结构，这种目录结构像一棵倒置的有根树，所以也称为树型目录结构。从树根向下，每一个节点是一个目录，叶节点是文件。MS-DOS 和 UNIX 等操作系统均采用多级目录结构。

采用多级目录结构的文件系统中，用户要访问一个文件，必须指出文件所在的路径名，路径名是从根目录开始到该文件的通路上所有各级目录名拼起来得到的。各目录名之间，目录名与文件名之间需要用分隔符隔开。例如，在 MS-DOS 中分隔符为“ \ ”，在 UNIX 中分隔符为“ / ”。绝对路径名（absolute path name）是指从根目录 “ ／ ” 开始的完整文件名，即它是由从根目录开始的所有目录名以及文件名构成的。

【例 3.16】 在下图所示的树型文件系统中，方框表示目录，圆圈表示文件，“/” 表示路径中的分隔符，“/”在路径之首时表示根目录。



假设当前目录是 D2，进程 A 以如下两种方式打开文件 f2。

方式①　fd1=open("__（1）__/f2",O_RDONLY);

方式②　fd1=open("/D2/W2/f2", O_RDONLY);

其中，方式①的工作效率比方式②的工作效率高，因为采用方式①，文件系统是从__（2）__。

（1）A. /D2/W2　　B. D2/W2　　C. W2　　D. /W2

（2）A. 根目录开始查找文件 f2，系统查找时间少，读取 f2 文件次数不变

B. 当前路径开始查找文件 f2，系统查找时间少，读取 f2 文件次数少

C. 根目录开始查找文件 f2，系统查找时间少，读取 f2 文件次数少

D. 当前路径开始查找文件 f2，系统查找时间少，读取 f2 文件次数不变

分析：空（1）的正确答案是 C。因为在树型目录结构中，树的根节点为根目录，数据文件作为树叶，其他所有目录均作为树的节点。在树型目录结构中，从根目录到任何数据文件之间只有唯一的一条通路。从树根开始，把全部目录文件名与数据文件名依次用“/”连接起来，构成该数据文件的路径名，且每个数据文件的路径名是唯一的。这样，可以解决文件重名问题。所以，虽然数据文件名均为 f2，但不一定是相同的文件。从树根开始的路径名为绝对路径名，如果文件系统有很多级时，使用绝对路径即不方便又费时间，所以引入相对路径名，即从当前目录开始，再逐级通过中间的目录文件，最后到达所要访问的数据文件。根据题意，方式①的工作效率比方式②的工作效率高，正确答案为 C。同样，从当前目录开始，采用相对路径名较之采用绝对路径名可以减少系统访问目录文件的次数，但是访问文件 f2 的次数是不变的，所以对于空（2）的正确答案为 D。

3.5.4 存取方法和存储空间的管理

1．文件的存取方法

文件的存取方法是指读写文件存储器上的一个物理块的方法，通常有顺序存取和随机存取两种方法。顺序存取是指对文件中的信息按顺序依次读写的方式；随机存取是指对文件中的信

息可以按任意的次序随机地读写文件中的信息。

（1）顺序存取法。在提供记录式文件结构的系统中，顺序存取法就是严格按物理记录排列的顺序依次读取。如果当前读取的是 R_i 记录，下一次要读取的记录自动地确定为 R_{i+1}。在只提供无结构的流式文件中，顺序存取法是按读写的位移（offset）从当前位置开始读写，每读完一段信息，读写位移自动加上这段信息的长度，以便读下一段信息。下面着重讨论记录式文件的存取法。

（2）直接存取法。直接存取法允许用户随意存取文件中任意一个物理记录。对于无结构的流式文件，采用直接存取法，必须事先将读写偏移移动到待读写信息的位置上，然后再进行读写。

（3）按键存取法。按键存取法是直接存取法的一种，它不是根据记录的编号或地址来存取文件中的记录，而是根据文件中各记录的某个数据项内容来存取记录的，这种数据项称之为“键”。因此，将这种存取法称之为按键存取法。

2．文件存储空间的管理

外存具有大容量的存储空间，被多用户共享，用户执行程序经常要在磁盘上存储文件和删除文件，因此，文件系统必须对磁盘空间进行管理。外存空闲空间管理的数据结构通常称为磁盘分配表（disk allocation table）。常用的空闲空间的管理方法有位示图、空闲区表和空闲块链三种。

（1）空闲区表。将外存空间上一个连续未分配区域称为“空闲区”。操作系统为磁盘外存上所有空闲区建立一张空闲表，每个表项对应一个空闲区，空闲表中包含序号、空闲区的第一块号、空闲块的块数和状态等信息，如表 3-4 所示。它适用于连续文件结构。

表 3-4　空闲区表

序　　号	第一个空闲块号	空 闲 块 数	状　　态
1	18	5	可用
2	29	8	可用
3	105	19	可用
4	—	—	未用

（2）位示图。这种方法是在外存上建立一张位示图（bitmap），记录文件存储器的使用情况。每一位对应文件存储器上的一个物理块，取值 0 和 1 分别表示空闲和占用。文件存储器上的物理块依次编号为 0，1，2，…。假如系统中字长为 32 位，那么在位示图中的第一个字对应文件存储器上的 0，1，2，…，31 号物理块；第二个字对应文件存储器上的 32，33，34，…，63 号物理块，依此类推。

这种方法的主要特点是位示图的大小由磁盘空间的大小（物理块总数）决定，位示图的描述能力强，适合各种物理结构。

（3）空闲块链。每个空闲物理块中有指向下一个空闲物理块的指针，所有空闲物理块构成一个链表，链表的头指针放在文件存储器的特定位置上（如管理块中）。不需要磁盘分配表，节省空间。每次申请空闲物理块只需根据链表的头指针取出第一个空闲物理块，根据第一个空闲物理块的指针可找到第二个空闲物理块，依此类推即可。

（4）成组链接法。在 UNIX 系统中，将空闲块分成若干组，每 100 个空闲块为一组，每组的第一个空闲块登记了下一组空闲块的物理盘块号和空闲块总数。假如一个组的第一个空闲块号等于 0 的话，意味着该组是最后一组，无下一组空闲块。

3.5.5 文件的使用

文件系统将用户的逻辑文件按一定的组织方式转换成物理文件存放到文件存储器上，也就是说，文件系统为每个文件与该文件在磁盘上的存放位置建立了对应关系。当用户使用文件时，文件系统通过用户给出的文件名，查出对应文件的存放位置，读出文件的内容。在多用户环境下，为了文件安全和保护起见，操作系统为每个文件建立和维护关于文件主、访问权限等方面的信息。为此，操作系统在操作级（命令级）和编程级（系统调用和函数）向用户提供文件的服务。

操作系统在操作级向用户提供的命令有目录管理类命令、文件操作类命令（如复制、删除和修改）和文件管理类命令（如设置文件权限）等。

3.5.6 文件的共享和保护

1．文件的共享

文件共享是指不同用户进程使用同一文件，它不仅是不同用户完成同一任务所必须的功能，而且还可以节省大量的主存空间，减少由于文件复制而增加的访问外存的次数。文件共享有多种形式，采用文件名和文件说明分离的目录结构有利于实现文件共享。

常见的文件链接有硬链接和符号链接两种。

1）硬链接

文件的硬链接是指两个文件目录表目指向同一个索引节点的链接，该链接也称基于索引节点的链接。换句话说，硬链接是指不同文件名与同一个文件实体的链接。文件硬链接不利于文件主删除它拥有的文件，因为文件主要删除它拥有的共享文件，必须首先删除（关闭）所有的硬链接，否则就会造成共享该文件的用户的目录表目指针悬空。

例如，在 UNIX 系统中的 ln 命令，可以将多个文件名与一个文件体建立链接，其格式为：

in 文件名 新文件名

或

in 文件名 目录名

ls 命令放在/bin 子目录下，可在/usr/bin 子目录下设置一个 DOS 兼容的命令 dir，执行该命令相当于执行 ls 命令。使用命令 ln 可给一个已存在文件增加一个新文件名，即文件链接数增加 1，此种链接是不能跨越文件系统的。为了共享文件，只是在两个不同子目录下取了不同的文件名 ls 和 dir，但它们具有相同的索引节点。UNIX 这种文件的结构称为树型带勾链的目录结构。在文件的索引节点中，di_nlink 变量表示链接到该索引节点上的链接数；在用命令 ls-l 长列表显示时，文件的第 2 项数据项表示链接数。

2）符号链接

符号链接在建立的新文件或目录并与原来文件或目录的路径名进行映射，当访问一个符号链接时，系统通过该映射找到原文件的路径，并对其进行访问。

例如，在 UNIX 系统中的 ln -s 命令建立符号链接。此时，系统为共享的用户创建一个 link 类型的新文件，将这新文件登录在该用户共享目录项中，这个 link 型文件包含链接文件的路径名。该类文件在用 ls 命令长列表显示时，文件链接数为 l。

采用符号连接可以跨越文件系统，甚至可以通过计算机网络连接到世界上任何地方的机器中的文件，此时只需提供该文件所在的地址，以及在该机器中的文件路径。

符号链接的缺点是其他用户读取符号链接的共享文件比读取硬链接的共享文件需要增加读盘操作的次数。因为其他用户去读符号链接的共享文件时，系统中根据给定的文件路径名，逐个分量地去查找目录，通过多次读盘操作才能找到该文件的索引节点，而用硬链接的共享文件的目录文件表目中已包括了共享文件的索引节点号。

2．文件的保护

文件系统对文件的保护常采用存取控制方式进行。所谓存取控制，就是不同的用户对文件的访问规定不同的权限，以防止文件被未经文件主同意的用户访问。

（1）存取控制矩阵。理论上，存取控制方法可用存取控制矩阵，它是一个二维矩阵，一维列出计算机的全部用户，另一维列出系统中的全部文件，矩阵中每个元素 A_{ij} 是表示第 i 个用户对第 j 个文件的存取权限。通常存取权限有可读、可写、可执行以及它们的组合，如表 3-5 所示。

表 3-5　存取控制矩阵

文件 / 用户	ALPHA	BETA	REPORT	SQRT		
张军	RWX	—	R-X	—		
李晓钢	R-X	—	RWX	R-X	—	

续表

文件 / 用户	ALPHA	BETA	REPORT	SQRT		
王伟	—	RWX	R-X	R-X		
赵凌	—	—	—	RWX		
.	.					

存取控制矩阵在概念上是简单清楚的，但实现上却有困难。当一个系统用户数和文件数很大时，二维矩阵要占很大的存储空间，验证过程也将耗费许多系统时间。

（2）存取控制表。存取控制矩阵由于太大而往往无法实现。一个改进的办法是按用户对文件的访问权力的差别对用户进行分类，由于某一文件往往只与少数几个用户有关，所以这种分类方法可使存取控制表大为简化。UNIX 系统就是使用这种存取控制表方法。它把用户分成三类：文件主、同组用户和其他用户，每类用户的存取权限为可读、可写、可执行以及它们的组合。在用 ls 长列表显示时，每组存取权限用三个字母 RWX 表示，如读、写和执行中哪一样存取不允许，则用“ - ”字符表示。用 ls -l 长列表显示 ls 文件如下：

```
-r-xr-xr-t 1 bin    bin    43296    May 13    1997    /opt/K/SCO/Unix/5.0.4Eb/bin/ls
```

显示前 2～10 共 9 个字符表示文件的存取权限，每 3 个字符为一组，分别表示文件主、同组用户和其他用户的存取权限。由于存取控制表对每个文件按用户分类，所以该存取控制表可存放在每个文件的文件控制块中，对 UNIX 只需 9 位二进制来表示三类用户对文件的存取权限，该权限存在文件索引节点的 di_mode 中。

（3）用户权限表。改进存取控制矩阵的另一种方法是以用户或用户组为单位将用户可存取的文件集中起来存入表中，这称为用户权限表。表中每个表目表示该用户对应文件的存取权限，这相当于存取控制矩阵一行的简化。

（4）密码。在创建文件时，由用户提供一个密码，在文件存入磁盘时用该密码对文件内容加密。进行读取操作时，要对文件进行解密，只有知道密码的用户才能读取文件。

3.5.7 系统的安全与可靠性

1．系统的安全

系统的安全涉及两类不同的问题，一类涉及到技术、管理、法律、道德和政治等问题，另一类涉及操作系统的安全机制。随着计算机应用范围扩大，在所有稍具规模的系统中，都从多个级别上来保证系统的安全性。一般从 4 个级别上对文件进行安全性管理：系统级、用户级、目录级和文件级。

系统级安全管理的主要任务是不允许未经授权的用户进入系统，从而也防止了他人非法使用系统中各类资源（包括文件）。系统级管理的主要措施有注册与登录。

用户级安全管理是通过对所有用户分类和对指定用户分配访问权，不同的用户对不同文件设置不同的存取权限来实现。例如，在 UNIX 系统中将用户分为文件主、组用户和其他用户。有的系统将用户分为超级用户、系统操作员和一般用户。

目录级安全管理是为了保护系统中各种目录而设计的，它与用户权限无关。为保证目录的安全，规定只有系统核心才具有写目录的权利。

文件级安全管理是通过系统管理员或文件主对文件属性的设置来控制用户对文件的访问。通常可设置以下几种属性：只执行、隐含、只读、读/写、共享、系统。用户对文件的访问，将由用户访问权、目录访问权限及文件属性三者的权限所确定，或者说是有效权限和文件属性的交集。例如对于只读文件，尽管用户的有效权限是读/写，但都不能对只读文件进行修改、更名和删除。对于一个非共享文件，将禁止在同一时间内由多个用户对它们进行访问。

通过上述 4 级文件保护措施，可有效地对文件进行保护。

2．文件系统的可靠性

文件系统的可靠性是指系统抵抗和预防各种物理性破坏和人为性破坏的能力。比起计算机的损坏，文件系统破坏往往后果更加严重。例如，将开水撒在键盘上引起的故障，尽管伤脑筋但毕竟可以修复；但如果文件系统破坏了，在很多情况下是无法恢复的。特别是对于那些程序文件、客户档案、市场计划或其他数据文件丢失的客户来说，这不亚于一场大的灾难。尽管文件系统无法防止设备和存储介质的物理损坏，但至少应能保护信息。

（1）转储和恢复。文件系统中无论是硬件或软件，都会发生损坏和错误，例如自然界的闪电、电压的突变、火灾和水灾等均可能引起软、硬件的破坏。为了使文件系统万无一失，应当采用相应的措施。最简单和常用的措施是通过转储操作，形成文件或文件系统的多个副本。这样，一旦系统出现故障，利用转储的数据使得系统恢复成为可能。常用的转储方法有静态转储和动态转储、海量转储和增量转储。

（2）日志文件。在计算机系统的工作过程中，操作系统把用户对文件的插入、删除和修改的操作写入日志文件。一旦发生故障，操作系统恢复子系统利用日志文件来进行系统故障恢复，并可协助后备副本进行介质故障恢复。

（3）文件系统的一致性。影响文件系统可靠性的因素之一是文件系统的一致性问题。很多文件系统是先读取磁盘块到主存，在主存进行修改，修改完毕在写回磁盘。但如读取某磁盘块，修改后在将信息写回磁盘前系统崩溃，则文件系统就可能会出现不一致性状态。如果这些未被写回的磁盘块是索引节点块、目录块或空闲块，那么后果是不堪设想的。通常解决方案是采用文件系统的一致性检查，一致性检查包括块的一致性检查和文件的一致性检查。

3.6 作业管理

作业是系统为完成一个用户的计算任务（或一次事务处理）所做的工作总和。例如，对用户编写的源程序，需要经过编译、连接装入以及执行等步骤得到结果，这其中的每一个步骤称为作业步。操作系统中用来控制作业进入、执行和撤销的一组程序称为作业管理程序。操作系统可以进一步为每个作业创建作业步进程，完成用户的工作。

3.6.1 作业与作业控制

1．作业控制

可以采用脱机和联机两种控制方式控制用户作业的运行。在脱机控制方式中，作业运行的过程是无须人工干预的，因此，用户必须将自己想让计算机干什么的意图用作业控制语言（JCL）编写成作业说明书，连同作业一起提交给计算机系统。在联机控制方式中，操作系统向用户提供了一组联机命令，用户可以通过终端输入命令将自己想让计算机干什么的意图告诉计算机，以控制作业的运行过程，因此整个作业的运行过程需要人工干预。

作业由程序、数据和作业说明书三部分组成。作业说明书包括作业基本情况、作业控制、作业资源要求的描述，它体现用户的控制意图。其中，作业基本情况包括用户名、作业名、编程语言和最大处理时间等；作业控制描述包括作业控制方式、作业步的操作顺序、作业执行出错处理；作业资源要求描述包括处理时间、优先级、主存空间、外设类型和数量、实用程序要求等。

2．作业状态及转换

作业状态分为 4 种：提交、后备、执行和完成。

（1）提交。作业提交给计算机中心，通过输入设备送入计算机系统的过程状态称为提交状态。

（2）后备。通过 Spooling 系统将作业输入到计算机系统的后备存储器（磁盘）中，随时等待作业调度程序调度时的状态。

（3）执行。一旦作业被作业调度程序选中，为其分配了必要的资源，并为其建立相应的进程后，该作业便进入了执行状态。

（4）完成。当作业正常结束或异常中止时，作业进入完成状态。此时有作业调度程序对该作业进行善后处理。如撤销作业的作业控制块，收回作业所占的系统资源，将作业的执行结果形成输出文件放到输出井中，由 Spooling 系统控制输出。

作业的状态及转换如图 3-28 所示。

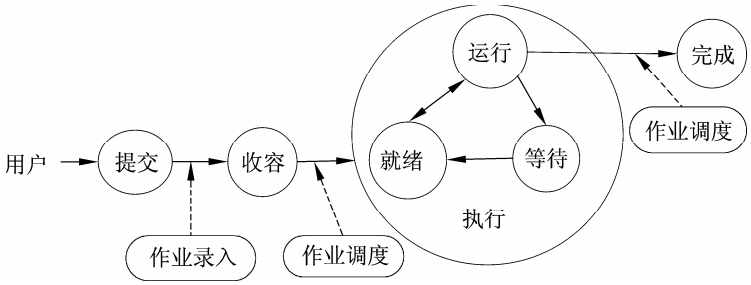


图 3-28　作业的状态及其转换

3．作业控制块和作业后备队列

所谓作业控制块（JCB），是记录与该作业有关的各种信息的登记表。JCB 是作业存在的唯一标志，包括用户名、作业名和状态标志等信息。

由于在输入井中有较多的后备作业，为了便于作业调度程序调度，通常将作业控制块排成一个或多个队列，而这些队列称为作业后备队列。也就是说，作业后备队列是由若干个 JCB 组成的。

3.6.2　作业调度

选择调度算法需要考虑的因素：与系统的整个设计目标一致，均衡使用系统资源，以及平衡系统和用户的要求。对用户来说，作业能“立即执行”往往难以做到，但是应保证进入系统的作业在规定的截止时间内完成，而且系统应设法缩短作业的平均周转时间。

1．作业调度算法

常用的作业调度算法如下。

（1）先来先服务。按作业到达先后进行调度，即启动等待时间最长的作业。

（2）短作业优先。以要求运行时间长短进行调度，即启动要求运行时间最短的作业。

（3）响应比高优先。响应比高的作业优先启动。

定义响应比如下：

$$R_p=\frac{\text{作业响应时间}}{\text{作业执行时间}}$$

其中作业响应时间为作业进入系统后的等候时间与作业的执行时间之和，即 $R_p=1+\frac{\text{作业等待时间}}{\text{作业执行时间}}$。

响应比高者优先算法，在每次调度前都要计算所有被选作业（在作业后备队列中）的响应

比，然后选择响应比最高的作业执行。该算法比较复杂，系统开销大。

（4）优先级调度算法。可由用户指定作业优先级，优先级高的作业先启动。也可由系统根据作业要求的紧迫程度，或者照顾“I/O 繁忙”的作业，以便充分发挥外设的效率等。

（5）均衡调度算法。这种算法的基本思想是根据系统的运行情况和作业本身的特性对作业进行分类。作业调度程序轮流地从这些不同类别的作业中挑选作业执行。这种算法力求均衡地使用系统的各种资源，即注意发挥系统效率，又使用户满意。

2．作业调度算法性能的衡量指标

在一个以批量处理为主的系统中，通常用平均周转时间或平均带权周转时间来衡量调度性能的优劣。假设作业 $J_i(i=1,2,\cdots,n)$ 的提交时间为 t_{si}，执行时间为 t_{ri}，作业完成时间为 t_{oi}，则作业 J_i 的周转时间 T_i 和带权周转时间 W_i 分别定义为：

$T_i = t_{oi} - t_{si} \quad (i=1,2,\cdots,n)$，$W_i = T_i / t_{ri} \quad (i=1,2,\cdots,n)$

n 个作业的平均周转时间 T 和平均带权周转时间 W 分别定义为：

$$T = \frac{1}{n}\sum_{i=1}^{n} T_i \ ,\quad W = \frac{1}{n}\sum_{i=1}^{n} W_i$$

站在用户的角度来说，总是希望自己的作业在提交后能立即执行，这意味着当等待时间为 0 时作业的周转时间最短，即 $T_i = t_{ri}$。但是，作业的执行时间 t_{ri} 并不能直观地衡量出系统的性能，而带权周转时间 W_i 却能直观反应系统的调度性能。站在整个系统的角度来说，不可能满足每个用户的这种要求，而只能是系统的平均周转时间或平均带权周转时间最小。

3.6.3 用户界面

用户界面（user interface）是计算机中实现用户与计算机通信的软件、硬件部分总称。用户界面也称用户接口，或人机界面。

用户界面的硬件部分包括用户向计算机输入数据或命令的输入装置及由计算机输出供用户观察或处理的输出装置。用户界面的软件部分包括用户与计算机相互通信的协议、约定、操纵命令及其处理软件。目前常用的输入输出装置有键盘、鼠标、显示器和打印机等。常用的人机通信方法有命令语言、选项、表格填充及直接操纵等。从计算机用户界面的发展过程来看，可分为如下阶段。

（1）控制面板式用户界面。这是计算机发展早期，用户通过控制台开关、板键或穿孔纸带向计算机送入命令或数据，而计算机通过指示灯及打印机输出运行情况或结果。这种界面的特点是人去适应现在看来十分笨拙的计算机。

（2）字符用户界面。字符用户界面是基于字符型的。用户通过键盘或其他输入设备输入字符，由显示器或打印机输出字符。字符用户界面的优点是功能强、灵活性好、屏幕开销少；缺

点是操作步骤繁琐，学会操作也较费时。

（3）图形用户界面。随着文字、图形、声音和图像等多媒体技术的出现，使各种图形用户界面应运而生，用户既可使用传统的字符，也可使用图形、图像和声音同计算机进行交互，操作将更为自然，更加方便。现代界面的关键技术是超文本。超文本的“超”体现在它不仅包括文本，还包括图像、音频和视频等多媒体信息，即将文本的概念扩充到超文本，超文本的最大特点是具有指向性。

（4）新一代用户界面。虚拟现实技术将用户界面发展推向一个新阶段：人将作为参与者，以自然的方式与计算机生成的虚拟环境进行通信。以用户为中心、自然、高效、高带宽、非精确、无地点限制等是新一代用户界面的特征。多媒体、多通道及智能化是新一代用户界面的技术支持。语音、自然语言、手势、头部跟踪、表情和视线跟踪等新的、更加自然的交互技术，将为用户提供更方便的输入技术。计算机将通过多种感知通道来理解用户的意图，实现用户的要求。计算机不仅以二维屏幕向用户输出，而且以真实感（立体视觉、听觉、嗅觉和触觉等）的计算机仿真环境向用户提供真实的体验。

3.7 网络与嵌入式操作系统基础知识

3.7.1 网络操作系统

计算机网络系统除了硬件外，还需要有系统软件，二者结合构成计算机网络的基础平台。操作系统是最重要的系统软件。网络操作系统是网络用户和计算机网络之间的一个接口，它除了应具备通常操作系统应具备的基本功能外，还应有联网功能，支持网络体系结构和各种网络通信协议，提供网络互连功能，支持有效、可靠安全的数据传送。

一个典型的网络操作系统的特征包括硬件独立性，网络操作系统可以运行在不同的网络硬件上，可以通过网桥或路由器与别的网络连接；多用户支持，应能同时支持多个用户对网络的访问，应对信息资源提供完全的安全和保护功能；支持网络实用程序及其管理功能，如系统备份、安全管理、容错和性能控制；多种客户端支持，如 Windows NT 网络操作系统包括 OS/2、Windows 98 和 UNIX 等多种客户端，极大地方便了网络用户；提供目录服务，以单一逻辑的方式让用户访问位于世界范围内的所有网络服务和资源的技术；支持多种增值服务，如文件服务、打印服务、通信服务和数据库服务等。

网络操作系统可分为如下三类。

（1）集中模式。集中式网络操作系统是由分时操作系统加上网络功能演变而来的，系统的基本单元是由一台主机和若干台与主机相连的终端构成，将多台主机连接起来形成了网络，信息的处理和控制是集中的。UNIX 就是这类系统的典型例子。

（2）客户端/服务器模式。这是流行的网络工作模式。该种模式网络可分为服务器和客户端。服务器是网络的控制中心，其任务是向客户端提供一种或多种服务，服务器可有多种类型，如提供文件/打印服务的文件服务器等。客户端是用于本地处理和访问服务器的站点，在客户端中包含了本地处理软件和访问服务器上服务程序的软件接口。

（3）对等模式（peer-to-peer）模式。采用这种模式的操作系统网络中，各个站点是对等的。它既可作为客户端去访问其他站点，又可作为服务器向其他站点提供服务，在网络中既无服务处理中心，也无控制中心，或者说，网络的服务和控制功能分布在各个站点上。可见，该模式具有分布处理及分布控制的特征。

现代操作系统已把网络功能包含到操作系统的内核中，作为操作系统核心功能的一个组成部分。微软公司的 Windows NT、AT & T 公司的 UNIX System V、Sun 公司的 SunOS、HP 公司的 HP/OX、IBM 公司的 AIX 和 Linux 等都已把 TCP/IP 网络功能包含在内核中。

网络操作系统是整个网络的灵魂，它决定了网络的功能，并由此决定了不同网络的应用领域及方向。目前，网络操作系统主要有三大阵营：UNIX、Windows NT 和 NetWare。各种网络操作系统具有不同的特点，随着网络技术的发展，新的网络操作系统还会不断出现，用户可根据自己的需要进行选择，而不要仅局限于其技术水平的高低。

3.7.2 嵌入式操作系统

在嵌入式系统中的操作系统，称为嵌入式操作系统。嵌入式操作系统是运行在嵌入式智能芯片环境中，对整个智能芯片以及它所操作、控制的各种部件装置等资源进行统一协调、调度、指挥和控制的系统软件。嵌入式系统广泛应用于各种工业控制系统、计算机外设、微波炉、洗衣机和冰箱等低端设备；也用在信息化家电、掌上电脑、机顶盒、WAP 手机和路由器等高端设备中。

1．嵌入式操作系统的特点

一般而言，嵌入式操作系统不同于一般意义的计算机操作系统，它有占用空间小、执行效率高、方便进行个性化定制和软件要求固化存储等特点。嵌入式操作系统和其他嵌入式软件都具有如下特点。

（1）微型化。由于硬件平台的局限性，如主存少、字长短、运行速度有限、能源少（用微小型电池）、外部设备和控制对象千变万化，因此，不论从性能还是从成本角度考虑，都不允许它占用很多资源，系统代码量少，应保证应用功能的前提下，以微型化作为特点来设计嵌入式操作系统的结构与功能。

（2）可定制。嵌入式操作系统的运行平台多种多样，应用更是五花八门，所以表现出专业化的特点。从减少成本和缩短研发周期考虑，要求它能运行在不同的微处理器平台上，能针对

硬件变化进行结构与功能上的配置，以满足不同应用需要。

（3）实时性。嵌入式操作系统广泛应用于过程控制、数据采集、传输通信、多媒体信息及关键要害领域需要迅速响应的场合，实时响应要求严格，因此实时性是其主要特点之一。

（4）可靠性。系统构件、模块和体系结构必须达到应有的可靠性，对关键要害应用还要提供容错和防故障措施，以进一步提高可靠性。

（5）易移植性。为了提高系统的易移植性，通常采用硬件抽象层（Hardware Abstraction Level，HAL）和板级支撑包（Board Support Package，BSP）的底层设计技术。HAL 提供了与设备无关的特性，屏蔽硬件平台的细节和差异，向操作系统上层提供统一接口，保证了系统的可移植性。而一般由硬件厂家提供，按给定的编程规范完成 BSP，保证了嵌入式操作系统可在新推出的微处理器件平台上运行。

2．嵌入式系统开发环境

嵌入式系统开发环境通常配有源码级可配置的系统模块设计，丰富的同步原语，可选择的调度算法，可选择主存分配策略，定时器与计数器，多方式中断处理支持，多种异常处理选择，多种通信方式支持，标准C语言库，数学运算库和开放式应用程序接口。较著名的嵌入式操作系统有 Windows CE、VxWorks、pSOS、Palm OS 和μC/OS-Ⅱ。

3.8 UNIX 操作系统实例

3.8.1 UNIX 操作系统

UNIX 操作系统是由美国贝尔实验室发明的一种多用户、多任务的分时操作系统。现已发展成为当前使用普遍、影响深远的工业界主流的操作系统，成为重要的企业级操作平台。UNIX 广泛运行于 PC、小型机等各种环境，用于大型信息系统的关键业务服务，如数据库和 Internt 主机。UNIX 结构如图 3-29 所示。UNIX 最内层硬件提供基本服务，内核提供全部应用程序所需的各种服务。

3.8.2 UNIX 文件系统

UNIX 文件系统的目录结构是树型带交叉勾连的，根目录记为“/”，非叶节点为目录文件，叶节点可以是目录文件，也可以是文件或特殊文件。目录是一个包含目录项的文件，在逻辑上，可以认为每个目录项都包含一个文件名，同时还包含说明该文件属性的信息。文件属性包括文件类型、文件长度、文件主、文件的许可权（例如，其他用户能否访问该文件）和文件最后的修改时间等。当创建一个新目录时，系统自动创建了两个文件名：“.（称为点）”和“..（称

为点-点）”。点表示当前目录，点-点表示父目录。在最高层次的根目录中，点-点与点相同。某些 UNIX 文件系统限制文件名的最大长度为 14 个字符，BSD 版本则将这种限制扩展为 255 个字符。

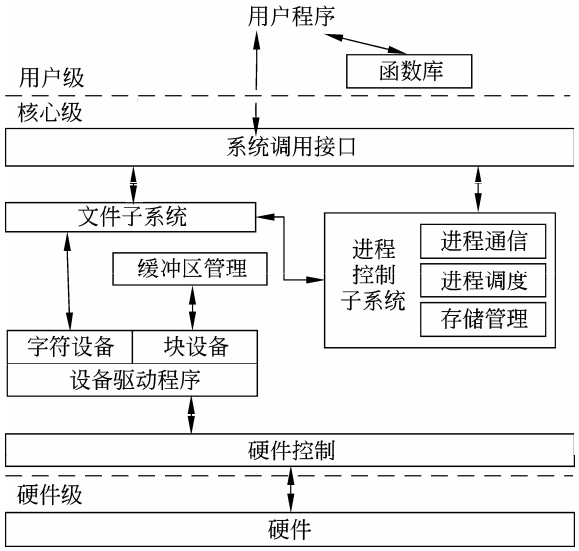


图 3-29　UNIX 系统结构

UNIX 文件系统具有图 3-30 所示的结构。

引导块	超级块	索引节点区	数据存储区

图 3-30　文件系统的布局

其中各参数的含义如下。

- 引导块：占据文件系统的开头，占一个物理块，包含引导代码段。
- 超级块：描述文件系统的状态，如容量、空闲块号和空闲索引节点号（i_node）等。
- 索引节点区：第一个索引节点是文件系统的根索引节点，当执行了 mount 命令之后，该文件系统的目录结构就可以从这个根索引节点开始进行存取了。
- 数据存储区：存放数据的区域。

进程可以通过系统调用访问文件，如 Open（打开文件）、Close（关闭文件）、Write（写文件）、Read（读文件）、Stat（查询文件属性）、Chmod（改变文件的许可权）、Chown（改变文件所有者）、Creat（创建一个文件）、Mkdir（创建一个目录文件）、Cd（改变当前目录）、Link（建立连接）和 Unlink（删除文件连接）等。

3.8.3 UNIX 进程与存储管理

1．进程管理

在 UNIX 中，进程由控制块（PCB）、正文段和数据段组成。其中，进程控制块由常驻主存的基本进程控制块（proc）和非常驻主存的进程扩充控制块（user）构成；正文段是可供多个进程使用，为了管理可共享的正文段，UNIX 设置了一张正文表 text[]，每个正文段都占据一个表项，用来指明正文段在主存和磁盘的位置；数据段是进程执行时用到的数据段，若进程执行的程序是非共享的，则也构成数据段的一部分。

1）进程控制

在 UNIX 中的进程控制子系统负责进程同步、进程间通信、存储管理及进程调度。控制进程的系统调用有 Fork，创建一个子进程；exec，改变执行程序的映像；exit，结束一个进程的执行；wait，暂停进程的执行，用于进程之间的同步，例如父进程等待子进程执行结束；signal，控制进程对特别事件的响应；kill，发送软中断信号；msgsnd，发送消息；msgrcv，接受消息等。

2）进程调度

UNIX 系统对进程的调度采用动态优先数调度算法，进程的优先数随进程的执行情况而变化。就绪进程是否能占用处理机的优先权取决于进程的优先数，优先数越小优先权越大。UNIX 系统中优先数确定方法有两种：设置方法和计算方法。设置方法用于要进入睡眠的进程，当进程正在或即将转入用户态运行时，用计算方法确定优先数。UNIX 计算优先数的公式为：

p-pri=p-cpu/2 + PUSER + p-nice + NZERO

其中，p-pri 表示进程的优先数；p-cpu 为处理器的占用时间；PUSER 和 NZERO 表示偏置常数；p-nice 允许用户使用 shell 命令 nice 或系统调用 nice 修改，用户有权将其值设置为 0～39 之间的数。

2．存储管理

UNIX 早期的版本采用“对换技术”扩充主存容量，进程可以被换出到对换区，也可以从对换区换进到主存。高版本的 UNIX 主存管理采用分页式虚拟存储机制，对换技术作为一种辅助手段。采用二次机会页面替换算法。

3.8.4 UNIX 设备管理

1．设备管理

在 UNIX 系统中，文件等于系统中可用的任何资源。UNIX 的设计者们遵循一条这样的规

则：UNIX 系统中可以使用的任何计算机资源都用一种统一的方法表示。他们选择用“ 文件 ”这个概念作为一切资源的抽象表示方法。

UNIX 系统包括两类设备：块设备和字符设备。用户可以通过文件系统与设备接口，因为每个设备有一个文件名，可以像文件那样存取。设备文件有一个索引节点，在文件系统目录中占据一个节点，但其索引节点上的文件类型与其他文件不同，是“ 块 ”或者是“ 字符 ”特殊文件。文件系统与设备驱动程序的接口是通过设备开关表。硬件与驱动程序之间的接口控制寄存器、I/O 指令，一旦出现设备中断，根据中断矢量转相应的中断处理程序，完成用户所要求的 I/O 任务。UNIX 设备管理的主要特点如下。

（1）块设备与字符设备具有相似的层次结构。这是指对它们的控制方法和所采用的数据结构、层次结构几乎相同。

（2）将设备作为一个特殊文件，并赋予一个文件名。这样，对设备的使用类似于对文件的存取，具有统一的接口。

（3）采用完善的缓冲区管理技术。引入“ 预先读 ”、“ 异步写 ”和“ 延迟写 ”方式，进一步提高系统效率。

2．输入输出转向

shell 即是一种命令语言，又是一种程序设计语言。在 UNIX 中，任何一个存放一条或多条命令的文件称为 shell 程序或 shell 过程。

UNIX 系统的 shell 向用户提供了输入输出转向命令，可以在不改变应用程序本身的情况下自由地改变其数据的输入源和输出目的地。其中，“ > ”、“ >> ”表示输出转向，“ < ”表示输入转向。例如，cat 命令用来将输入文件的数据显示在屏幕上：

```
cat input.txt
```

上述 cat 命令将 input.txt 文件中的内容输出到屏幕（标准输出设备）上。但是，如果将命令写成：

```
cat input.txt > output.txt
```

那么 cat 命令就会将原本输出到屏幕上的内容输入到文件 output.txt 中去并覆盖 output.txt 的内容。如果使用 cat input.txt >> output.txt，则将 input.txt 文件的内容添加到 output.txt 文件的末尾。

如果希望使用 cat 命令时没有指定输入文件，cat 就会要求用户在键盘（标准输入设备）上输入数据，直到用户输入 Ctrl+D 为止。但是，如果使用输入重定向符，就可以写成：

```
cat < input.txt
```

这样，cat 就不再从键盘上而是从 input.txt 文件中读取内容了。该条命令与 cat input.txt 在效果上是等价的，但它们代表的意义不同。没有使用重定向符的 cat 命令知道自己是在从文件中读取数据；而使用了重定向符的 cat 命令实际上并不知道自己得到的数据来自 input.txt，它只知道自己是从标准输入设备上读入信息的，shell 通过重定向耍了个花招，“骗”过了 cat 命令，使得它接收到的数据被调了包。输出重定向也一样，cat 命令在输出时也是只知道自己是向标准输出设备输出，而 shell 也同样用 output.txt 文件掉换了 cat 命令的标准输出设备，使得 cat 命令的输出转到了文件当中。

3．管道

在 UNIX 中“|”表示管道。一个管道总是连接两个命令，将左边命令的标准输出与右边命令的标准输入相连，于是左边命令的输出结果就直接成了右边命令的输入。这个功能使得用户可以在不改动程序本身的前提下使多个程序通过标准输入输出设备进行数据传递。利用管道命令可以简化命令行的写法，例如下面的三条命令：

```
ls > file
sort < file1 > file2
pr < file2
```

可以用管道命令表示为：

```
ls | sort | pr
```

例如，如果要统计当前目录中所有文件和目录中的数目，并将其记录在文件 output.txt 中。利用管道命令如下：

```
ls -a | wc -l > output.txt
```

其中，ls -a 用来列出当前目录下的所有文件和目录；wc -l 负责统计 ls 输出中的行数；最后将输出重定向到 output.txt 中。命令十分简单，完成的工作却相当复杂。

3.8.5 shell 程序

shell 不但负责管理命令行界面，而且 shell 自己也是一个编程的环境。实际上，可以将命令按照命令行的格式写入一个文件，再将其权限设置为可执行，就可以像普通命令一样执行它了。这个文件通常称为脚本（script）。熟悉 DOS 的用户自然想到 shell 脚本相当于 DOS 的批处理文件，而且 shell 脚本中也同样支持如 if、for 和 case 等程序控制流程，甚至还支持变量和函数定义。shell 实际上是一种编程语言。利用 shell 语言可以编写出功能很强的 shell 程序，将程

序段组合起来。

1．正则表达式

在 UNIX 中，正则表达式不仅用在 vi 中，还用在 shell 中。正则表达式用来确定字符串模式的一个规则集，是对文本字符串的一种描述，该描述能简洁而又完整地刻画文本字符串的关键特性。因此，正则表达式通常被用作字符串的匹配操作。正则表达式符号如表 3-6 所示。

表 3-6　正则表达式符号

符　　号	含　　义
.	能与除换行符之外的行内任何字符匹配
*	匹配前一字符的 0 次或多次出现
[]	[]中只能匹配一个字符，若要匹配多个，则要使用多个[]
^	如果方括号中的第一个字符是^，则匹配不属于[]中的字符
$	如果出现在正则表达式末尾，则表示行尾。$前面的正则表达式所匹配的字符串仅出现在行尾才匹配
\	转移符，用于改变特殊符号的含义，也可后跟一字符的八进制表示
""	双引号内的字符在匹配时忽视其特殊含义
\<	字首匹配
\>	字尾匹配

【例 3.17】 匹配字符串 32787、188567、1234567890 的正则表达式。

解 :这些字符串的共同特性都是数字, 若要匹配任意长度的数字序列可用如下正则表达式：

[0-9][0-9]*

而不能使用[0-9]*。

【例 3.18】 分别写出字符串 what 出现在行首和行尾的正则表达式。

解：/^what　当 what 出现在行首时才会被找到

/ what$　当 what 出现在行尾时才会被找到

若匹配文件的任意一行，应用^.*$。

2．shell 变量

shell 变量可分为三种类型：用户定义变量、系统定义变量和 shell 定义变量。

（1）用户定义变量。用户定义变量必须以字母或下划线开始，可以包含字母、下划线和数字的字符序列。用户定义的 shell 变量能用赋值语句置初值或重置值。例如 ux＝UNIX。

（2）系统定义变量。系统定义变量如表 3-7 所示。

表 3-7　常用的系统定义变量

变　量　名	含　　义
HOME	用户主目录名
PATH	定义 shell 在寻找命令时使用的查找路径
PS1	系统基本提示符，默认为$
PS2	系统辅助提示符，默认为>
IFS	内部字段分割符，默认是空格、制表符和换行符
MAIL	存放用户的邮件文件路径名
TERM	定义用户使用的终端类型
CDPATH	Cd 命令要查找的目录表
LOGNNAME	用户的注册名
SHELL	shell 程序的路径名
MANPATH	连接动态库时的搜索路径

（3）shell 定义变量。shell 定义变量如表 3-8 所示。

表 3-8　shell 定义变量

变　量　名	含　　义
$0	命令名，在 shell 程序内可用$0 获得调用该程序的名字
$1－$9	shell 程序的位置参量
$＃	位置参数的个数，不包括命令名
$*	所有位置参量，即相当于$1,$2,$3,…
$@	与$*基本相同，但当用双引号转义时，"$@"还是能分解成多个参数，而"$*"则合并成一个参数
$?	上一命令的返回代码，成功返回 0，否则返回 1
$$	当前命令的进程标识数
$!	shell 执行的最近后台进程标识数
$-	shell 标识位组成的字符串，可由 shell 传递来，或由 set 命令设置

3．shell 程序

shell 向用户提供了许多用于简化输入的符号，这些符号包括各种通配符、字符串定义符、转义符和变量定义符等。这些符号可以被看做是 shell 的保留字，通常称为“元字符”。元字符的种类和作用非常多，它们无论在 shell 的命令行输入还是在 shell 程序设计中都起着非常重要的作用。

【例 3.19】 编写显示用户登录名、用户主目录以及当前命令的进程标识符的 shell 程序。

解：Shell 程序如下。

```
echo username：$LOGNAME
echo Home directory：$HOME
echo Carrent shell's PID：$$
```

shell 命令行本身也是一个交互式的脚本执行环境，也就是说，在命令行上同样可以使用脚本中的控制语句，也可以定义变量（实际上就是环境变量），甚至可以定义函数。这都与脚本文件中的命令一样。但是有一点必须注意，shell 程序有许多种，不同的 shell 有不同的编程命令和语法。虽然它们基本上大同小异，但还是有许多差别。

【例 3.20】 在 UNIX 操作系统中，若用户输入的命令参数的个数为 1 时，执行 cat $1 命令；若用户输入的命令参数的个数为 2 时，执行 cat >> $2 < $1 命令。将下面所示的 shell 程序的空缺部分补齐。

```
case __（1）__ in
      1）cat $1 ;;
      2）cat >> $2 < $1;;
      *) echo 'default….  '
esac
```

分析：空（1）处应填写$#。因为在 UNIX 操作系统中，shell 定义变量$$、$@、$#和$*的含义如下。

- $$：表示当前命令的进程标识数。
- $@：与$*基本相同，但当用双引号转义时，"$@"还是能分解成多个参数，而"$*"则合并成一个参数。
- $#：表示位置参数的个数，不包括命令名。
- $*：表示所有位置参量，即相当于$1，$2，$3，…。

【例 3.21】 将下面 shell 程序段中的空缺部分补齐，使得它可以将指定的一个或多个输入文件的内容依次添加到输出文件的末尾。如果指定的输出文件不存在，则程序应自动产生一个输出文件。

```
if [ "$#" -lt 2 ]; then
        echo "Usage $0 <output-file> <input file 1> [<input file2>  …]"
        exit 0
fi
output="$1"
shift
```

```
for i in  （1）  ; do
        if [ -e "$i" ]; then # or use '-a $i'
          cat  （2）
        fi
done
```

（1）A. $#　　B. $i　　C. $！　　D. $@

（2）A. "$i" > $output　　B. "$i" >> $output

C. $i > $output　　D. $i >> $output

分析：程序的功能是指定一个或多个输入文件，将它们的内容依次添加到输出文件的末尾。如果指定的输出文件不存在，则程序应自动产生一个。本题命令行的格式应为：

```
concatenate-files.sh <输出文件> <输入文件 1> [<输入文件 2> …]
```

第一个 if 条件语句的条件段"$#" -lt 2 的含义是位置参量的个数小于 2，则显示提示信息并退出。

output="$1"语句的含义是将第一个参数作为输出文件。shift 语句的含义是将位置参量左移。空（1）应填$@，表示所有位置参量，即相当于$1，$2，$3…。

由于试题要求将它们的内容依次添加到输出文件的末尾。对于 UNIX 系统的 shell 向用户提供了输入输出转向命令，可以在不改变应用程序本身的情况下自由地改变其数据的输入源和输出目的地。其中，“ > ”、“ >> ”表示输出转向，“ < ”表示输入转向。例如，cat 命令用来将输入文件的数据显示在屏幕上。

```
cat input.txt
```

上述 cat 命令将 input.txt 文件中的内容输出到屏幕（标准输出设备）上。但是，如果将命令写成：

```
cat input.txt > output.txt
```

那么 cat 命令就会将原本输出到屏幕上的内容输入到文件 output.txt 中去并覆盖 output.txt 的内容。

如果使用 cat input.txt >> output.txt，则将 input.txt 文件的内容添加到 output.txt 文件的末尾。可见，空（2）应填"$i" >> $output。