普通高等教育“十五”国家级规划教材

Foundations of Software Technology

计算机软件技术导论

庞丽萍　张文彬　吴永英　李胜利　编

高等教育出版社
Higher Education Press

普通高等教育“十五”国家级规划教材

Foundations of Software Technology

计算机软件技术导论

庞丽萍　张文彬　吴永英　李胜利　编

高等教育出版社

内容提要

本书是普通高等教育“十五”国家级规划教材。

本书站在计算学科的高度上，勾画计算机软件技术较完整的视图，讲解了计算机软件技术的核心内容。全书共分6章。第一章概论，概要地阐述计算学科的主要领域以及计算机软件的核心概念，讨论了软件开发方法与技术。第二章数据结构与算法，介绍最基本的数据结构及其应用实例，给出了常用的查找与排序算法。第三章操作系统及应用，讨论计算机核心系统软件——操作系统的基本概念、用户界面以及并发活动处理与系统资源的管理，介绍了当前使用广泛的两种操作系统——Windows系统和Linux系统的特点、结构及其使用方法。第四章数据库系统及应用，阐述数据模型和关系数据库基础，给出了数据库应用系统的设计方法，介绍了当前流行的数据库管理系统。第五章计算机网络及应用，给出了计算机网络的基本知识，并从应用的角度出发简单介绍了网络互连与Internet的应用。第六章实验与指导，给出了14个实验，涉及数据结构与算法、操作系统、数据库系统和计算机网络的应用。

本书适合作为高等学校非计算机专业、计算机应用专业（大专）的教材，亦可供从事计算机应用的广大工程技术人员和管理人员自学参考。

图书在版编目（CIP）数据

计算机软件技术导论/庞丽萍等编. —北京：高等教育出版社，2004.8

ISBN 7-04-015126-X

Ⅰ. 计… Ⅱ. 庞… Ⅲ. 软件-高等学校-教材 Ⅳ. TP31

中国版本图书馆CIP数据核字（2004）第067256号

出版发行	高等教育出版社	购书热线	010-64054588
社　　址	北京市西城区德外大街4号	免费咨询	800-810-0598
邮政编码	100011	网　　址	http://www.hep.edu.cn
总　　机	010-82028899		http://www.hep.com.cn
经　　销	新华书店北京发行所		
排　　版	高等教育出版社照排中心		
印　　刷	北京市鑫霸印务有限公司		
开　　本	787×1092　1/16	版　　次	2004年8月第1版
印　　张	23.5	印　　次	2004年8月第1次印刷
字　　数	480 000	定　　价	29.30元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

作者简介

庞丽萍 1944年7月生，浙江省宁波人。1967年毕业于北京邮电学院。现任华中科技大学教授，博士生导师，曾任国家教育部工科计算机基础课程教学指导委员会委员、中国计算机学会教育与培训专业委员会委员。长期从事分布式计算机系统、操作系统的研究和教学工作。主持并完成湖北省教学改革项目两项。正式出版《操作系统原理》(电子工业部九五规划教材)、《软件开发基础——面向对象技术与操作系统虚拟机》(教育部面向21世纪课程教材)、《计算机软件技术导论》(教育部十五规划教材)等10多部教材。其中《操作系统原理》(第三版)获2002年教育部优秀教材二等奖。2003年获湖北省科技进步一等奖。1995年获香港柏宁顿(中国)教育基金会首届“孺子牛金球奖”。

教育部高等学校非计算机专业计算机基础课程教学指导分委员会推荐教材出版说明

进入21世纪之后，我国明显地加快了建设世界教育大国的步伐，现在正向世界教育强国的目标迈进。实现这个历史性任务的最为关键指标是要有国际公认的高等教育质量，而高水平的教材是一流教育质量的重要保证。

在“九五”和“十五”期间，两届计算机基础课程教学指导委员会都把教材建设列为重点工作。非计算机专业计算机基础课程的教育部“面向21世纪课程教材”和“普通高等教育‘十五’国家级规划教材”均取得了可喜成果，教材被选用率高，不少还被评为国家、省部级的优秀教材。

本届教学指导分委员会一直着力于研究在新形势下，如何进一步加强高校的计算机基础教学。提出了许多重大的改革举措、新的课程体系框架，计算机基础教学的内容组织和课程设置已反复与各高校教务部门、有关教师研讨，取得许多共识；更令人兴奋的是广大高校表现出极大的热情，一批有创新、改革精神，且有丰富教学经验的教师积极投身到新一轮的计算机基础课程教材编写中。我们对这些教师表示深深的敬意，感谢他们用自己创造性的思维、辛勤的汗水诠释本届教指委的改革思想，把教指委新设计的课程体系和教学内容生动地传达给师生，进行有意义的教学实践。

为了把计算机基础教育的优秀教材及时地推荐给广大从事计算机基础教育的教师和同学，便于他们选用和研究，我们新设计开发了本届教指委组织推荐的“计算机基础课程系列教材”，并将已经出版和即将新出的部分“面向21世纪课程教材”、“普通高等教育‘十五’国家级规划教材”与这些新编教材进行了整体规划，系统组织，内容严格把关，形成符合新的教学基本要求的新的教材体系，希望这些教材的出版能起到推动计算机基础教育改革的作用，使我们高校的计算机基础教育质量更上一个台阶。

计算机基础教育改革一直在不断地深化，课程体系和教学内容趋于更加合理和科学。本系列教材与以前出版的教材比较会有较大的变化，这也是我们期待的。

每一本教材都有它的适用范围，面向不同办学层次、学科、地域和人才培养模式的教材必然有差异。本系列教材将会考虑这种差异，以满足各种层次和类型的教学所需。

列入本系列的教材，当在国内同类教材的优秀之列，我们希望作者把它打造成国家级的精品教材，要求做到“三新”，即体系新、内容新、方法新；每一本教材都做成既有文字教材、又有电子教材，既有教科书、又有辅助教材，成为真正意义上的“立体化”。教材的出版仅是“万里长征的第一步”，要成为精品教材，作者还必须根据读者的反映和需求不断修订原作，真正做到“与时俱进”。

“一切为了教学，一切为了读者”是我们的心愿，书中不足之处，恳望教师和同学们指正。

教育部高等学校非计算机专业计算机基础课程教学指导分委员会

2004 年 6 月

前　言

随着计算机应用的日益广泛和深入，越来越多的人们加入到计算机应用开发和使用的行列中来。他们对计算机应用的相关方面越来越感兴趣，越来越有迫切的需求。他们希望所面对的计算机应用环境、应用开发的方法和技术不是“黑匣子”；希望所面对的计算机应用技术不是只停留在知其然、不知其所以然的水平上。

计算机应用的开发和实施涉及对应用问题的认识、信息的表示和信息的处理等方面。它包括如何描述客观世界中的问题和解决已描述问题的方法和环境。前一个问题涉及计算机软件系统与实现技术、软件开发方法和工具；后一个问题是软件开发的环境和平台。当前，计算机应用的环境和平台是多用户、多任务系统；或者是局域网、广域网的环境，在这样的平台中，除了作为物质基础的硬件系统外，为信息定义、信息描述、信息处理提供方法、技术和优质服务(QoS)的各种软件也是实现计算机应用的技术基础。从需求牵引的观点出发，了解和认识计算机软件这一计算机系统的灵魂是至关重要的。人们既然不满足只会使用，不了解内幕的现状，就应该对计算机软件及其开发技术有一个最基本、最核心的了解。

从应用的角度出发，给出一个最基本、较完整的软件技术导论是迫切需要的，但不是一件容易的事。因为，计算机软件涉及的内容非常丰富，它的每一个方面都是计算机科学中的重要分支。本书的宗旨是给出计算机软件技术的导论，站在计算学科的高度上，概要地阐述计算学科的主要领域以及计算机软件的核心概念，讨论应用问题的抽象与信息描述方法、算法与数据结构；程序设计技术；软件开发方法与技术；核心系统软件——操作系统、重要的系统支撑软件——数据库系统以及计算机网络的基本概念、原理、实现技术及使用。

为了能给出实用的、有实质性内容的、又易为非计算机专业的学生和其他初学者所掌握的计算机软件技术导论，编者对本书的深度和广度进行了深入的讨论，遵循注重基础、强调应用，力求深入浅出、通俗易懂、循序渐进的原则，对本书的内容作了精心的选择和安排。书中的例子和实验的选择经过认真推敲，并通过上机验证以确保其正确性；每章配有小结和习题，以使读者能更好地学习和掌握软件技术的基础知识和基本技能。本书的编者是华中科技大学(原华中理工大学)计算机学院长期从事计算机专业的教学和科学研究工作的教师。这些教师对计算学科有深入的理解和认识，对教学有丰富的经验，对计算机应用有广泛的了解，对学生有深厚的感情，因而使本书的内容和质量得以保证。

本书共分6章。第一章概论，站在计算学科的高度上，概要地阐述计算学科的主要领域以及计算机软件的核心概念，勾画了软件技术较完整的视图。第二章数据结构与算法，介绍了最基本的数据结构及其应用实例，给出了常用的查找与排序算法。第三章操作系统及应

用,讨论了计算机核心系统软件——操作系统的基本概念、用户界面以及并发活动处理与系统资源的管理,介绍了当前使用广泛的两种操作系统——Windows 系统和 Linux 系统的特点、结构及其使用方法。第四章数据库系统及应用,阐述了数据模型和关系数据库基础,给出了数据库应用系统的设计方法,介绍了当前流行的数据库管理系统。第五章计算机网络及应用,给出了计算机网络的基本知识,并从应用的角度出发简单介绍了网络互连与 Internet 的应用。第六章实验与指导,给出了 14 个实验,涉及数据结构与算法、操作系统、数据库系统和计算机网络的应用,通过这一组实验,可以加深对概念的理解,提高计算机应用的基本技能。

本书由庞丽萍教授任主编,进行了全书的统一修改与定稿,并编写了第一章和第三章,其他各章的编者依次是:第二章张文彬副教授,第四章吴永英副教授,第五章李胜利教授,第六章中各实验由撰写相应内容的教师完成。同时感谢杨一平老师为本书审稿。

本书适用于高等学校非计算机专业、计算机应用专业(大专)的教材,亦可供从事计算机应用的广大工程技术人员和管理人员自学参考。本书作为教材,是一门计算机软件知识的综合课程,讲授学时可安排为 32 ~ 40 学时,实验学时可安排为 16 ~ 20 学时,选做第六章中的部分内容。课堂讲授应注意取材,即讲授教材中最基本的内容,而有些小节的内容可让学生自学。

由于计算机软件技术导论课程自身的特点,我们的教材难免存在不足和疏漏之处,恳请广大读者提出宝贵意见。

编 者

2004 年 3 月于武汉

目　录

第一章 概 述

在科学技术飞速发展的当今时代，计算机科学技术以磅礴之势迅猛向前推进。计算机的各种应用已广泛深入到人类活动的各个领域，对人类社会的进步与发展产生了巨大的影响。计算机应用是计算机科学技术和各领域的实际密切结合的产物，没有计算机学科的理论、方法作为指导，没有计算机技术和设计过程，也就不会有现代计算机应用；但没有各领域的实际应用需求，没有实际应用作为基础，也就没有具有真正实用价值的应用系统。

随着计算机应用的日益广泛，更多的领域涉及计算机，更多的人们接触计算机，感受到计算机应用的强大功能。但也有不少人对计算机及其技术的认识存在误区，不少人认为计算机仅仅是一个工具，计算机解决的仅仅是编程问题，计算机没有什么理论可言。这些看法是不对的，或者说是片面的。

计算机科学与技术随着计算机应用的快速发展，新技术、新方法不断产生，人们对它的认识也不断深入，计算机科学与技术已发展、扩展为一个学科——计算学科。计算学科作为一个学科，包含重要的研究领域，具备理论、抽象、设计等科学方法。计算学科中最根本的研究方法是数学方法，数学以高度抽象性、逻辑严密性、普遍适用性指导计算学科的研究工作，并为计算学科的研究提供了简洁精确的形式化语言，提供分析和计算方法和逻辑推理工具。计算学科采用一些重要的系统、科学的方法，如模型方法、结构化方法和面向对象方法等。

对计算的本质、计算学科的研究领域及主要内容、计算学科的核心概念和计算学科中的系统科学方法有一个初步的认识，将有助于理解计算机软件的核心基础、软件技术涉及的问题以及软件开发的科学方法。

1.1 计算学科及其研究内容

国际上最有影响的（美国）计算机协会（Association for Computing Machinery，ACM）和（美国）电气和电子工程师学会计算机分会（Institute of Electrical and Electronics Engineers - Computer Society，IEEE - CS）联手组成攻关组，用新的思维方式来认识计算学科。1989 年该攻关组提交了《计算作为一门学科》（Computing as a Discipline）的报告。接着，IEEE - CS 和 ACM 又在《计算作为一门学科》报告的基础上进行计算机学科教学计划的研究工作，分别形成了 Computing Curricula 1991（简称 CC1991）和 Computing Curricula 2001（简称 CC2001）。

1.1.1 计算学科的研究领域

1. 计算学科的主领域

计算学科是一门范围宽广的学科,在学科中,将具有共同的、能反映学科一个方面本质特征的内容归纳为一个主领域,这样有利于对学科的理解。在 CC2001 中,根据计算领域中技术和文化方面的变化,将整个学科划分为 14 个主领域。这 14 个领域的名称和缩写如表 1.1.1所示。

表 1.1.1 计算学科的主领域

序号	学科领域缩写	学科领域名称
1	DS	离散结构
2	PF	程序设计基础
3	AL	算法与复杂性
4	AR	体系结构
5	OS	操作系统
6	NC	网络计算
7	PL	程序设计语言
8	HC	人机交互
9	GV	图形化和可视化计算
10	IS	智能系统
11	IM	信息管理
12	SE	软件工程
13	SP	社会和职业问题
14	CN	科学计算

2. 计算学科主领域的基本问题

计算学科共有 14 个主领域,每个领域中都有它所包含的内容和该领域的基本问题。

(1) 离散结构

计算学科要解决的根本问题是:什么能被自动地进行。所谓自动进行,也就是“能行性”问题。大家所熟悉的计算机具有二进制机器的本质特点,任何问题最终都会变换为“0”和“1”两个字符,而这两个字符可以有任何的解释。如“0”代表对,“1”代表错;“0”表示输入,“1”表示输出等。计算机所具有的“能行性”决定了它所处理的对象只能是离散的,连续

的对象很难进行自动处理。那些连续对象(如声音、图像等)都必须正确地转换为离散对象后,才能被计算机处理;许多连续性问题也必须转换为离散型问题以后,才能被计算机处理。例如,计算定积分就是通过先将它变成离散量,再用分段求和的方法来处理的。

计算学科的"能行性"这一根本问题决定了计算机本身的结构和它处理的对象都是离散的。在计算学科中有一个重要的学科方向是离散数学,它研究计算学科中用到的离散量的各种数学问题,并对这些数学问题进行系统、全面的分析和论证,为计算学科提供有力的理论基础和工具。目前,为了强调计算学科对离散数学的依赖性,将它作为学科的一个主领域,命名为"离散结构"。

离散结构的主要内容包括:集合论、逻辑学、图论和组合数学等。该领域与计算学科的其他领域有着密切的联系,为计算学科各领域解决各自的基本问题提供了强有力的数学理论和工具。例如,数据结构与算法分析中有大量的离散结构的内容;在操作系统、编译系统和计算机网络中会用到图论的概念;在软件工程和数据库中会用到集合论的概念等。

(2) 程序设计基础

程序设计基础的主要内容包括程序设计基本结构、算法与问题求解、数据结构等。

程序设计的基本问题有以下几个:

- 对给定的问题,如何进行有效的描述并给出算法?
- 如何确定基本数据结构?
- 如何进行设计、编码、调试和测试?

(3) 程序设计语言

程序设计语言的主要内容包括:程序设计范型、虚拟机、语言翻译系统、声明和类型、抽象机制以及面向对象程序设计等。

程序设计语言研究语言本身的数据类型、操作、控制结构以及新的类型和操作机制,以抽象的原则认识过程、函数和参数化机制;研究程序设计语言的语义,即什么样的语言表示(语义)可以有效地用来描述计算机应该做什么;研究从语言角度如何组织虚拟机,即语言表示的虚拟机的层次结构。

(4) 算法与复杂性

算法与复杂性包括算法分析基础、典型的算法策略、基本算法、分布式算法以及可计算理论等核心内容,还包括自动机理论、高级算法分析、并行算法和容错算法等扩充内容。

算法与复杂性涉及的主要问题是:

- 对于给定的问题,如何分析算法的时间复杂性和空间复杂性。
- 最好、最坏和平均情况下的复杂性差异。
- 算法的效率和性能问题。

(5) 操作系统

操作系统的主要内容包括:操作系统结构化设计、并发处理、资源分配与调度、各类资源

的管理策略和实现技术、安全与保护、系统性能评价等。

操作系统是裸机(计算机硬件构成的机器)上的第一层核心系统软件,通过操作系统对系统各类资源的有效管理和调度,对多用户、多任务的并发活动实施有效的控制和协调,为用户提供了一个功能强大、使用方便的多用户、多任务工作环境。

操作系统主领域的主要问题是:

- 如何对物理资源进行抽象,为用户提供虚拟的逻辑资源。
- 对每一类资源调度分配的策略是什么?并发任务间的通信、协调策略、方法和机制是什么?
- 在操作系统的不同层次上,可见的对象和允许的操作各是什么?
- 如何组织用户接口,使用户能方便、有效地与操作系统交互。

(6) 信息管理

信息管理的核心内容包括:信息的模型与信息系统、数据建模、关系数据库、数据库查询语言、关系数据库设计、事务处理等,还包括分布式数据库、数据挖掘、信息存储和信息检索、超文本和超媒体、多媒体信息和系统、数据图书馆等扩充内容。

信息管理的主要问题是:

- 数据模型问题,包括数据的逻辑结构和数据元素之间的关系模型(如 E - R 模型、关系模型、面向对象的模型等)。
- 如何组成有效的事务?如何能有效地与用户交互?
- 信息的安全性、隐私性、完整性。

(7) 软件工程

软件工程的主要内容包括:软件过程、软件需求与规格说明、软件设计、软件演化、软件项目管理、软件开发工具与环境等,还包括扩充的形式化方法、软件可靠性、基于构件的计算等内容。

软件工程领域的研究涉及构建满足用户需求的软件系统所必须的理论、知识、设计和实践过程。用软件工程的科学方法进行各类软件系统的开发,包括了软件系统生存周期的各个阶段:需求分析、设计、构建、测试、运行和维护。软件系统在开发过程中,开发小组或个人都必须选择最合适的开发环境、工具、方法和途径,以确保软件系统的质量。

(8) 体系结构

体系结构主要内容包括:数字逻辑、数据的机器级表示、汇编级机器组织、存储技术、接口和通信、多处理和其他系统结构、性能优化、网络与分布式系统结构等。

计算机体系结构主要研究计算机内部结构、功能部件、控制机制以及性能等问题,还提出了对于计算学科而言,如何设计和控制大型计算机系统,如何设计能有效地进行并行处理的体系结构等问题。

(9) 网络计算

网络计算的主要内容包括网络的体系结构、网络安全、网络管理、多媒体数据技术、无线和移动计算等。

计算机技术和网络通信技术的快速发展和紧密结合，特别是基于 TCP/IP 网络的发展，使计算机网络和基于网络的应用变得十分重要。在计算机网络及其网络计算的领域中，主要研究计算机网络协议、万维网标准和技术、网络安全、多媒体系统、移动计算和基于计算机网络构建的分布式系统等。

(10) 其他领域

在计算学科的主领域中，除了上述 9 个领域之外，还有人机交互、图形化和可视化计算、智能系统、社会和职业问题、科学计算等 5 个领域。下面，就这 5 个领域作一简要介绍。

人机交互领域的主要内容包括以人为本的软件开发和评估、图形用户界面的时间和编程、多媒体系统的人机接口等。

图形化和可视化计算领域的主要内容包括：计算机图形学、可视化、虚拟现实、计算机视觉等。

智能系统领域研究搜索和约束可满足性问题、知识表示和推理、高级搜索 (gent)、自然语言处理技术、机器学习和神经网络、人工智能规划系统和机器人等问题。

社会和职业问题领域的主要内容包括：计算的历史、计算的社会背景、分析方法和工具、职业责任和道德责任、基于计算机系统的风险与责任、知识产权、隐私与公民的自由、计算机犯罪、与计算有关的经济问题等。

科学计算领域的主要内容是数值分析、建模与模拟、运筹学、高性能计算等。

1.1.2 计算学科的 3 个重要过程

计算学科作为一门学科包含如下 3 个问题：对学科的认识过程、学科自身的理论和研究方法。这 3 个问题在计算学科中表现为 3 个过程，或称 3 个形态——抽象、理论、设计。对计算的本质、计算学科的本质内容需要进行不断的抽象过程，这是一个认识过程，通过抽象可以得到计算学科的感性认识，在此基础上，逐步形成对计算学科的理性认识，即为理论过程。

在科学技术方法中，科学抽象是指对同类事物去除表面的、次要的方面，抽取共同的、主要的方面，从而能从个别中把握一般，从现象中把握本质的认识过程和思维过程。科学抽象是实现从感性认识到理性认识飞跃的决定性环节。从科学抽象的结果可以得出一些科学概念、科学符号和模型。

对学科的感性认识上升到理性认识，就形成了学科的理论。这一理论应经过实践的检验，最后形成学科的知识体系，包括科学概念、科学理论以及整个体系。

在感性认识和理性认识的基础上，就可以进行过程设计，以完成一个具体任务。对工程设计中遇到的问题可以由理论去指导、解决；同时，对工程设计中所积累的经验和教训进行

总结,可以形成设计方法。例如,存储程序式计算机(冯·诺依曼计算机)就是计算机组成结构的设计方法,它的实现是一个工程设计过程,也就是计算学科的第3个形态——设计过程。

从计算学科的3个学科形态——抽象、理论和设计,可以看出计算学科具有理论性、实践性紧密结合的学科特征,计算领域中人们的认识从感性认识(抽象)到理性认识(理论),再由理论认识(理论)回到实践(设计)中去。这是一个科学的思维方法。

例如,对高级语言的认识也体现了抽象、理论和设计3个过程。这3个过程具体体现如下:

(1) 抽象

- 高级语言中的常用符号,如数字(0~9),大小写字母(A~Z,a~z),括号、运算符(+、-、*、/)等。
- 各种数据类型的抽象。
- 算法的高级语言描述。
- 语言编译系统中的词法分析、编译、代码优化方法。

(2) 理论

- 形式语言和自动机理论。
- 形式语义学。

(3) 设计

- 特定的语言,如过程语言 COBOL、FORTRAN、C 等;函数式语言 Lisp;面向对象语言 Smalltalk、C++;并发程序设计语言 Concurrent Pascal、Modula 2。
- 词法分析器和扫描器的产生器,如 YACC、LEX。
- 编译器产生器。
- 语法和语义检查,调试和跟踪器。

1.1.3 计算学科及其研究内容

1. 什么是计算学科

什么是计算学科? 1990年,ACM 和 IEEE-CS 联合攻关组在《计算作为一门学科》报告中给计算学科作了如下描述:计算学科是研究在计算机上建立模型并通过模拟物理过程进行科学调查和研究的学科。该学科对信息描述和变换算法进行系统研究,这一研究包括理论、分析、效率、实现和应用等内容。

计算学科来源于对计算本质的认识,通过计算理论的研究、计算模型的建立、自动计算机器的设计研究、存储程序式计算机的研制成功,于20世纪40年代初期开始形成。半个世纪以来,随着计算机应用领域越来越宽广,计算机技术的发展越来越迅猛,使计算学科在理论和实践两个方面都得以快速发展,并更密切地结合。在这样的基础上,计算学科的内容不

断丰富,认知程度日益深入,导致计算学科已成为一个极为宽广的学科。

2. 计算学科的研究内容

计算学科研究的根本问题是:什么能被有效地自动进行。围绕这一根本问题,计算学科的研究归结为算法与可计算性研究、可计算硬件和可计算软件的设计与实现问题。所以,计算学科的研究内容包括:从总体上对算法和信息处理过程的研究;对满足给定规格要求的有效而可靠的软硬件设计,包含所有领域的理论研究、实验方法和工程设计。

算法和可计算性研究包含以下内容:求解问题如何抽象为数学模型;如何进行形式化描述;选择什么算法进行处理;是否能被有效地自动处理;算法的性能和效率如何。所有这些可归纳为对算法和信息处理过程进行的研究。

可计算硬件的设计与实现包含以下内容:如何构造可进行计算的计算机;功能部件、控制机制、工作原理是什么;机器上的数据如何存放,怎样被处理;如何构造单机系统或由多个处理部件构成的系统;如何实现系统中信息的交互和通信。

可计算软件的设计实现包含以下内容:对于给定的问题,如何进行程序设计;程序设计语言的结构和特征是什么;如何构造操作系统虚拟机;按语言的级别不同如何构造各种语言的虚拟机;如何构造信息管理系统;以什么样的科学方法开发软件系统,以保证软件系统的正确性、方便性、可维护性。

所以,计算学科的研究内容包括:从总体上对算法和信息处理过程的研究;对满足给定规格要求的有效而可靠的软硬件设计,包含所有领域的理论研究、实验方法和工程设计。

1.2 计算的本质与计算机系统

1.2.1 计算的本质

在远古时代,人们就遇到了需要计数的问题,例如结绳记事的传说;有经过考证在旧石器时代刻在骨制的或石头上的记号。这些记号是对某种计算的记录。从远古至今,人们没有停止过计算。我们的祖先对在进行生产活动和商业交易中需要解决的问题,归结为算术四则运算问题。从最初用大脑和手来计算,到后来大量使用算盘。人们要解决某一个问题,只有将问题的求解方法归结为算术四则运算后,才能用算盘来进行计算。我国的学者认为,对于一个数学问题,只有当确定了其可用算盘解算它的规则时,这个问题才算可解。这就是古代中国对“算法化”的认识,这一认识也就是对计算的根本问题——“能行性”问题的理解。

世界各国的人们都在进行计算,并在逐步认识计算的本质。到中世纪,有些哲学家就提

出:能否用机械来实现人脑活动的个别功能? 人们不断地进行研究和实验。1641 年,法国人帕斯卡(B. Pascal)利用齿轮技术制造了第一台加法器。1673 年德国人莱布尼茨(G. W. V. Leibniz)在帕斯卡研究的基础上又制造了能进行简单加、减、乘、除的计算机器。

人们不断地计算、试验和探索,与此同时,认识计算本质所必须的形式化的研究也在进行中。形式化方法和理论的研究是数学的基础研究,需要对数学的对象、性质及其发生、发展的一般规律进行科学研究。从 1854 年英国数学家乔治·布尔(G. Boole)创立逻辑代数体系(即布尔代数)开始,到 1874 年德国数学家康托尔(G. Cantor)创立集合论逐步为现代计算机的诞生作了主要的理论准备。

20 世纪 30 年代后期,数学家图灵(A. M. Turing)通过对一个数的计算过程的研究,实现了对计算本质的认识。图灵对计算本质的认识,以通俗方式描述如下:所谓计算就是计算者(人或机器)对一条两端可无限延长的纸带上的一串 0 和 1 执行指令,一步一步地改变纸带上的 0 或 1,经过有限步骤,最后得到一个满足预先规定的符号串的变换过程。

图灵用形式化方法表述了计算的本质,它深刻地揭示了计算所具有的"能行过程"的本质特征。

以上的描述实质上是针对数值计算而言的。众所周知,大量的计算机应用问题可以分为数值计算、数据处理、过程控制、人工智能等几大类。那么,上述的计算本质特征能否解决各类不同的计算机应用问题呢? 计算的本质是可以处理 0 和 1 组成的代码串,而英文字母和汉字均能用二进制数表示,这样,由数值和非数值(英文字母、汉字、数字)组成的字符串,既可以解释为数据,也可以解释为指令。在这样的前提下,在计算机上处理的任何问题,只要将其需要处理的过程(程序)和数据用字符串的形式进行编码,通过设计好的、"能行的"计算机硬件将程序和数据存储在计算机的存储器中,通过设计好的、"能行的"计算机软件,将程序翻译成计算机能认识的形式,再由硬件的处理器执行,最后得到计算的结果。

1.2.2 图灵机与冯·诺依曼型计算机

1. 图灵机

数学家图灵通过对数的计算过程的研究,提出了计算的本质。基于这种计算模型可以设想一种可以用于计算的机器,称为图灵机。图灵机是从计算过程这一角度刻画了计算的本质,通过一系列动作对一串 0 和 1 的符号进行变换,经过有限步骤,最后能得到一个满足预定要求的变换结果。以这种思想方法建立的图灵机结构简单、操作运算规则也较少,可为大家所理解。

图灵机可以作为一个数学机器去理解,但它不是一个实际的计算机。在此计算模型的基础上可以构建实际的计算机,这需要研究具体的实现方法和实现技术。

2. 冯·诺依曼型计算机

在图灵机研究的基础上,著名数学家冯·诺依曼(Von Neumann)总结了手工操作的规

律以及前人研究计算机的经验教训后，提出了“存储程序式计算机”方案，从而使计算实现了自动化。

要使计算机能够自动地计算，必须使机器可以“看到”计算方案，即计算机程序，能够“理解”程序语言的含义并顺序执行指定的操作，可以及时取得初始数据和中间数据，能够自动地输出结果。于是，机器必须有一个存储器，用来存储程序和数据；有一个运算器，用以执行指定的操作；有一个控制部件，以便实现自动操作；此外，还要有输入/输出（或简称 I/O）部件，以便输入原始数据和输出计算结果。由存储器、运算器、控制器、输入/输出部件构成的机器，将程序和数据一样看待，存储在存储器中，由计算机进行加工和处理，这样的机器称为“存储程序式计算机”，或称“Von Neumann 计算机”。

存储程序式计算机由以下 5 类部件组成：控制器、运算器、存储器、输入装置和输出装置。人们通常把控制器和运算器做在一起，称为中央处理机或中央处理部件（CPU）。输入装置和输出装置统称为 I/O 设备，如图 1.2.1 所示。时至今日，大多数计算机还是采用这种结构。

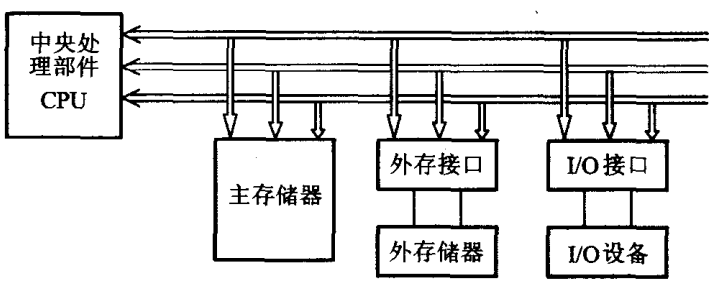


图 1.2.1 典型的单处理机系统结构

CPU 是计算机的“大脑”，能控制、指挥各个部件的工作。它是一种能够解释指令、执行指令并控制操作顺序的硬设备。在 CPU 中，控制器负责从主存储器提取指令并分析其类型。运算器则完成为实现该指令所需进行的操作。CPU 还包含一个小的高速存储器，用来存储一些暂时的结果和其他控制信息，这个存储器由若干个寄存器组成，每一个都具有某种功能，其中很重要的一个寄存器是程序计数器（PC），它指示下一步应该执行的指令。

存储器是计算机存储程序和数据的部件。如果没有一个使中央处理机能直接读、写信息的存储器，那就不存在用户所熟悉的可存储程序的数字计算机了。早期的主存储器是由磁芯做成的，价格比较昂贵。现在，主存大多数由半导体芯片组成，其容量可达到 32 MB、64 MB、128 MB，甚至更大。大部分计算机还有一个与主存相比，其存取速度较慢、价格较便宜、容量大得多的辅助存储器，用于保存大量的数据信息。现在，辅存的容量可达到 10 GB、20 GB，甚至更大。

I/O 设备则是完成信息传输任务的。当某一个问题需要计算机处理时，必须给定程序和初始数据，这些信息是通过输入设备进入计算机的；当得出解答后，计算机必须把计算结果通知用户，这是通过输出设备实现的。另外，还有一个让操作员用来实施控制和发布命令的控制台。

Von Neumann 计算机是人类历史上第一次实现自动计算的计算机，可以真正称得上是一架自动机。该机是人类历史上第一次出现的作为人脑延伸的智能工具，它的影响是十分深远的。它具有逻辑判断能力和自动连续运算能力。它的计算模型是顺序过程计算模型，其主要特点是：集中顺序过程控制，即控制部件根据程序对整个计算机的活动实行集中过程控制，并根据程序规定的顺序依次执行每一个操作。计算是过程性的，故这种计算机是模拟人们的手工计算的产物，即首先取原始数据，执行一个操作，将中间结果保存起来，再取一个数，和中间结果一起又执行一个操作，如此计算下去。在遇到有多个可能同时执行的分支时，也是先执行完一个分支，然后再执行第二个分支，直到计算完毕。由于 Von Neumann 计算机的计算模型是顺序过程计算模型，所以它的特点是集中顺序过程控制。

1.2.3 计算机系统的组成与操作系统虚拟机

1. 计算机系统的组成

现代计算机系统由计算机硬件和软件两大部分组成，如图 1.2.2 所示。

硬件是组成计算机的任何机械的、磁性的、电子的装置和部件。硬件也称为硬设备，它是由中央处理机、存储器和各种外部设备等组成的。由这些硬部件组成的机器称为裸机。裸机不能称为一个计算机系统，因为裸机上没有任何可以协助用户解决问题的方法和手段。想要直接在裸机上运行用户程序或处理某些应用，将是十分困难的，甚至是不可能的。

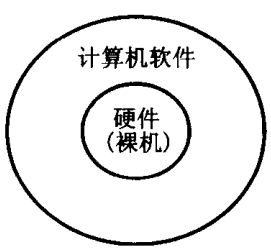


图 1.2.2 计算机系统的两个组成部分

用户提出的使用要求是多方面的，在功能上是非常复杂的。若把这一切都交给硬件完成，不仅在成本上不合算，而且对于用户使用机器也造成了极大的障碍。对用户提出的许多功能，特别是那些复杂而又灵活的功能，可以通过编制程序来实现，这类程序称为计算机系统程序（或称系统软件）。为了方便用户使用计算机，通常为计算机配置各种系统软件去扩充机器的功能。此外，还有大量用于解决用户具体应用的程序（如用于计算、管理、控制等方面的程序）。不论是系统程序，还是应用程序，在处理时都包含它们所需的数据，另外，在软件开发过程中会形成各种必须的文档资料。因此，软件是由程序、数据和在软件开发过程中的文档资料组成的。软件包括以下几部分内容：

- 系统软件，如操作系统、编译程序等。
- 应用软件，如应用程序、软件包等（如数理统计软件包等）。

- 工具软件，如各种诊断程序、检查程序、引导程序等。

裸机是计算机系统的物质基础，没有硬件就不能执行指令和实施最原始、最简单的操作，那么软件也就失去了效用；而若只有硬件，没有配置相应的软件，则计算机就不能发挥它的潜在能力，这些硬件资源也就没有活力。因此，硬件和软件这二者是互相依赖、互相促进的。

现代计算机系统都包含丰富的软件：各种程序设计语言和翻译系统；功能极强的编辑器；方便用户解答各类问题的应用程序；用于开发各种管理软件的数据库系统；负责维护系统正常工作的维护程序和工具软件等。在众多软件中有一个重要的系统软件，那就是操作系统。它负责管理计算机系统中的软、硬资源，负责计算机的工作流程，并将软件和硬件有机地组织在一起，使计算机系统真正体现系统的完整性和可利用性。

2. 操作系统虚拟机

用户最不喜欢裸机这种工作环境，因为裸机上没有任何软件可以帮助他解决问题。为了方便用户使用计算机，通常要为计算机配置各种软件去扩充机器的功能。在裸机上配置操作系统程序后就构成了操作系统虚拟机。操作系统的核心在裸机上运行，而其他系统软件、应用软件和用户程序则在扩充后的机器上运行。操作系统虚拟机如图 1.2.3 所示。

扩充后的虚拟机不仅可以使用原来裸机提供的各种基本硬件指令，而且还可使用操作系统中所增加的许多其他“指令”。这些指令通称为扩充机器的指令系统，又称为操作命令语言。

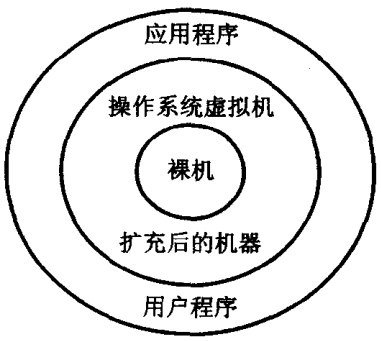


图 1.2.3 计算机系统的组成

操作系统虚拟机为用户提供了一个协助他们解决问题的装置，其功能是通过它提供的命令来体现的，用户也是通过这一组命令和操作系统虚拟机打交道的。

操作系统提供的基本命令形式有两大类，一类是操作命令，另一类是系统功能服务。

(1) 操作命令

这一组命令体现的是人机直接交互的形式，用户可以通过命令，或者通过图形用户界面发出指示，请求操作系统去做某一件事。

(2) 系统功能服务

这一组命令体现的是用户程序与计算机的交互。在用户程序中可以直接使用系统提供的功能服务，请求操作系统的各种服务。

若把操作系统看作一台为用户定义的虚拟机，那么，操作命令语言就给出了虚拟机所能执行的“指令”集合，也刻画了相应的虚拟机的功能。

现代操作系统大多数是多用户、多任务操作系统，无论是 UNIX 系统、Linux 系统、IBM 公司的 OS/2，还是微软公司的 Windows 系统，它们都支持多任务运行的环境。操作系统虚

拟机不仅为计算机系统其他的系统软件、应用软件、工具软件提供了一个功能强大、使用方便的机器，而且，也为多用户、多任务提供了一个具有强大处理能力、具有友好交互界面的运行环境和开发环境。

1.3 计算机软件的核心概念

1.3.1 算法

1. 算法定义与特征

(1) 算法定义

人们在科学实验、生产斗争和社会实践中有大量问题需要求解，如科学计算、数据处理及各种管理问题等。要解决这些问题，首先需要分析所研究的对象，提出对问题的形式化定义和给出求解方法的形式描述。对问题的形式化定义叫做数学模型，而对问题求解方法的形式描述称为算法。

也可以说，算法是解题方法的精确描述。算法具有如下性质：解题算法是一有穷动作序列；动作序列仅有一个初始动作；序列中每一动作仅有一个后续动作；序列终止表示问题得到解决或问题没有解答。

算法的非形式定义如下：

一个算法就是一个有穷规则的集合，其中的规则规定了一个解决某一特定类型问题的运算序列。

(2) 算法的特征

算法是用来描述解题方法的。任何在计算机上处理的问题都需要用算法进行描述，然后在一个可执行指令的计算机上处理，最后得到结果。作为能解决实际问题的算法，必须具备以下特征：

- 有穷性。一个算法在执行有穷步后一定结束，即一个算法所包含的计算步骤是有限的。
- 确定性。算法的每一个步骤必须经过明确的定义，即算法中所要执行的动作应有严格的规定，而没有歧义性。
- 能行性。算法中要执行的运算和操作是最基本的，它们都能够精确地进行。这样，经过算法描述的一系列步骤的确切动作，最后得到正确的结果。
- 输入。算法有零个或多个输入，即算法开始执行之前，要设置好初始值。
- 输出。算法有一个或多个输出，这一输出就是算法的最终结果。该结果是与输入

有关并经过确定的计算步骤后得到的。

2. 算法的表示方法

算法是解题过程的精确描述,那么,用什么描述这一解题过程呢?要描述必须采用某种语言,而语言有多种形式,如自然语言、流程图、伪代码和计算机程序设计语言等。

(1) 自然语言

自然语言是人们日常使用的语言,如汉语、英语、日语等,自然语言的规则是人们所使用语言的规则。用自然语言描述算法是最受欢迎的,因为它直观、通俗易懂、为人们所熟悉。当然,用自然语言来描述解题过程,可能会产生歧义性,容易导致算法执行的不确定性。另外,用自然语言表示的算法要翻译成计算机所能认识、理解的形式是不容易的,要做大量的工作。

(2) 流程图

流程图是用各种几何图形(如方框、菱形框)、流线及文字说明描述计算过程的框图。流程图是描述算法的常用工具。它的优点是直观,能清楚地表达设计者的思路,清楚易懂。流程图表示的算法不依赖于任何具体的计算机和计算机程序设计语言,有利于程序设计工作。

由于程序的结构主要包括顺序结构、选择结构和循环结构,而流程图又能方便地表示这3种结构,所以,用流程图描述的算法过程通过选择合适的程序设计语言后,形成的程序能方便地在计算机上运行。

(3) 伪代码

伪代码是界于自然语言和计算机语言之间的、用文字和符号描述算法的一种语言形式,是描述算法的工具。伪代码具有计算机语言的形式,例如有函数、过程的描述,有输入、输出的描述等。伪代码描述的算法侧重于描述算法应完成的主要功能、应满足的条件,而不注重在计算机上实现的细节,有的伪代码中还可以掺杂自然语言的描述。所以,用伪代码写成的算法描述格式紧凑、易于理解,而且便于向计算机程序设计语言描述的算法过渡。

(4) 计算机程序设计语言

计算机程序设计语言是编写计算机程序所用的语言。用自然语言、流程图和伪代码描述的算法,计算机是不认识的,这样的算法还必须转换为用计算机程序设计语言描述的程序,计算机才能理解和执行。

程序设计语言按级别可分为高级语言和低级语言。关于程序设计语言的进一步讨论,将在1.3.3节中进行。

3. 算法分析

为了解决计算机上的各类应用问题,需要用算法描述解题方法。一个算法必须用一个合适的程序设计语言书写成程序,然后才能在计算机上处理,并得到最后的结果。

这里有一个重要的问题是,用算法描述的解题方法是否有效,即需要对算法的效率进行

分析。算法对应的程序要到一个具体的计算机上去执行，一个给定的算法在一个具体的计算机上执行的时间是多少？它占用的计算机的存储空间又是多少？另外，对一个给定的应用问题，如果可用多种算法进行描述，那么，不同的算法各自所需的计算时间和占用的计算机存储空间的大小又各是多少？

算法分析的描述为：算法分析（analysis of algorithms）是对一个算法所需的计算时间和占用的存储空间作定量的分析。

经过算法分析工作，不仅可以预计所设计的算法在什么样的计算机上运行是有效的，还可以分析最好、最坏、平均情况下算法执行的效果。如果为同一问题设计了不同的算法，经过算法分析后可以对这些不同算法的有效性作出比较和判断。

在算法分析中，一般应考虑的 3 个问题是：算法的时间复杂度；算法的空间复杂度；算法是否便于阅读、修改和测试。

要分析一个算法，首先应确定该算法所需的计算时间。算法的时间复杂度指的是：算法有关操作的次数的度量，用 $T(n)$ 表示。其中，T 是 Time 的第一个字母，n 是问题规模的大小。

例如，一个 n 次循环程序，每次循环做 $a+b=>a$ 的操作，该程序计算的次数为 n；而在累加求和中，n 表示要计算的加数的个数。

在算法的时间复杂度分析中，用 $O(f(n))$ 作为 $T(n)$ 的上界函数。其中，大 O 作为算法性能描述的工具，表示计算时间的上界函数。$f(n)$ 是在分析中确定的某个形式简单的函数，并且是关于正整数 n 的函数，如 $\log n, 2n, n!$ 等，它是独立于机器和语言的函数。

对于 $T(n)$ 和 $f(n)$ 而言，若存在两个正常数 $n0$ 和 C，当 $n \geqslant n0$ 时，有

$$|T(n)| \leqslant C|f(n)|$$

则算法时间复杂度 $T(n)$ 可表示为

$$T(n) = O(f(n))$$

其含义是：一个算法具有 $O(f(n))$ 的计算时间，指的是该算法的计算时间总是小于 $|f(n)|$ 的常数倍。所以可以说，$f(n)$ 是计算时间 $T(n)$ 的一个上界函数，$T(n)$ 的数量级是 $f(n)$。

常用的大 O 表示形式有：

- $O(1)$ 称为常数级
- $O(\log n)$ 称为对数级
- $O(n)$ 称为线性级
- $O(n^2)$ 称为二次多项式级
- $O(2^n)$ 称为指数级
- $O(n!)$ 称为阶乘级

上述表示的计算时间复杂度从上到下越来越复杂。一个算法的计算时间复杂度应尽可

能选用多项式级、线性级及以下的时间复杂度，因为太复杂的算法的实现是非常困难的，且这样的算法也没有实用价值。

算法的空间复杂度是指算法在计算机上的执行过程中所占用的存储空间的大小，用 $S(n)$ 表示，S 是英文单词 Space 的第一个字母，n 为问题规模的大小。算法的空间复杂度与时间复杂度的表示方法类似，为

$$S(n) = O(g(n))$$

1.3.2 数据结构

算法是对各类问题解题方法的描述，对于数值计算和非数值计算而言，都需要处理大量的数据。数据不仅具有属性，而且在大多数情况下互相联系，尤其是在非数值处理领域中更是如此。在程序中，数据的特征被抽象为数据类型和值，而信息及其联系被抽象为数据结构。在计算机领域中，数据结构指的是一类定性数学模型，它是计算机算法设计的基础，在计算科学中占有重要的地位。

1. 数据类型

在计算机中，要处理的数据信息表征为量，它是对客观世界实体的一种抽象。例如，计算学生某科学习成绩平均值的程序要处理的数据是学生的考试成绩、学生总数和平均成绩。它抽取了学生学习成绩这方面的特征，而并不关心其他方面。这个抽象的数据可以表示任何学校、任何一个班级、任一科目的学习成绩的平均值，甚至可以用于某一生产车间中工人产值的平均值。

计算机中使用的量可以分为常量和变量。其中，在计算处理中始终保持值不变的量称为常量，而随时间变化的量称为变量。与量对应的数据存储在计算机存储器中，这些量的值是相应存储字中的内容。那么，如何引用这些值呢？这些值有没有一定的范围呢？为了引用这些值，需对数据引进名的概念，用这个名可以访问相应存储器的字。而且，为了刻画数据的可能取值范围；还引进了类型的概念。例如，C 语言提供的基本数据类型为以下 4 种：

char	字符型
int	整数型
float	单精度浮点数型
double	双精度浮点数型

char、int、float、double 等都是类型的名。其中，类型为 int 的量能取任何整数值，类型为 char 的量能取任何字符值等。类型是值的集合，即类型是可能取值所组成的集合。当然，类型也表征了数据的属性。变量应通过类型说明来规定它的属性，如“int i,j;”说明变量 i、j 为整数型。常量数据从书写形式上就可以了解它的值属于什么类型。

类型概念不仅规定了一个量所能取值的范围，同时也规定了对标定类型的量所能进行的运算，例如，对类型为 int 的量只能进行整型运算。所以，概括起来说，量涉及名、值与类型

3 个方面,而类型则涉及名、值集和运算集 3 个方面。

不同的高级语言所提供的数据类型以及类型定义的方式是不相同的。一般数据类型分为基本数据类型(简单数据类型)和构造数据类型两种。构造数据类型是由基本数据类型按某种方式组合而成的,如数组(array)、指针(pointer)、结构(struct)等。

2. 数据结构的组成

数据结构研究的是数据的组织形式,它涉及 3 个方面的内容:抽象数据结构、内部存储结构和数据结构上所施加的运算。

(1) 抽象数据结构

抽象数据结构又称数据的逻辑结构,它是客体之间联系的抽象。组成数据结构的数据元素是基本数据类型或构造数据类型。数据元素又称为结点,抽象数据结构就是各结点之间的逻辑结构。

抽象数据结构分为线性结构和非线性结构。线性结构有串、栈、队、表等,非线性结构有树、图等。

(2) 内部存储结构

内部存储结构又称数据的物理结构,它是数据逻辑结构的物理实现方式,是依赖于计算机的,即是数据的逻辑结构在计算机的存储器中的排布方式。它一般有两种方式:顺序存储结构和链式存储结构。

① 顺序存储结构。借助数据元素在存储器中的相对位置来表示数据元素的逻辑关系。

② 链式存储结构。借助指针来表示数据元素的逻辑关系。一般情况下,在数据元素中增加了一个或多个指针类型的属性,以表示元素之间的连接关系。

(3) 数据结构上所施加的运算

在数据结构上所施加的运算有以下几种:建立数据结构;清除数据结构;插入数据元素;删除数据元素;更新数据元素;查找数据元素;按序重新排列;统计数据元素的个数;判定某个数据结构是否为空,或者是否已达到最大允许的容量。

1.3.3 程序和程序设计语言

1. 程序

要在计算机系统中求解一个实际问题,就必须将问题的数学模型和算法描述用一种计算机能认识的语言来描述,这就是程序设计语言的指令和语句,计算机执行这一系列指令或语句就能完成预期的任务。

程序是按某种顺序排列的,使计算机能执行某种任务(例如解题、检索数据或对一个系统进行控制等)的指令集合。也可以这样定义:程序是计算机系统中计算任务的处理对象和处理规则的描述。

一个程序具有一个单一的、不可分的结构,它规定了某个数据结构上的一个算法。瑞士

著名的计算机科学家尼可莱·沃思(Nikiklaus Wirth)在1976年曾说道:

程序 = 算法 + 数据结构

程序的基本目标是实现算法和对初始数据进行处理,从而获得所期望的效果。而算法实现过程是由一系列操作所组成的。算法使用一些最基本的操作(如加、减、乘、除、"与"、"或"、"非"、传送、比较),通过对已知条件进行一步一步的加工、变换,最终实现解题目标。这些操作的执行实现了程序应完成的功能,但这只是问题的一个方面。这些操作的执行顺序正确与否也是至关重要的,而操作之间的执行顺序就是控制结构。有3种最基本的控制结构:顺序结构、选择结构、循环结构。

顺序结构是3种结构中最简单的一种,即语句按照书写的顺序来依次执行;选择结构又称为分支结构,它是根据计算表达式的值来判断应选择执行哪一个流程的分支;循环结构则是在一定条件下反复执行一段语句的流程结构。这3种结构就构成了程序局部模块的基本框架。

1966年,Bohm和Jacopini证明了任何复杂的算法都可以用顺序、选择、循环3种结构组合而成。所以,这3种控制结构称为程序的3种基本控制结构。如图1.3.1所示为这3种基本控制结构的一般形式,其中(c)和(d)是循环结构的两种形式,(c)称为直到型循环,(d)称为当型循环。直到型循环是首先进入循环体,然后判断条件是否成立,若不成立再返回执行循环体,否则退出循环。而当型循环是首先判断条件是否成立,若成立则执行循环体,执行后再判断条件是否成立,若成立则继续执行,否则退出循环。

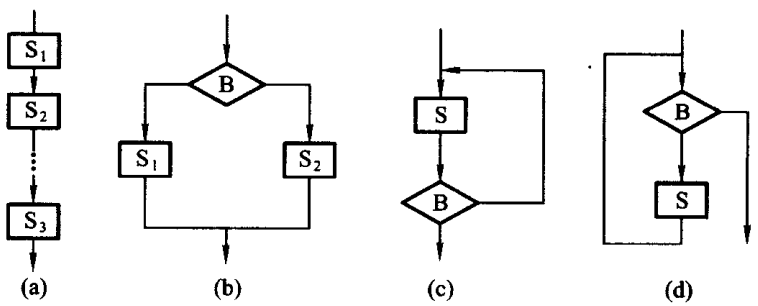


图1.3.1 程序的基本控制结构

每个基本控制结构可以看作一个算法单位,整个算法可以由许多个算法单位组合而成。这样的算法容易理解、容易阅读,称为结构化算法。

算法和数据结构是计算机程序的两个基本的、重要的组成部分。算法是程序的核心,而数据结构是对所要加工数据的抽象和组织,将程序要处理的各种松散的数据按某种要求组成数据结构,以便程序能有效地处理。

2. 程序设计语言

程序设计语言是编写计算机程序所用的语言,程序设计语言按语言的级别可分为低级语言和高级语言。

(1) 机器语言和汇编语言

低级语言包括机器语言和汇编语言。机器语言是计算机直接使用的语言,由机器指令组成。它使用绝对操作码,即直接使用"0"和"1"两个符号来表示。用机器指令写的程序不需翻译可直接为机器所识别。机器语言效率高,但用机器指令编程序对程序员而言是十分困难的。

汇编语言是一种面向机器的程序设计语言,是一种用符号表示的低级语言,通常是围绕特定的计算机或计算机族而专门设计的。与机器语言相比,汇编语言使用助记码代表机器的操作码,用标识符代替地址码和变址码。汇编语言使程序员摆脱了计算机的一些细节问题,从而可以集中精力考虑程序中的内在联系。

(2) 高级语言

高级语言是易为人们所理解的程序语言,它是接近日常用语与数学表示方法的程序设计语言。常用的高级语言有 Basic、Pascal、FORTRAN、COBOL、C、Ada 等。

在进行软件开发时,人们往往采用高级语言来编程。有时也部分采用汇编语言编程,这种情况一般在编制系统程序或为了提高效率时才会出现。现在,已基本没有直接用机器语言编程的情况了。

通用的高级语言是在商业、科学、工程计算等方面能广泛应用的程序设计语言。这些语言有大量的软件库,并为大家所熟悉和接受。其中,历史悠久且最基础的通用高级语言有 Basic、FORTRAN、COBOL 等。还有一类通用语言是结构化的程序设计语言,它直接提供结构化的控制结构,具有很强的过程控制能力和数据结构能力,常用的有 Pascal、C 以及 Ada 语言等。

高级语言及语言处理自 20 世纪 50 年代末出现后发展迅速,现在已有几百种语言存在,按照程序设计语言表达的各种概念和各种结构的不同,可以分为以下几类:

① 过程式程序设计语言

过程式程序设计语言具有过程处理的特点,程序主要包含数据结构和算法描述,表现为过程或函数。程序执行被看作各过程调用的序列。Pascal、FORTRAN 和 C 语言就是过程式的程序设计语言。

② 面向对象程序设计语言

面向对象程序设计语言是以数据为中心、面向对象的一种语言形式,其程序包含定义的各种对象、类,并规定它们之间传递消息的规律。从程序的执行来看,归结为各对象和它们之间以消息传递方式进行的通信。具有面向对象特征的语言有C ++ 、Java 等。

③ 函数式程序设计语言

程序被看作描述输入与输出之间关系的一个数学函数,完全消除状态或变量的概念。Lisp 语言是一种函数式程序设计语言。

(3) 语言翻译

计算机只能执行用机器语言编写的程序,对其他语言编写的程序都必须经过翻译,将它们转换为与之等效的机器语言的程序,才能为计算机所认识和理解。根据语言的类别不同,执行方式的不同,常用翻译工具有汇编程序、编译程序和解释程序 3 种。当前用得比较多的、采用面向对象技术、面向网络编程的 Java 语言有新的工作机制,下面将进行简单介绍。

汇编程序是一种翻译程序,其功能是将用汇编语言编写的程序翻译成机器语言程序。汇编程序的特点是,其指令与翻译后的机器语言指令具有一一对应的关系。

编译程序也是一种翻译程序。它将用面向过程语言编写的源程序作为数据接收,经过翻译转换,产生并输出与源程序等效的目标程序。该目标程序可以是直接转换生成的机器代码程序,也可以是汇编语言描述的程序,若为后者,还需要经过汇编阶段。

解释程序将用高级语言编写的程序作为输入,但并不产生目标程序,而是边解释边执行,这样的翻译程序称为解释程序。在解释程序控制下,按源程序中各语句动态执行的顺序,一次读一个语句,根据该语句的功能,翻译成与该语句对应的机器代码,并立即进行处理。翻译执行完一个语句后,按执行顺序读入下一个语句,直至将全部源程序动态处理完为止。这种逐条语句分析、翻译、执行的方式称为解释执行。目前,流行的 Visual Basic 语言是解释性语言。

(4) Java 语言的工作机制

Java 语言是面向对象的、基于 Internet 的程序设计语言。那么,Java 如何能适应 Internet 环境下网络程序设计的需要呢? 为此,讨论 Java 语言的工作机制。

由于互联网具有异构性,它是由各种不同类型的计算机(如大型机、小型机、集群、PC 机等)、不同类型的终端、不同类型的操作系统等组成的,所以,在 Internet 上运行的应用程序应具有在各种不同的软硬件平台运行的能力,即具有良好的可移植性。

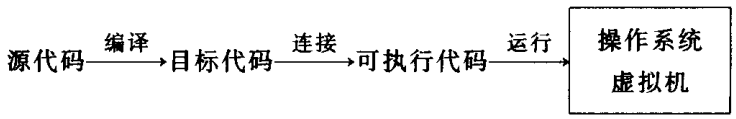
传统的程序设计语言,如 C 或C ++,只能实现源代码级的可移植性,即在一台机器上可运行的 C 或C ++ 程序,要在另一台异构环境上运行,需要将该源程序在新的计算机上重新编译、连接,生成在新环境下可执行的文件。

针对异构的互联网,一个 C 或C ++ 编写的程序要想在环境不同的计算机上运行,需经过多次重新编译、连接工作,这十分麻烦,而且也是大量的、重复性工作,也给应用程序的升级和维护带来了很大的困难。所有这些都在很大程度上限制了网络应用程序的发展和推广。

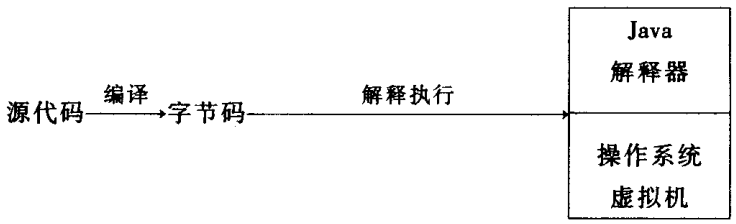
Java 语言为解决上述问题,提供了一个崭新的、有效的方法,它不需要翻译程序,也不是简单的解释,而是采用了如下的工作机制。

首先,用 Java 语言编写好源代码,然后经编译生成一种二进制的中间码,称为字节码,

最后再通过运行于操作系统平台环境上的一种称为 Java 解释器的运行机构来执行编译生成的字节码。这种运行机制与传统的高级语言不同,用图 1.3.2 说明。



(a) 传统语言的运行机制



(b) Java 语言的运行机制

图 1.3.2 传统语言与 Java 语言的运行机制

传统语言的可移植性差,是因为它们的可执行程序直接作用于不同的操作系统平台,所以需要能适应不同平台环境的可执行程序;而 Java 的字节码是运行在操作系统之上的 Java 解释器上的,虽然不同的平台环境需要有各自相应的解释器,但任何一个平台上的解释器,对于一段 Java 程序的字节码来说却是完全相同的,也就是说,Java 的运行机制利用解释器来隐藏网络上平台环境的差异性,在配置了针对不同平台环境的解释器之后,它们对 Java 字节码呈现完全相同的面貌。

1.3.4 计算机软件技术概述

计算机的应用非常广泛,涉及工业、农业、科学研究、教育、医疗、商业、娱乐、国防、行政管理、直至家庭和个人等各个领域。在这些领域中的应用一般可分为科学研究、数据处理、过程控制和人工智能等几大类。为了使计算机系统能有效地解决各类应用问题,对应用问题需要建立模型、提出解题的方法、进行软件系统的开发、并在一个人机交互极为方便的计算环境中运行该软件,最终得到用户满意的结果。

面向计算机应用的各类活动都与计算机系统密切相关。计算机系统分为硬件和软件两部分,与应用开发关系最为密切的是软件部分。软件着重研究的是解题算法、程序及数据的表示方法、控制程序执行的机制、语言翻译、管理计算机系统资源与操作过程的各种程序及实现技术。

对应用问题的求解,首先必须能清晰地描述问题。由于应用问题的复杂性可能超出人们的认识能力,因此,要采用有效的方法将复杂的问题分解成人们能理解的、能清晰表述的许多简单问题。在现代软件开发中采用了科学的方法,如结构化方法和面向对象方法,提出

了软件工程的概念，以工程的手段、技术和方法来开发与维护软件。另外，对每一个问题应明确其被处理的对象和处理过程的描述，即确定算法与数据结构。

计算机软件技术涉及软件及实现技术、软件开发方法与技术。

软件及实现技术包括：程序设计技术与程序设计语言；编译技术；操作系统及实用程序；数据库技术；软件工具及实现技术等。

软件开发方法与技术包括数据结构与算法，软件工程以及程序设计方法。

计算机软件自 20 世纪 50 年代到目前的发展过程中，在以下 4 个方面有重大进展：程序设计语言及语言处理；操作系统；数据设置与处理；软件工具、技术和规定。

程序设计语言是编写计算机程序的语言。为了实现从用户使用的语言形式到计算机能理解的机器语言的转换，还需要语言翻译（编译）系统。在研制编译系统时，重点是提供快速翻译和产生高效目标代码的技术。操作系统是控制计算机工作流程、处理并发活动、管理计算机软硬件资源的系统软件，并提供方便的用户界面，使用户能灵活、方便地使用计算机系统。操作系统的实现技术涉及多道程序设计技术、分时技术、资源的分配与调度等极为丰富的内容。数据设置与处理是指用来处理大量数据的技术与工具，它涉及描述文件与单个记录的技术与工具，以及对数据进行排序、分类、查询、计算的技术与工具。目前，在这方面发展成熟的有数据库技术。数据库是为了解决对大量数据的有效管理、实现数据共享的问题而产生的。数据库系统实现了应用程序与数据的物理分离，还能实现它们的逻辑分离；提供数据间联系的描述方法与使用方法；能有效地控制数据冗余和实现数据的共享。第 4 方面是软件工具。研制软件工具的目的是使软件开发“自动化”。软件工具按功能可划分为说明工具系统、设计工具系统、实现工具系统、维护工具系统和管理工具系统。例如，结构化编辑器、源程序调试器都是软件工具。另外，编译程序也可以说是一种翻译工具。

1.4 软件工程与软件工程模型

1.4.1 软件与软件开发的特点

软件是程序、数据及相关文档的集合，是计算机系统的组成部分，称为软部件，或称逻辑部件。而计算机系统的另一部分——硬件，则是一些机械的、磁性的、电子的装置或物理部件。软件和硬件相比有以下特点：

- 软件是逻辑部件，缺乏“可见性”。
- 软件出现的错误不是因为使用时间过长，而是由于在开发阶段引入的、但在测试阶段没有检测出来的错误。

软件的规模大小、复杂程度决定了软件开发的难度。大型软件系统和小型软件系统相比较，二者有本质的区别。对于一个软件而言，它的程序复杂性将随着程序规模的增加而呈指数级上升趋势。

在小型软件系统中，程序规模较小，程序长度不太长，一个人或几个人就能管理和控制其复杂性，因此可以用较简单的方法完成程序的编制并达到预期的效果。但随着计算机应用的日益广泛，软件系统的规模越来越大，它对应的程序也就具有规模大、复杂度高、研制周期长、可靠性低等特点。这样的软件系统往往是不可维护的，而且许多程序中的错误难以纠正。人们在软件开发过程中，对复杂度的认识程度是有限的，所以，必须采用科学的软件开发方法，采用抽象、分解等科学方法降低复杂度，以工程的方法管理和控制软件开发的各个阶段，以保证大型软件系统的开发具有正确性、易维护性、可读性和可重用性。

1.4.2 软件工程

软件开发是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。软件工程是指按工程化的原则和方法来组织软件开发的工作，它力图自始至终地掌握软件开发的整个过程，并将这个过程用标准化的文档规范、记录下来。软件工程正是从管理和技术两个方面，研究如何更好地开发和维护计算机软件的一门学科。

所以，软件工程是指导计算机软件开发和维护的工程科学，采用工程的概念、原理、技术和方法来开发与维护软件。在开发与维护软件过程中，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，这就是软件工程。它用计算机科学、数学及管理科学等原理，借鉴传统工程的原则、方法，创建软件，以达到提高质量、降低成本的目的。其中，计算机科学、数学用于构造模型与算法，工程科学用于制造规范、评估成本及确定权衡，管理科学用于计划、资源质量、成本等管理。

由于软件和软件开发具有不可见性、抽象性等特点，所以需要用工程的方法进行开发，而且，还必须采用若干原则。这些原则包括分解（降低问题的复杂度）；制定计划（分阶段实施）和制定规范（把握质量）等，具体描述如下：

（1）划分生命周期，制定阶段计划

软件具有定义、开发、使用、维护直至废弃的一个生命过程。将软件生命周期划分成若干个阶段，每个阶段制定出切实可行的计划，然后严格地按计划对软件的开发和维护进行管理。

（2）进行阶段评审

与一般的工程项目一样，软件开发要严格按计划管理，在生命周期的每个阶段进行严格的评审，以尽早发现在软件开发过程中的错误和问题。

（3）制定规范，实行严格的产品控制

① 在软件工程的每一阶段要编制完整、精确的文档。在阶段评审后应提交各阶段

文档。

② 当用户提出要改变需求时，为了保证原文档资料的一致性，必须按严格的规程进行评审，获得批准后才能实施修改。

③ 为了保证软件产品的审查，应根据软件开发项目的目标及完成期限，规定开发者的责任和产品应达到的标准，使开发的结果能够清楚地被审查。

1.4.3 软件过程

软件具有生命周期，为了保证软件的质量，应对软件的这一过程划分为一系列阶段，对每个任务阶段能明确所运用的方法、需使用的资源（包括人力、时间、计算机系统、软件工具等）、应形成的文档资料、需要采用的管理措施以及任务阶段的标志性成果。

软件生命周期可以划分为4个时期，每个时期又可分为若干阶段。这4个时期为：软件计划、需求分析、软件开发、软件维护。在软件计划时期又分为问题定义和可行性分析两个阶段。软件开发时期又分为概要设计、详细设计、软件编码和软件测试阶段。每个阶段应完成的基本任务和产生的文档如表1.4.1所示。

表1.4.1 软件生命周期各阶段的任务

时 期	阶 段	任 务	文 档
软件计划	问题定义	理解用户要求，划清工作范围	计划任务书
	可行性分析	可行性方案及代价	
需求分析	需求分析	软件系统的目标及应完成的工作	需求规格说明书
软件开发	概要设计	系统的逻辑设计	软件概要设计说明书
	详细设计	系统的模块设计	软件详细设计说明书
	软件编码	编写程序代码	程序、数据、详细注释
	软件测试	单元测试、综合测试	测试后的软件、测试大纲、测试方案与结果
软件维护	软件维护	运行和维护	维护后的软件

（1）软件计划

软件生命周期的第一个阶段是进行调研和分析，搞清楚用户想干什么，不想干什么，以确定工作范围。通过调查后进行高度地抽象，概括出“用户想要解决的问题是什么”。

在上述工作的基础上进行可行性分析。这个阶段要回答的问题是“用户要解决的问题是否能解决”，即可行性问题。本阶段具体的工作是：分析所需研制的软件系统是否具备必要的资源和其他条件，包括技术上、经济上的可能性以及社会因素的影响。经过以上工作

后,确定该项目的可行性。

(2) 需求分析

经过问题定义、可行性分析后,进入需求分析阶段。这一阶段要解决"做什么的问题"。

可行性分析阶段对用户提出的问题进行了主要的、大的方面的分析,有许多细节是被忽略的。而在需求分析阶段就要考虑所有的细节问题,以确定最终的目标系统必须做哪些工作,形成目标系统完整的、准确的要求。

该阶段最后提交规格说明书。这份文档准确地说明该系统的目标以及对系统的要求。

(3) 软件开发

软件开发包括概要设计、详细设计、软件编码和软件测试4个阶段。

概要设计又称为总体设计、逻辑设计。该阶段要回答"怎样实现目标系统"的问题。首先,应考虑为实现目标系统的可能方案,并选择一个最佳方案。确定方案后应完成系统的总体设计,即确定系统的模块结构,给出模块的相互调用关系,并产生概要设计说明书。

详细设计阶段回答"应该怎样具体实现目标系统"的问题。在概要设计的基础上,要给出模块的功能说明和实现细节,包括模块的数据结构和所需的算法,最后,产生详细设计说明书。

在详细设计基础上进入软件编码阶段。程序员根据系统的要求和开发环境,选用合适的高级程序设计语言或部分选用汇编程序设计语言(在编写系统软件时需要适当地选用)编写程序代码。

软件测试分为单元测试和综合测试两个阶段。单元测试是对每一个编制好的模块进行测试,发现和排除程序中的错误。综合测试是通过各种类型的测试检查软件是否达到预期的要求。综合测试中主要有集成测试和验收测试。集成测试是将软件系统中的所有模块装配在一起进行测试。验收测试是按照规格说明书的规定,由用户(或有用户参加)对目标系统进行验收。

软件维护阶段是长期的过程,因为,经过测试的软件还可能有错;用户的要求还会发生变化;软件运行的环境也可能变化,在上述各种情况发生时,都需要进行软件的维护。因此,交付使用的软件仍然需要继续排错、修改和扩充,这就是软件维护。

1.4.4 瀑布模型

瀑布模型是一个生命周期模型。该模型依照软件生命周期的划分,描述软件开发阶段的任务、各阶段执行顺序和信息反馈的路径。图1.4.1描述了软件开发过程的瀑布模型,图中说明瀑布模型按软件生命周期分为6个阶段:软件计划(包括问题定义和可行性分析阶段)、需求分析、软件设计(包括软件概要设计和详细设计阶段)、软件编码、软件测试和软件维护,任何软件的开发都要顺序地经历这6个阶段。

在软件开发过程中,这6个阶段是顺序进行的,前一阶段的工作完成后,下一阶段的工

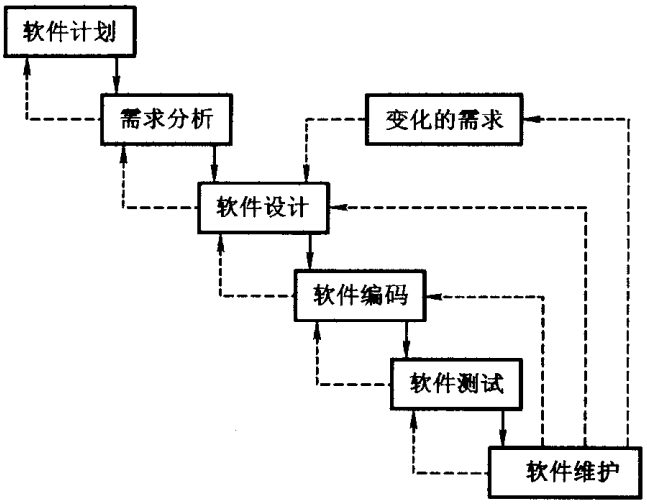


图 1.4.1 瀑布模型

作才能开始;前一阶段产生的文档是下一阶段工作的依据。

瀑布模型适合在软件需求比较明确、开发技术比较成熟的场合下使用,这种软件开发模型是软件工程中应用得最广泛的过程模型。

1.5 软件开发方法与技术

1.5.1 结构化方法的核心问题

结构化方法(Structured Methodology)是计算学科的一种典型的系统开发方法,它采用了系统科学的思想方法,从层次的角度,自顶向下地分析和设计系统。结构化方法包括结构化分析(Structured Analysis,SA)、结构化设计(Structured Design,SD)以及结构化程序设计(Structured Program Design,SP)3 部分。

结构化方法是由结构化程序设计语言发展而来的。在计算机程序设计语言发展过程中,最初的程序设计是手工式的设计方法,到了 20 世纪 60 年代,计算机软硬件技术得到了迅速的发展,向传统的手工式程序设计方法提出了挑战。1960 年计算机学者发表了重要的论文,给出了任何程序的逻辑结构都可以用 3 种最基本的结构(即顺序结构、选择结构和循环结构)来表示。后来,又有学者发表了《GOTO 语句是有害的》和《带有 GOTO 语句的结构化程序设计》论文,使人们开始重视良好的程序结构,逐步形成结构化程序设计方法。

在结构化程序设计的基础上，又发展形成了结构化设计和结构化分析方法。

用结构化方法进行软件开发的过程经过如下步骤：从系统需求分析开始，运用结构化分析方法建立环境模型（即用户要解决的问题是什么，以及要达到的目标、功能和环境）；需求分析完成后需采用结构化设计方法进行系统设计，确定系统的功能模型；最后，进入软件开发的实现阶段，这时，运用结构化程序设计方法确定用户的实现模型，完成目标系统的编码和调试工作。该软件开发过程可用图 1.5.1 说明。

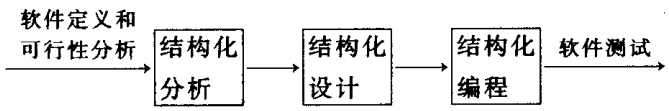


图 1.5.1 用结构化方法进行软件开发的过程

1.5.2 结构化设计

在进行软件系统的设计时，可采用结构化设计方法。结构化设计方法遵循抽象、模块化、逐步求精等基本设计原理。

人们在认识复杂事物的过程中，经常采用的方法是抽象。在现实世界中，许多复杂事物存在着复杂的特征和状态，但可以先抽取这些事物最本质的、共性的特征，而暂时忽略其他的细节，这就是抽象。利用抽象方法，通过不同层次的抽象，可以将一个复杂的事物逐步分解。对软件系统而言，就是模块化分解。模块化是把程序划分成独立命名的、且可以独立访问的模块。每个模块完成一个子功能，将这些模块组合起来就构成了一个整体，可以完成指定的功能。这样可以提高程序的可读性、可靠性和可维护性，以满足用户的需求。

在模块化分解过程中，问题一步一步细化，实际上就是采用了逐步求精的原则。结构化设计包括以下方面：自顶向下逐步求精的设计方法；分而治之的分解技术；模块化的组织形式。

(1) 逐步求精的设计方法

在进行程序设计时，要将一个复杂的问题最终变为一个书写正确的程序，最好的方式是逐步展开。而采用逐步展开的方式研制程序的策略应是自顶向下的策略。把要解决的复杂问题分解成若干个子问题，然后分别独立解决这些子问题，再将各个子问题的解以某种方式连结起来，这就是原始问题的正确解答。如果子问题仍较复杂，则又可以把这些子问题看作新的要解决的问题，而对它们进行分解。这样不断地分解，最终使得子问题简单得可用若干行程序设计语言来表达，于是整个问题便得到了解决。

Wirth 曾对逐步细化的方法作过如下说明："我们对付复杂问题的最重要的办法是抽象，因此，对一个复杂的问题不应该立刻用计算机指令、数学和逻辑符号来表示，而应该用较自然的抽象语句来表示，从而得出抽象程序。抽象程序对抽象的数据进行某些特定的运算

并用某些合适的证号(可能是自然语言)来表示。对抽象程序作进一步的分解,并进入下一个抽象层次。这样的精细化过程一直进行下去,直到程序能被计算机接受为止。这样的程序可能是用某种高级语言或机器指令书写的"。

使用自顶向下、逐步细化的设计方法符合人们解决复杂问题的一般规律,是人的智力能接受的方法,可以显著地提高程序设计的效率。在这样的设计方法指导下,实现了先全局后局部,先整体后细节,先抽象后具体的逐步细化的过程。这样研制出来的程序具有结构清晰的特点,有很高的可读性和可维护性。

(2) 模块化组织结构形式

程序设计是把复杂问题的求解转换为计算机能执行的简单操作的过程。对复杂问题进行求解,一种有效的手段是把它分解成若干小的子问题。如果子问题还比较复杂,则将其继续分解成若干个子问题,直到所有子问题的复杂性都能被控制。模块化程序设计就是把一个大程序按一个人能理解的规模进行分解的一种方法。模块化指的是把一个程序按功能分解成若干彼此具有一定独立性同时也具有一定联系的组成部分,这些组成部分称为模块。每个程序由一个或多个模块组成。

在按功能划分模块时,根据所需完成的功能的明显差异,划分组成系统的各个模块。一个模块内部包括所需完成的单一任务的所有成分。模块内部的联系应该紧密,而各模块之间的联系应尽量少。模块内部的实现细节在模块外部是不可见的,但每个模块都提供与外界联系的接口。

采用模块化结构形式组织时,一个系统被划分为若干模块。因此,在结构化设计方法下,要求在设计程序系统时按层次结构组织模块。其中,最上层的模块是对系统的整体功能的抽象,它指出系统应该"做什么",而不涉及"如何做"的细节;下层模块是对上层所分解的几个"做什么"进行进一步的描述。这样逐层向下分解,直到得到便于实现的单一功能的模块为止。在最后一层模块中,才对"如何做"进行精确的描述。图 1.5.2 给出了一个程序的模块结构。程序 P 有 3 个子模块 P_1、P_2 和 P_3,它们又分别分解为 P_{11}、P_{12}、P_{21}、P_{22}、P_{31}、P_{32}。各模块之间的关系如图 1.5.2 所示。在模块结构中还规定某级模块可为上一级或同一级模块所调用,而决不能被下一级模块所调用。

对大型程序设计而言,模块化是一种必然的趋势。它的优点很多,如可使程序易读、易写、易调试、易维护、易修改等,具体说明如下:

- 模块间的接口关系简单,且每个模块的复杂度都是人的智力能控制的。这种程序的可读性及可理解性比较好。
- 各模块的功能单一,当要修改某一功能时,只涉及一个模块,不会影响到全局。这种程序的可修改性和可维护性较好。
- 人们可以单独地验证一个模块的正确性,即脱离程序的上、下文也能静态(人工)或动态(上机)验证该模块的正确性,所以程序的可验证性较好。

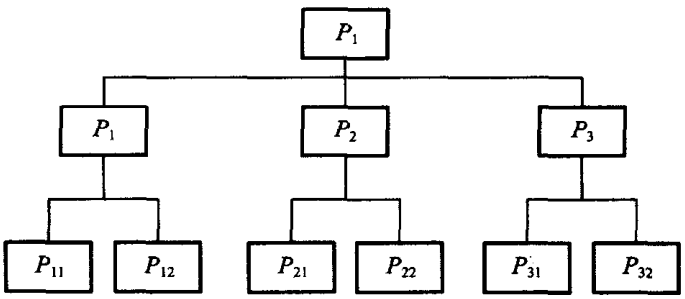


图 1.5.2 程序的模块结构

过程(包括子程序和函数)是程序设计中实现模块化的有力工具。

1.5.3 结构化实现

软件开发的实现阶段包括编码和测试。结构化实现就是按照一定的原则编制正确易懂的程序的过程。

作为软件设计后的一个阶段,软件编码是设计的自然结果。在这一阶段,选用合适的程序设计语言、良好的编码风格,对保证程序的可读性、可靠性、可测试性和可维护性将产生重要的作用。

选用好的程序设计语言实现编码,也称为结构化程序设计,其目的是要使程序具有一个合理的结果,以保证程序的正确性。

结构化程序设计的主要特征有以下几点:

- 结构化程序设计要求把程序的结构限制为顺序、选择和循环3种基本结构,即以3种基本结构的组合来描述程序。
- 有限制地使用 GOTO 语句,在非用不可的情况下,也要十分谨慎,并且只限于在一个结构内部跳转,不允许从一个结构跳到另一个结构。这样可缩小程序的静态结构与动态执行过程之间的差异,使人们能正确理解程序的功能。
- 以控制结构为单位,只有一个入口,一个出口,各单位之间接口简单,每个单位也容易理解。
- 采用结构化程序设计语言来书写结构化程序,并采用一定的书写格式以提高程序结构的清晰性和增进程序的易读性。
- 结构化程序设计还应注意程序设计的风格,例如,采用表达信息含义的易于理解的标识符名,控制模块的大小等。

1.5.4 结构化方法的优点及问题

结构化方法采用了分解和抽象的手段，依据逐步求精的思想设计方法，形成层次的模块化结构形式，从而有效地将一个较复杂的程序系统设计成易于控制和处理的子任务，这些子任务都是可独立编程的子任务模块。每个子程序都有一个清晰的界面和接口，使整个软件系统易于实现和维护。

结构化方法具有许多优点，从20世纪70年代末至今，对于开发大型软件发挥了很大的作用。但随着计算机技术、软件技术的飞速发展，计算机应用的日益普及，要研制的系统愈来愈复杂，它的不足之处也暴露出来了：1）因为结构化方法是一种面向过程处理的设计方法，它把数据和过程分离为相互独立的实体，其程序结构是由数据结构加算法构成的，程序员在编程时必须时刻考虑该算法是在什么样的结构上进行操作，对于不同的数据格式即使做同样的处理或对相同的数据格式做不同的处理都要编制不同的程序，因此结构化程序的可重用性不好；2）另外，由于数据和过程相分离，每一个过程在那些数据结构上的操作必须自始至终没有错误。要使数据与程序始终保持相容，已经成为程序员的一个沉重负担。由于这种分离，还造成对系统维护的困难，因为只要对某一数据结构作修改，就要改动这个程序以及相应的其他模块；3）另外，随着软件开发规模的扩大和更新、扩展的加速，仅仅使用结构化的方法已经不能够驾驭软件开发的整个过程。正是在这种种新需求的呼唤之下，面向对象的设计方法应运而生。

面向对象方法，相对于以前的结构化方法，代表了一种全新的思维模式，这种全新的思维模式能够方便、有效地实现软件扩展、软件管理和软件复用，使高效率、高质量地开发、维护和升级大型软件成为可能，从而为软件开发技术拓展了一片新天地。

1.5.5 面向对象方法的产生及要点

1. 面向对象方法的产生

面向对象方法的基本思想是，尽可能按人类的思维方式，从现实世界的客观现状出发去考虑软件开发的方法。在这一思想方法的指导下，使开发软件的方法与过程尽可能地接近解决实际问题的方法和过程，使描述实际问题的问题空间与计算机系统的解空间在结构上尽可能一致。

所以，面向对象方法是以面向对象思想为指导进行系统开发的一类方法的总称。这种方法以对象为中心，以类和继承为构造机制来抽象现实世界，通过建立模型，最终构建相应的软件系统。

与结构化方法一样，面向对象方法也起源于面向对象程序设计语言。面向对象程序设计语言早在20世纪60年代后期就出现了。1967年挪威科学家研制的Simula语言，首次引入面向对象概念。随后，1980年提出了Smalltalt-80语言，标志着面向对象程序设计语言进

入了实用阶段。20 世纪 80 年代,相继出现了一大批实用的面向对象语言,如 Lisp、Object Pascal、C ++ 等,其中的C ++ 语言大大推动了面向对象方法与技术的普及与发展。

随着面向对象语言的发展,面向对象技术很快被用到软件开发过程中的系统分析和系统设计中来。20 世纪 90 年代,面向对象分析(Object-Oriented Analysis,OOA)和面向对象设计(Object-Oriented Design,OOD)开始成熟,出现了许多面向对象的方法和技术。

2. 面向对象方法涉及的要点

计算机处理的各类问题,包括数值计算和非数值计算问题,不论哪一类问题最终都归结为对数据的处理。数据和处理是密切相关、密不可分的,把数据和处理人为地分离为两部分,一定会增加软件开发的难度。

结构化方法就存在上述的问题。结构化设计是面向过程的设计方法,它把数据和处理分离为相互独立的部分。当程序规模不大时,用这种方法处理是有效的、成功的。但当程序规模不断扩大时,就越来越暴露出这种方法存在的问题。

面向对象方法是以数据为中心,将数据和处理相结合的一种方法。面向对象方法把对象看作是一个由数据及可以施加的操作构成的统一体。对象与数据有着本质的区别。传统的数据是被动的,它等待着外界对它施加操作。而对象是处理的主体,要想使对象实施某一操作,必须发消息给对象,请求对象主动地执行该操作,外界是不能直接对对象施加操作的。

在对软件系统进行分析、设计时,结构化方法是对软件系统进行功能分析,而面向对象方法是用一种新的思维方法,进行对象分解,即以数据为中心认识问题、分解问题、解决问题。所以,面向对象方法的实质是从系统的组成上进行分解,而不是从功能上、或是从处理问题的算法上来考虑。

面向对象方法涉及的要点如下:

- 客观世界是由对象组成,任何事物都是对象,复杂的对象可以由比较简单的对象以某种方式组成。
- 所有对象划分成各种对象类,每个对象类都定义了一组数据和一组方法(方法就是对象能执行的操作)。
- 类可以派生,父类(或称基类)可以派生子类(或称派生类)。按照子类和父类的关系,可以将若干对象类组成一个层次结构的系统。一般而言,处于下一层上的对象可以自动继承位于上一层次的对象的属性和行为。
- 对象之间只能通过消息传递的方式相互联系。

1.5.6 面向对象的基本概念

1. 对象和类

对象的概念是面向对象方法中的核心概念。以面向对象的观点看来,所有面向对象的程序都是由对象组成的。这些对象是自治的,同时它们又可以互相通信、协调配合,共同完

成整个程序的任务和功能。

对象是现实世界中某个具体的物理实体在计算机逻辑中的映射和体现。例如,汽车是一个实体,有各种汽车零件,并具有实在的功能。在面向对象方法中,这样一个实体可以表示为一个计算机可理解的、可进行某些操作的、并具有一定属性和行为的对象。

类是面向对象方法中另一个重要的概念,是同种对象的集合和抽象。例如,汽车可以分为卡车、小轿车、越野车、警车等。这些实体在面向对象方法中将被映射成不同的对象。但这些代表不同实体的对象之间存在着很多实质相同的特点,如都能开、停、倒退等。为了处理问题的需要,在面向对象方法中,人们抽取同种对象的公共属性和特点,以类的概念来表示。

类是一个抽象的数据类型,它是所有具有一定共性的对象的抽象。而属于类的某一个对象则称为类的一个实例,是类的一次实例化的结果。面向对象方法中的类和对象的关系,在现实世界中也大量存在。例如,定义了一个类——"人",那么,李甲、王乙、张丙就是一个个实例。

2. 对象的属性

对象是计算机软件的一个结构,具有状态、行为、标志等主要属性。

对象的状态又称为对象的静态属性,包括对象内部所包含的信息,也就是变量。每个对象都具有自己专有的内部变量。这些变量的值说明了对象所处的状态。当对象实施了某种操作后,其状态会发生变化,这一变化体现在它的属性变量的内容的改变上。

对象的行为又称为对象的操作,主要表述对象的动态属性。操作的作用是设置或改变对象的状态。比如,汽车可以有启动、停止、倒退等行为或操作。在对象上实施的操作都是基于对象内部的变量,并企图改变这些变量的值。在计算机内部,对象的状态用变量来表示,而对象的行为用方法来表示,方法实际上类似于面向过程程序中的函数。对象的行为或操作就定义在方法的内部。

对象的方法一方面把对象的内部变量包裹、封装、保护起来,使得只有对象自己的方法才能操作这些内部变量;另一方面,对象的方法还是对象与外部环境和其他对象交互、通信的接口,对象之外的其他对象可以通过这个接口来调用对象的方法,操纵对象的行为和改变对象的状态。

仅仅使用状态和行为并不能完全区分不同的对象,如两辆由同一个汽车制造厂生产的、同一型号的汽车就很难区分。为此,在对象的属性中引入了标志,以区分不同的对象。每一个对象都有仅属于它自己的惟一的标志。该标志量在全球范围内通用而不会重复,保证了对象在不同的环境下被重复使用,有利于提高开发的效率和质量。

综上所述,面向对象方法中的对象具有状态、行为、标志3个基本属性,在计算机实现中分别用对象的变量、方法和对象名来表示。

1.5.7 面向对象的软件开发过程

1. 面向对象的软件开发过程

面向对象的软件开发过程包括面向对象分析(OOA)、面向对象设计(OOD)和面向对象编程(OOP)。面向对象的开发方法是构筑一条从现实世界开始,到达计算机上可正确执行的软件系统为止的一条通路。

在1.4.3节中已讨论了软件生命周期所经历的过程。针对这一软件过程,面向对象方法和结构化方法所采用的方法不同,但经历的阶段和过程是一样的。这种情况可用图1.5.3说明。

图1.5.3中说明结构化软件开发方法,通过对现实世界的实际应用问题的调查和分析(即经过问题定义和可行性分析)后进入结构化分析阶段,然后经过结构化设计(包括概要设计和详细设计)、结构化编程,最后到达存储程序式计算机的过程世界。

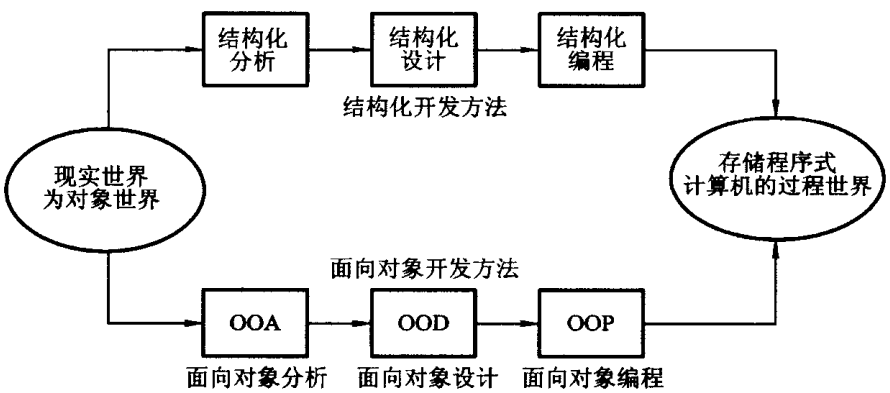


图1.5.3 面向对象开发方法与结构化开发方法的比较

而面向对象开发方法同样通过对现实世界的实际应用问题的调查和分析后,在经过面向对象分析(OOA)、面向对象设计(OOD)和面向对象编程(OOP)后,到达存储程序式计算机的过程世界。

下面简单介绍面向对象软件开发方法的各个过程,包括它们的目的和方法。

2. 面向对象分析

面向对象分析阶段包括需求分析和需求模型化两个步骤,它的主要作用是明确用户的需求,并用标准化的面向对象模型来规范地表述这一需求。在面向对象的分析中,通常要形成3种模型,它们是对象模型、动态模型和功能模型。分析阶段完成后,会生成一系列的文档,包括对任务完整的、明确的定义、软件的功能说明、初步的用户手册等。分析阶段的工作应该由用户和开发人员协作完成。

(1) 需求分析

在面向对象开发过程中,需求分析的主要任务是明确用户的需求,包括对用户需求的全面理解和分析,明确所要开发的软件系统的工作范围、可行性研究,以及制订资源、进度预算等。

在开发过程的初期,用户很可能对其自身的需求表述得不清楚、或者不完全,需求分析的任务就是把这些模糊的概念具体化,并在用户和开发人员之间达成共识。在这个过程中,开发人员需要向用户虚心请教,因为用户对需要解决的问题的具体业务比较清楚。另一方面,开发人员应注意到用户的需求不一定全都是合理的,需要开发人员根据技术、资金、进度的可行性和效益对这些需求进行分析和筛选,并与用户协商,以形成一个合理的方案。

经验证明,需求分析的过程是一个复杂、烦琐、甚至艰难的过程,通常需要用户和开发人员反复地讨论、协商和修改,才能达成最后的一致。同时需求分析又是整个软件开发过程中关键而又重要的一个步骤,如果需求分析的工作做得不够彻底,或者不够明确和完善,那么隐藏起来的错误在后期的工作中会形成很大的隐患和障碍,给整个开发工作带来困难。

(2) 建模

需求分析阶段明确了用户对即将开发的软件系统的具体要求之后,就要转入需求模型化的步骤,将这些需求以标准化模型的形式规范地表述出来,即将用户和开发人员脑子里形成的需求以准确的文字、图表等形式表述出来,形成双方都认可的文件。

在传统的面向过程的开发方法中,这个步骤较多是借助于结构化分析方法,用数据流图和数据字典等工具来完成的。这种分析方法的优点是可以帮助开发人员了解和掌握系统中数据流的运动情况,对软件系统的各种工作状态和这些状态之间的切换有清晰的认识和控制,为后期工作的顺利完成铺平道路。但是这种分析方法的缺点是过于烦琐,不够灵活,一旦因某种原因需要改变需求时,很多原有的工作不能得到继承,造成各方面的浪费。

面向对象的软件开发过程所采用的需求分析方法最终是要完成建模的工作,即要抽取存在于用户需求中的各对象实体;分析、明确这些对象实体的静态数据属性和动态操作属性,以及对象之间的相互关系;更重要的是,要形成由多个对象组成的系统的整体功能和状态(包括各种状态变迁、对象在这些变迁中起到的作用、对象在整个系统中的位置)等。需求模型化方法是面向对象的分析中常用的方法,这种方法通过对需要解决的实际问题建立模型来抽取、描述对象实体,将用户的需求准确地表达出来。比较常用的需求模型有以下几种。

① 对象模型。

对象模型描述所要开发的应用中存在的各个对象实体的数据属性、可能处于的各种状态,以及可能的继承、集合或其他相互关系等,它代表了系统的抽象数据结构。

② 动态行为模型。

动态行为模型负责描述应用及其各个对象实体的生命周期过程,定义可能的系统事件

和各实体对各种事件应有的响应，以及其他一些系统的动态特征。动态行为模型描述了系统的控制结构。

③ 用户界面模型。

用户界面模型描述所要开发的应用程序的用户界面，包括外观和各种具体的功能操作，用户界面模型形成的同时，需求分析的另一个文档——初步的用户手册也应同时生成。

3. 面向对象设计

设计阶段的主要任务是把分析阶段生成的对象实体模型转换成面向对象的类的描述，包括类的属性的详细描述、类的行为操作和类间相互关系的准确表达等。这个阶段包括概要设计和详细设计两个步骤。

(1) 概要设计

概要设计的主要任务是用严格表达方法创建若干对象类，这些对象类应与分析阶段产生的对象实体模型相对应。一般说来，对象类可以直接采用对象实体的属性作为自己的属性，也可以把对象实体模型间的相互关系，如互操作、继承等，在对象类中体现出来。

概要设计应该明确整个应用软件的结构框架和外部接口，如输入、输出操作；同时，分析阶段产生的动态行为模型在概要设计生成的对象类中也应该有所体现。当然，概要设计并不需要把对象实体模型的每个方面都严格地表述出来，有些部分，如一些补充性的内容，可以放到后面的实现阶段再细化完成。

另外，在概要设计中如果出现需求模型二义性等问题，还需要重新返回需求分析的步骤，进一步明确有关内容，了解用户的确切需要。

(2) 详细设计

详细设计的主要任务是对概要设计所得的对象类的表达做进一步的细化分析、设计和验证。细化设计包括对类静态数据属性的确定，对类方法(即操作)的参数、返回值、功能和功能的实现的明确规定等。细化验证主要指对各对象类公式间的相容性和一致性的验证，对各个类、类内成员的访问权限的严格合理性的验证，也包括验证对象类的功能是否符合用户的需求。

在经过详细设计明确了各对象类的功能和组成时，还需考虑的一个很重要的工作是充分利用已存在的、可获得的对象类或部件。使用已存在的，并已验证为正确有效的对象类或部件可以大大提高开发效率、可靠性，降低开发成本。这些可利用的对象类或部件可以是以前开发工作的结果，也可以是网络上免费下载或有关软件厂商购买到的产品，还可以是由同一开发小组的其他开发人员设计实现供大家共享的模块。如果暂时没有现成可以引用的对象类或部件，在详细设计中，还可以分析所要开发的软件系统中哪些类或哪些功能是可以重用的，把这些可重用的部件交给专人优先开发。这样，不但开发人员的工作负担大大减轻，而且保证了质量和效率，提高了软件开发过程的标准化程度。在比较大型的开发项目中，可以设置专人专门负责管理所有的可重用资源，将这些资源组织成类库或其他的可重用结构，

这些资源对整个开发任务将是非常关键的。

在详细设计的细化分析、设计和验证过程中,如果发现概要设计中有不合理或不确切的地方,就需要返回概要设计的步骤,重新设计各个对象类,以及它们的功能和相互关系。

4. 面向对象实现

详细设计完成之后,就可以进入最后的实现阶段——编码阶段。实现阶段的主要任务包括:

- 选择一种合适的面向对象的编程语言,如C ++ 、Object Pascal、Java 等。
- 用选定的语言进行编码,对详细设计步骤所得到的软件系统的各对象类进行详尽的描述。
- 将编写好的各个类代码模块根据类的相互关系集成。
- 进行单元测试和综合测试。利用开发人员提供的测试样例和用户提供的测试样例分别对编码完成的各个模块和整个软件系统进行测试。

如果在实现阶段的工作过程中,发现设计或分析阶段隐藏的问题,需要及时返回相关的步骤做相应的调整。

实现阶段完成后,最终的可运行应用软件系统就全部完成了。

综上所述,面向对象的软件开发可概括为如下的过程:首先是分析用户需求,从问题中抽取对象模型;接着将模型细化,设计对象类,包括类的属性和类间相互关系,同时考虑是否有可以直接引用的已有的类或部件;然后选定一种面向对象的编程语言,用具体编码实现上一阶段类的设计;最后进行测试,完成整个软件系统。

由于对象的概念能够从应用问题实际构成(对象)的各个事物之间的关系(对象之间的联系)出发,来表述和处理应用问题,所以面向对象的软件开发方法对于结构化方法而言,具有更好的直观性、可重用性和可扩展性,使得"分析、设计、实现"的开发过程更加高效、快捷。当用户需求变化时,也比较容易在以前工作的基础之上修改和完成。

1.5.8 面向对象方法的特点

面向对象方法与传统的面向过程的方法相比,具有许多优点,主要表现在如下几个方面。

1. 符合人们的思维方式

传统的面向过程的方法以算法为核心,把数据和处理过程作为相互独立的两个部分,忽略了数据和操作的内在联系。这种数据和处理(功能)分离的软件设计结构与人类现实世界的实际情况很不一样,与人们的思维方式也不一致。因此,对现实世界的认识与对该问题的解存在着一道很深的理解上的鸿沟。

实际上,用计算机解决的问题都是现实世界中的应用问题,这些应用问题都是由一些相互间存在一定联系的事物组成。每个事物都具有属性和一定的行为特征。因此,把描述事

物属性的数据结构和表示事物的动态行为的操作放在一起构成一个整体，才能完整地、自然地表示客观世界的实体。面向对象方法正是把客观世界的事物看作对象，理解对象之间的关系构成整个软件系统。这种方法符合现实世界的实际，也符合人们的思维方式。而且，这种方法也提供了一套好的机制，当人们对某个应用问题的认识逐步深入时，开始设计的对象系统可以利用类的派生机制，逐步派生出更具体的派生类。这样的开发过程符合人们认识客观世界、解决复杂问题时逐步深化的渐进过程。

2. 可重用性

软件系统是由模块组成的。可重用性指的是，在一个软件系统中用到的模块，可以重复地被其他项目所使用，从而，可以在多个不同的系统中发挥作用。采用可重用模块来构造软件系统有如下优点：提高开发效率，缩短开发周期并可降低成本；由于采用了已被证明是正确的、有效的模块，所以程序的质量能够得以保证，维护也就容易；采用可重用模块来构建软件，能提高软件系统的标准化程度，符合大规模软件系统开发的需要。

模块要能够被重用，它必须是一个结构完整、功能明确、接口清晰的独立的软件结构。该结构具有良好的可移植性，可以使用在各种不同的软硬件环境和不同的程序框架中。传统的重用技术以标准函数库中的函数作为可重用的部件，但若标准函数是用传统的面向过程的方法开发的，它就不具备独立性，而必须在数据上运行。要采用这样的模块，相应的数据也必须重用。若不同场合下的不同应用有着不同的数据，那么就必须修改数据，或者修改这个模块。

面向对象方法将数据和其操作结合在一起，又具备封装性和信息隐藏的机理，使对象的外部只能看到该对象具有的功能（即可以实施的操作），但看不到对象内部的具体实现方法，这就使对象具有良好的独立性。而且，在面向对象方法中，若要实施软件重用，可以采用创建类的实例，或派生出一个满足当前需要的新类等方法。这种方法具有很大的灵活性，继承机制使子类不仅可以重用父类的数据结构和程序代码，而且可以在父类的基础上方便地修改和扩充，这种修改并不影响对原有类的使用。

面向对象方法所实现的可重用性是自然的、且是最成功的一种方法。

3. 可管理性

面向过程的开发方法是以过程或函数作为基本单元来构建整个系统的，它不适合大规模软件系统的开发。因为，当软件系统的规模变大时，需要的过程和函数数量也成倍增多，不利于管理和控制。而面向对象的开发方法采用类作为构建系统的部件，而类作为软件的部件，其内涵比过程和函数丰富、复杂得多，这将使整个软件项目的组织管理更加合理和方便。

一个软件系统，如采用面向过程的开发方法来实现，可能需要 1 000 个过程或函数，要管理好这 1 000 个过程或函数，正确控制它们在系统各种可能状态下的行为，处理它们之间错综复杂的关系，是一件非常困难和麻烦的工作，也容易出现失误和遗漏。若系统所包含的

过程或函数大量增加时，将难以控制和管理。但如果采用面向对象的开发方法来实现同一系统，该系统可能仅用50个类，平均每个类包含20个方法就可以完成同样的功能。50相对与1 000，大大降低了管理、控制的工作量，从开发效率和质量保证等各个方面，都有很大的优越性。

另外，面向对象开发方法中的类，把数据和其上的操作封装在一起，使得仅本类的有限个方法才可以操纵、改变这些数据。这样，仍以上面的例子为例，当出现数据的错误时，只需要检查与该数据相关的在同一个类中的30个方法即可，而在面向过程开发方法中处理相同的问题时，则可能需要把所有的1 000个过程或函数统统检查一遍，两者的工作量、效率和难易程度是不言而喻的。

正是由于面向对象方法符合人们的思维方式，具有可重用性好、易管理的特点，使得采用这种方法开发的软件系统容易理解、容易修改且稳定性好，易于开发大型的软件产品。

本章小结

本章对计算机及软件技术作了概要的介绍，给出了“计算作为一门学科”的概念，使读者初步了解“计算作为一门学科”的重要性。本章介绍了计算学科的研究内容，包含离散结构、程序设计基础、算法、计算机体系结构、操作系统、信息系统、计算机网络等领域，归纳起来，这些领域涉及计算的理论基础、计算机的硬件结构、计算机软件系统等方面。

了解了计算学科的概貌后，进一步讨论了计算机软件的核心概念以及计算机软件技术涉及的内容。接着，给出了软件工程的概念，介绍了软件开发方法和技术，包括结构化方法和面向对象的软件开发方法和过程。

习 题 一

1. 计算学科要解决的根本问题是什么？
2. 什么是计算学科？
3. 计算学科研究的主要内容是什么？
4. 图灵是如何揭示计算的本质的？
5. 什么是存储程序式计算机？它由哪几部分组成？
6. 计算机系统由哪两大部分组成？
7. 什么是操作系统虚拟机？
8. 什么是算法？算法最主要的特征是什么？

9. 数据结构主要由哪3个方面组成?
10. 什么是程序?程序的3种最基本的控制结构是什么?
11. 什么是软件?它的主要作用是什么?
12. 什么是软件工程?
13. 软件生命周期可以划分为哪几个阶段?
14. 用结构化方法进行软件开发的过程要经过哪些步骤?
15. 结构化方法的优点是什么?存在的主要问题是什么?
16. 什么是面向对象方法?它涉及的要点是什么?
17. 用面向对象方法进行软件开发的过程是什么?
18. 面向对象方法有什么优点?

第二章　数据结构与算法

2.1　数据结构概述

利用计算机进行数据处理是计算机应用的一个重要领域。在进行数据处理时，实际需要处理的数据元素可能很多，如何组织这些数据，数据元素之间的逻辑关系是什么，使用何种存储结构将这些数据元素存放在计算机中，如何提高数据处理的效率，节省计算机的存储空间，都是数据结构主要研究和讨论的问题。本章仅介绍线性表、栈和队列、树和二叉树等基本数据结构，并介绍查找和排序的基本方法。

2.1.1　基本概念和术语

（1）数据（Data）

数据是指所有能输入到计算机中并被计算机识别、存储和加工处理的符号的总称。例如，整数、实数、字符、声音、图形、图像等，都是计算机处理的信息的某种特定的符号表示形式。

（2）数据元素（Data Element）

数据元素是数据的基本单位，有时将数据元素简称为元素。例如，在一列实数（2.5，3，-21.5，94）中，一个数据元素是一个实数。有时一个数据元素由多个不同类型或同类型的数据项组成，此时习惯上称数据元素为记录。

（3）数据项（Item）

数据项是数据的最小单位，又称为数据域。例如，由数据项学号、姓名、性别、成绩可构成一个学生记录，多个学生记录组成一张学生登记表，如表2.1.1所示。

表2.1.1　学生登记表

学号	姓名	性别	成绩
20023001	李志成	男	86
20023002	顾晓燕	女	83
20023003	王　伟	男	91
20023004	陈　鹏	男	78
20023005	刘洁云	女	85

(4) 数据对象(Data Object)

数据对象是指具有相同性质的数据元素的集合,是数据的一个子集。例如,整数数据对象是集合 $D=\{0,\pm1,\pm2,\cdots\}$,大写英文字母数据对象是集合 $L=\{'A','B',\cdots,'Z'\}$。数据对象可以是无穷集,也可以是有穷集。

一般情况下,数据对象中,数据元素之间不是孤立无关的,它们之间存在着某种关系,数据元素之间的关系称为结构。

(5) 数据结构(Data Structure)

数据结构是指数据元素之间存在一种或多种特定关系的数据元素的集合。数据结构包括逻辑结构和物理结构两个方面。数据元素之间的逻辑关系,称为数据的逻辑结构,它与数据的存储无关,是独立于计算机的,可以用一个数据元素的集合和定义在此集合上的若干关系来表示。而数据元素及其关系在计算机存储器内的表示(或映像),称为数据的物理结构,又称为存储结构。

数据的逻辑结构通常有下列4类。集合结构指的是数据元素之间除了"同属于一个集合"外,别无其他关系;例如,白菜、萝卜、番茄、黄瓜等都同属于蔬菜类,但这些蔬菜之间无其他关系;线性结构指的是数据元素之间存在着"一对一"的线性关系,例如表2.1.1所示的学生登记表,学生记录之间存在一种前驱与后继的线性关系;树形结构指的是数据元素之间存在着"一对多"的关系,例如,在一本书的目录中,目录与它的子目录之间存在着"一对多"的关系;图状或网状结构指的是数据元素之间存在着"多对多"的网络关系。例如,城市之间的公路网。图2.1.1(a)、(b)、(c)、(d)分别表示上述4类基本结构。

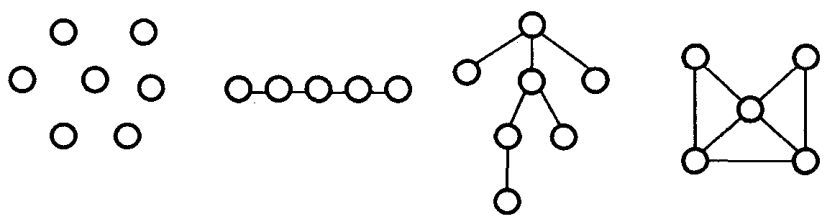


图2.1.1 4类基本结构

在数据的存储结构中,除了要存放数据元素的信息外,还需要存放表示各数据元素之间关系的信息。存放数据之间的关系,常用的有两种不同的方法,相应得到两种不同的存储结构:一种是把逻辑上相邻的数据元素存储在物理位置也相邻的存储单元里,数据元素之间的逻辑关系由存储单元的邻接关系来体现。另一种是使用指针表示数据元素之间的逻辑关系,称为链式存储结构。这种存储表示中,各个数据元素的存储位置可以随意,不要求逻辑上相邻的数据元素在物理位置上也相邻。

除了上述两种存储结构外，还可以采用索引存储结构与散列存储结构，限于篇幅，本章不再介绍。

一般来说，根据需要一种数据的逻辑结构可以表示成为多种存储结构。例如，表 2.1.1 所示的学生登记表可以用顺序存储结构的一维数组表示，也可以用链式存储结构的单链表来表示。

2.1.2 算法及其描述

1. 算法

算法(Algorithm)是对特定问题求解过程的一种描述，它是指令的有限序列。算法应具有以下 5 个基本特征：

- 有穷性。在任何情况下，算法必须在执行有穷步后结束，即算法中的操作步骤为有限步，且每个步骤都能在有限时间内完成。
- 确定性。算法中每一条指令必须有确切的含义，无二义性。在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入只能得出相同的输出。
- 可行性。算法中描述的每一个步骤操作必须能够实现，且能够达到预期的目的。
- 输入。一个算法有零个或多个输入。
- 输出。一个算法有一个或多个输出，是算法进行信息加工后得到的结果。

算法和程序是有区别的。程序可以不满足有穷性特征。例如，一个操作系统可以无休止地运行，直到系统停工。此外，程序还必须考虑运行环境的限制及程序实现时的各种细节问题。

求解某个具体问题时，可能有多个不同的算法，如何评价算法的优劣，从中选出性能较好的算法呢？

通常，设计一个好的算法需主要考虑如下 4 点：

- 正确性。算法首先应当是正确的，能够满足具体问题的要求。
- 可读性。算法应当可读性好，便于阅读与交流，便于调试与修改。
- 健壮性。当输入非法数据时，算法能作出反应或进行相应的处理，而不会产生莫名其妙的输出结果。
- 效率与存储需求。

2. 算法的评价

评价一个算法优劣的主要标准是，算法的执行效率与存储需求。算法的效率指的是时间复杂度(Time Complexity)，存储需求指的是空间复杂度(Space Complexity)。

一个算法转换为程序后的执行时间与计算机的处理速度、选用的程序设计语言及其编译程序等诸多因素有关。客观地评价算法本身的效率，不应考虑与计算机硬件、软件有关的因素，而应使用时间复杂度来评估算法。为说明时间复杂度，可看下例。

设某个算法，其主要的基本操作含如下程序段：

```
① for ( i = 1;i < = n;i + + )        /* 其中 i≤n 共计被执行 n + 1 次 */
②    for ( j = 1;j < = n;j + + )     /* 其中 j≤n 共计被执行 n(n + 1) 次 */
③        x = x + 1;                  /* x = x + 1 共计被执行 n²次 */
```

一条语句在算法中被重复执行的次数称为该语句的频度。上例中语句① 的循环控制变量 i 从 1 增加到 $n+1$，循环才会终止，故语句① 的频度为 $n+1$。语句② 作为语句① 的循环体应该执行 n 次，每次语句② 本身要执行 $n+1$ 次，故语句② 的频度是 $n(n+1)$。语句③ 是语句② 的循环体，应该执行 n^2 次，故语句③ 的频度是 n^2。该算法中所有语句的频度之和

$$T(n) = n + 1 + n(n+1) + n^2 = 2n^2 + 2n + 1 \qquad (式\ 2.1.1)$$

可以看出 $T(n)$ 是 n 的函数。其中 n 为问题的规模（如线性表的长度，矩阵的阶数等）。时间复杂度是指当问题的规模 n 趋向无穷大时，$T(n)$ 的增长率。对于式 2.1.1，当 n 趋向无穷大时，有

$$\lim_{n\to\infty}\frac{T(n)}{n^2} = \lim_{n\to\infty}\frac{2n^2 + 2n + 1}{n^2} = 2 \qquad (式\ 2.1.2)$$

当 n 充分大时，$T(n)$ 和 n^2 之比是一个不等于零的常数，$T(n)$ 的增长率和 n^2 的增长率相同，即 $T(n)$ 与 n^2 是同数量级，或者说是同阶的，可记作 $T(n) = O(n^2)$。称 $O(n^2)$ 是上述算法的时间复杂度。数学符号“O”的概念如下。

若 $T(n)$、$f(n)$ 是正整数 n 的两个函数，则 $T(n) = O(f(n))$ 表示，存在正的常数 M 和 $n0$，使得当 $n \geqslant n0$ 时都满足 $|T(n)| \leqslant M|f(n)|$，即下式成立。

$$\lim_{n\to\infty}\frac{T(n)}{f(n)} = M \neq 0$$

一般认为，求解同一个问题，随着 n 的增大，函数 $T(n)$ 增长速度较慢的算法为优。

下面再举例，说明如何求算法的时间复杂度。对于下列几个程序段：

```
① x = x + 1;
② temp = i;
   i = j;
   j = temp;
③ for(i = 1; i < = n;i + + )
      x = x + 1;
④ for(i = 1; i < = n;i + + )
      for (j = 1;j < = n;j + + )
          for (k = 1;k < = n;k + + )
              x = x + 1;
```

程序段①的时间复杂度为 $O(1)$。程序段②中,3 条语句的频度均为 1,程序段的执行时间是常数,故时间复杂度也为 $O(1)$。程序段③语句频度为 $n+1$,故时间复杂度为 $O(n)$。程序段④的时间复杂度为 $O(n^3)$。

常用的时间复杂度,按数量级递增排列,依次为:常数阶 $O(1)$,对数阶 $O(\log_2 n)$,线性阶 $O(n)$,线性对数阶 $O(n\log_2 n)$,平方阶 $O(n^2)$,立方阶 $O(n^3)$,……,k 次方阶 $O(n^k)$,指数阶 $O(2^n)$。

类似于时间复杂度的讨论,一个算法的空间复杂度定义为该算法所耗费的存储空间,它也是规模 n 的函数。

一个算法可以用多种方式来描述,如自然语言、程序语言、流程图等。本章中采用 C 语言作为描述工具。算法用函数描述,在描述时尽量压缩语言的细节,每一部分所使用的数据类型统一定义。应用举例部分所给出的实例都是完整的 C 语言程序,便于读者上机验证。

3. C 语言的简要说明

下面对本章使用的 C 语言作简要说明:

(1) 数据结构表示

数据结构的表示(存储结构)都用类型定义(typedef)的方式描述。数据元素类型约定为 ElemType,由用户在使用该数据类型时再自行具体定义。

(2) 实现算法

实现基本操作的算法都用以下形式的函数描述:

```
函数类型  函数名(函数参数表)
{ 内部数据说明;
  语句序列;
}
```

函数类型说明函数返回值的类型。如果要明确表示无返回值,则用 void 定义为无类型。函数参数表指明每个参数的类型。由花括号“{”和“}”括起来的部分是函数体,包括内部数据说明部分和语句部分。

(3) 赋值语句

简单赋值:变量名 = 表达式;

串联赋值:变量名 1 = 变量名 2 = …… = 变量名 k = 表达式;

条件赋值:变量名 = 条件表达式? 表达式 T:表达式 F;

(4) 选择语句

条件语句 1:if(表达式) 语句;

条件语句 2:if(表达式) 语句;

else 语句;

开关语句： switch(表达式)

```
{ case 值1:语句序列1;break;
  case 值2:语句序列2;break;
  ...
  case 值n:语句序列n;break;
  default: 语句序列n+1;
}
```

(5) 循环语句

for 语句：for(表达式1;表达式2;表达式3) 语句;

首先计算表达式1的值,然后计算表达式2的值,若结果非0,则执行循环体语句,接着计算表达式3,然后再次计算表达式2的值,若结果仍为非0,则继续执行循环体,如此循环,直到表达式2的值为0时为止。

while 语句： while(表达式) 语句;

首先计算表达式的值,若结果非0,则执行循环体语句,然后再次计算表达式的值,若结果仍为非0,则继续执行循环体,如此循环,直到表达式的值为0时为止。

do-while 语句：

```
do
{ 语句序列;
} while(表达式);
```

首先执行循环体语句,然后计算表达式的值,若值为非0,则继续执行循环体,如此循环,直到表达式的值为0时为止。

(6) 结束语句

函数结束语句:return 表达式;

return;

case 及循环结束语句：break;

异常结束语句：exit (异常代码);

(7) 输入和输出语句

输入语句： getchar();

gets();

scanf([格式串],变量1,…,变量n);

输出语句： putchr();

puts();

printf([格式串],表达式1,…,表达式n);

(8) 注释 /*文字序列*/

(9) 逻辑运算

非运算！ 若 a 为 0，则 ！a 为 1，否则为 0。
与运算 && 若 a、b 都为非 0，则 $a\&\&b$ 值为 1，否则为 0。
或运算 || 若 a、b 都为 0，则 $a||b$ 为 0，否则为 1。

2.2 线 性 表

线性表是最简单且最常用的一种数据结构，其基本特点是除第一个和最后一个数据元素外，每个数据元素只有一个直接前驱和一个直接后继。本节讨论的线性表和随后要讨论的栈、队列都属于线性结构，只是基于数据元素之上的操作不同。

2.2.1 线性表的定义及基本操作

线性表(Linear List)是由 $n(n\geq 0)$ 个相同类型数据元素 $a_1,a_2,\cdots,a_n$ 构成的有限序列。记为$(a_1,a_2,\cdots,a_n)$。线性表中数据元素的个数 n 称为线性表的长度，$n=0$ 时称为空表。a_i $(1\leq i\leq n)$是线性表中第 i 个数据元素。a_i 是抽象表示符号，在不同的情况下含义不同，它可以是一个数、或一个符号，也可以是其他更复杂的信息。

例如，某个班级某一门课程的成绩表(97,94,90,…,54) 是一个线性表，表中的数据元素是整数。

又如26个英文字母的字母表$(A,B,\cdots,Z)$ 也是一个线性表，表中的数据元素是单个字母。

有时，一个数据元素可以由若干个数据项组成。在这种情况下，常把数据元素称为记录(Record)。含有大量记录的线性表又称文件(File)。

例如，表 2.2.1 所示的职工情况登记表中，每个职工的情况为一个记录，它由职工号、姓名、性别、住址和电话号码等 5 个数据项组成。

表 2.2.1 职工情况登记表

职工号	姓名	性别	住址	电话号码
00026	赵 明	女	北京路 54 号	87554017
00027	孙小宁	男	南京路 78 号	87364502
00123	吴 虹	女	青岛路 25 号	85427625
00243	李大海	男	大连路 89 号	83476521
…	…	…	…	…

由上述例子可见，线性表中的数据元素可以是各种类型，但同一线性表中的元素必定属于同一数据类型，相邻数据元素之间的相对位置是线性的。

当表的长度 $n>0$ 时，a_1 称为第一个元素，没有前驱，a_n 称为最后一个元素，没有后继。a_i 是第 i 个元素，a_{i-1} 是 a_i 的直接前趋($2\leqslant i\leqslant n$)，a_{i+1} 是 a_i 的直接后继($1\leqslant i\leqslant n-1$)。除了 a_1 和 a_n 之外，每个 a_i 都有且仅有一个直接前趋和一个直接后继。

根据需要可以对线性表中的任何数据元素进行访问，还可以进行插入和删除等操作。

对线性表进行的基本操作有以下几个。

(1) InitList(L)

构造一个空的线性表 L，即表的初始化。

(2) ListLength(L)

确定线性表 L 的长度。

(3) GetElem(L,i)

取线性表 L 的第 $i(1\leqslant i\leqslant n)$ 个数据元素。

(4) InsertList(L,i,x)

在线性表 L 的第 $i(1\leqslant i\leqslant n+1)$ 个位置插入一个同类型的新元素 x，使得表中原来的第 $i,i+1,\cdots,n$ 个元素的序号依次变为 $i+1,i+2,\cdots,n+1$，表的长度由 n 变为 $n+1$。

(5) DeleteList(L,i,x)

删除线性表 L 的第 $i(1\leqslant i\leqslant n)$ 个数据元素，使得表中原来的第 $i+1,i+2,\cdots,n$ 个元素的序号依次变为 $i,i+1,\cdots,n-1$，表的长度由 n 变为 $n-1$。

(6) LocateElem(L,x)

根据某个指定的元素值或数据项的值 x 对线性表 L 进行查找。若 L 中有元素的值与 x 相同，则返回首次找到的元素在 L 中的位置。若查找失败，则返回 -1。

(7) ListSort(L)

对线性表进行排序，即按元素值或某个数据项值的递增(或递减)次序重新排列线性表中各元素的位置。

数据元素按递增(或递减)次序排列的线性表称为递增(或递减)有序表，简称为有序表，否则称为无序表。

此外，还可以将两个表连接成一个表，或者把一个表拆分成两个或多个表等。对于实际问题中涉及的其他更为复杂的运算，可以用基本运算的组合来实现。

线性表运算的实现及其算法设计与它的存储结构有关。线性表最常用的存储结构分为顺序存储结构和链式存储结构两种。

2.2.2 线性表的顺序表示和实现

1. 线性表的顺序存储结构

线性表的顺序存储结构指的是用一组地址连续的存储单元依次存储线性表的各数据元素。这种存储结构的特点是，逻辑上前后相邻的数据元素，它们的物理次序也是邻接的。采

用这种方法存储的线性表简称为顺序表(Sequential List)。

数据元素在存储器中所占存储单元的数目取决于数据元素本身所含的信息量和具体机器的字长。数据元素的存储地址是指所占存储单元中的第一个存储单元的地址。假设线性表的每个数据元素需占用 k 个存储单元,$LOC(a_1)$ 为线性表的第一个数据元素的存储首址,则线性表中第 i 个数据元素的存储首址

$$LOC(a_i) = LOC(a_1) + (i-1) * k \qquad (式\ 2.2.1)$$

线性表第一个数据元素的存储地址称为该线性表的首地址或基地址。若设首地址为 b,则式 2.2.1 又可以写为:

$$LOC(a_i) = b + (i-1) * k \qquad (式\ 2.2.2)$$

一个长度为 n 的线性表存储在大小为 $maxlength$($n \leqslant maxlength$)的顺序结构中,前 n 个位置上的存储结构示意图如图 2.2.1(a)所示。

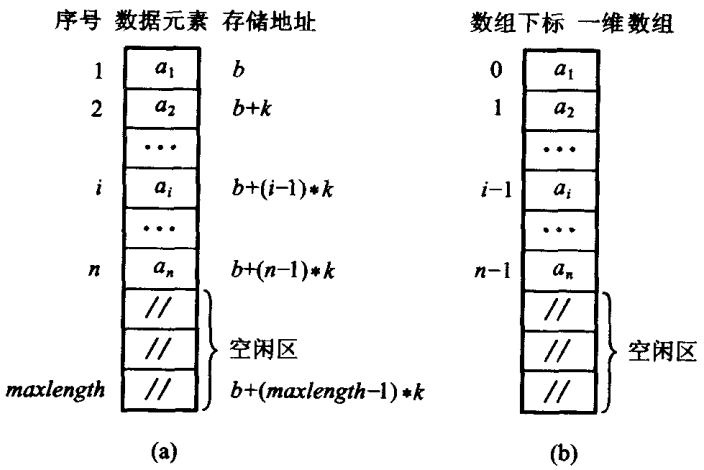


图 2.2.1 线性表的顺序存储结构示意图

线性表中每个元素的存储地址都可以由公式计算得到,只要确定了存储线性表的起始位置,线性表中任一数据元素都可随机存取,所以线性表的顺序存储结构是一种随机存取的存储结构。

高级程序设计语言允许使用一维数组,一维数组在计算机内被映像为向量,即顺序存储结构,因此在算法设计中,可以用一维数组来表示线性表的顺序存储结构,数组的基本类型与线性表中数据元素的类型相同。在 C 语言中,由于数组的下标从 0 开始,故线性表中的第一个元素存储在数组的起始位置,即下标为 0 的位置上,第二个元素存储在数组的下标为 1 的位置上,依此类推,第 n 个元素存储在数组的下标为 $n-1$ 的位置上,如图 2.2.1(b)所示。

线性表的长度 n 随着插入新元素而增大,随着删除原有的元素而变小。在线性表的最后一个元素的后面留下一个“空白”区,这个“空白”区称为自由区或空闲区。

在算法设计中，可以不考虑数据元素具体是何种数据类型，仅将其看作是一种抽象数据类型，定义为 ElemType。用结构类型定义描述线性表的顺序存储结构如下：

```
#define MAXLENGTH 100                    /* 假定表空间最多容纳 100 个元素 */
typedef int ElemType;                    /* 假定 ElemType 为整型 */
typedef struct
{ ElemType list[MAXLENGTH];              /* 用于存放线性表 */
  int length;                            /* 当前线性表的长度 */
}SeqList;
```

在上述定义中，*maxlength* 及 ElemType 要根据实际问题的需要选定。

2. 顺序存储结构中基本操作的实现

用顺序存储结构作线性表的存储结构时，线性表的有些操作很容易实现。例如，设 L 是指向 SeqList 的指针，则表的初始化操作是将表的长度置为 0，即 L -> length = 0；求表长和取表中第 i 个结点的操作只需分别返回 L -> length 和 L -> list[i - 1]即可。对线性表查找及排序的操作，将在后面的章节讨论。下面主要讨论插入和删除两种操作的实现。

(1) 插入算法

在线性表的第 $i(1 \leqslant i \leqslant n+1)$ 个位置插入一个新元素 x 时，意味着长度为 n 的线性表 $(a_1,\cdots,a_{i-1},a_i,\cdots,a_n)$ 变成长度为 $n+1$ 的线性表。表示为 $(a_1,\cdots,a_{i-1},x,a_i,\cdots,a_n)$，即 $(a_1,\cdots,a_{i-1},a'_i,a'_{i+1},\cdots,a'_{n+1})$。

其中 a'_i 为新插入的元素 x，a'_{i+1} 为原表中的 a_i，其余类推，a'_{n+1} 为原表中的 a_n。

如图 2.2.2 所示，在插入 x 之前，首先将线性表中的元素 $a_n,a_{n-1},\cdots,a_i$ 依次向后移到

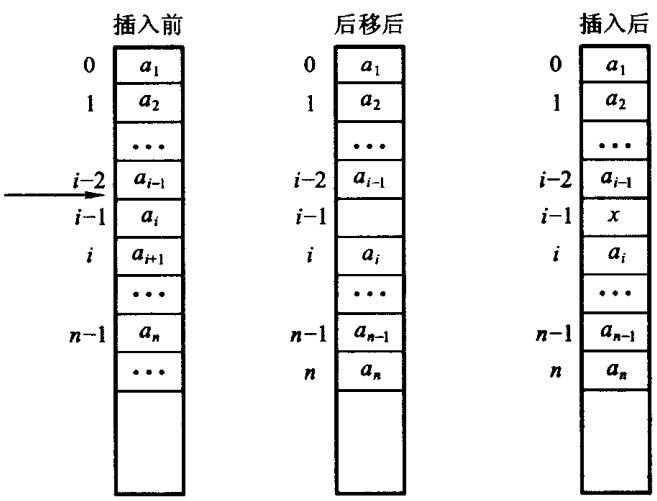


图 2.2.2　顺序表中插入元素过程示意图

第 $n+1, n, \cdots, i+1$ 个位置上，然后将待插入的元素 x 存储到第 i 个位置上。x 的序号为 i，原来的第 i 个到第 n 个元素的序号自动变为 $i+1, i+2, \cdots, n+1$。如果希望将新元素 x 插在最后元素 a_n 之后，即插在第 $n+1$ 个位置上，则不需要移动表中原有的元素。

值得注意的是，算法中元素后移时必须从线性表中最后一个元素开始后移，如果最后一个元素之后没有多余的自由空间，即当表的大小 n = maxlength 时，那么插入一个元素将会发生上溢(overflow)。解决的办法是，重新分配一个较大一些的连续存储区，作为新的顺序存储结构，并将原存储结构中的元素依次复制过来，再进行插入操作。

插入算法描述如下：

```
void InsertList(SeqList *L, int i, ElemType x)
{ /*在顺序表 L 中第 i 个位置之前插入一个新元素 x */
    int j,n = L -> length;
    if (i < 1 || i > n + 1)
    {    printf("\n i 值不合法");
         exit(1);
    }
    if (n >= MAXLENGTH)
    {    printf("\n  表空间溢出");
         exit(1);
    }
    for(j = n - 1; j >= i - 1; j - -)
    L -> list[j + 1] = L -> list[j];          /* 元素后移一个位置 */
    L -> list[i - 1] = x;                     /* 插入 x */
    L -> length + +;                          /* 表长加 1 */
}
```

(2) 删除算法

删除线性表的第 $i(1 \leqslant i \leqslant n)$ 个数据元素，长度为 n 的线性表 $(a_1, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$ 变成长度为 $n-1$ 的线性表 $(a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$，即为 $(a_1, \cdots, a_{i-1}, a'_i, \cdots, a'_{n-1})$。其中 a'_i 为原表中的 a_{i+1}，其余类推，a'_{n-1} 为原表中的 a_n。

此时，只需要把线性表中第 $i+1$ 个到第 n 个数据元素依次前移到第 i 个到第 $n-1$ 个位置上。若 $i=n$，则无需移动元素。删除线性表中第 i 个数据元素过程的示意图如图 2.2.3 所示。

删除算法描述如下：

```
void DeleteList(SeqList *L,int i,ElemType *x)
{ /*   删除顺序表 L 中第 i 个元素，并用 x 返回其值 */
```

```
int j,n=L -> length;
 if (i<1||i>n)
 { printf("\n i 值不合法");
   exit(1);
 }
 *x = L -> list[i-1];                    /* 被删元素的值赋给 *x */
 for(j=i; j<=n-1; j++)
     L -> list[j-1] = L -> list[j];      /* 元素前移一个位置 */
 L -> length--;                          /* 表长减 1 */
}
```

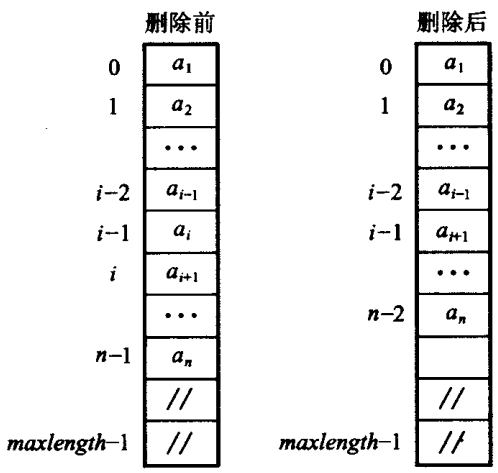


图 2.2.3　顺序表中删除元素过程示意图

从上面的讨论和算法可知，在顺序表中插入或删除一个数据元素时，其时间要耗费在移动元素上，而移动元素的个数取决于表长 n 及插入或删除元素的位置 i。

假设在长度为 n 的线性表的第 i $(1 \leqslant i \leqslant n+1)$ 个位置插入一个元素的概率为 pi，所需移动元素次数为 $n-i+1$，那么每插入一个元素所需移动元素次数的平均次数

$$E_{is} = \sum_{i=1}^{n+1} pi(n-i+1) \qquad (式\ 2.2.3)$$

假设在线性表的任何位置插入元素的概率相等，即 $p_i = \frac{1}{n+1}$，则

$$E_{is} = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{n}{2} \qquad (式\ 2.2.4)$$

由此可见，在顺序表中插入一个元素平均要移动表中一半的元素。就数量级而言，算法的时间复杂度是 $O(n)$。最好的情况是，当 $i=n+1$，即在表尾插入时，不需要移动元素；最

坏的情况是，当 $i=1$，即在表头插入时，需移动表中所有的 n 个元素。

假设 q_i 是删除第 i 个元素的概率，所需移动元素的次数为 $n-i$，那么每删除一个元素所需移动元素次数的平均数

$$E_{de}=\sum_{i=1}^{n}q_i(n-i) \qquad (式2.2.5)$$

假设在任何位置删除元素的概率相等，即 $q_i=\frac{1}{n}$，则

$$E_{de}=\frac{1}{n}\sum_{i=1}^{n}(n-i)=\frac{n-1}{2} \qquad (式2.2.6)$$

即在顺序表中删除一个元素平均约移动表中一半的元素。与插入算法相同，算法的时间复杂度也是 $O(n)$。最好的情况是当 $i=n$ 时，不必移动元素。最坏的情况是当 $i=1$ 时，需要移动 $n-1$ 个元素。

(3) 顺序存储结构的特点

顺序存储结构的特点是结构简单，逻辑上相邻的数据元素在物理位置上也相邻，存储空间利用率高。此外，线性表中任意数据元素的存储地址可由公式直接导出，即线性表中任一元素可随机存取，存取元素的效率高。

其缺陷是需要预先根据线性表的长度分配连续的存储空间，空间预分配过大，将造成空间浪费，预分配太小又将使空间溢出机会增多。在线性表长度 n 难以估计或者虽有足够的存储空间但空间不连续时，会给算法设计带来不便。这时采用动态分配的顺序存储结构，可以在一定程度上弥补静态分配的不足。

其次，每插入或删除一个数据元素，平均需要移动表中一半的元素，数据移动量大，故顺序存储结构一般用于线性表长度变化不大，易于事先估计其大小且主要操作是进行查找，很少做插入、删除操作。

2.2.3 顺序表应用举例

利用静态分配的顺序表建立一个学生登记表，表中每个记录包含学生的学号、姓名、某门功课的成绩。学生基本信息通过键盘输入，对学生登记表的处理包括以下选项：

- 输出所有的学生信息。
- 求当前表中登记的学生人数，即表长。
- 求表中第 i 个学生。
- 在表的第 i 个学生之前插入一个学生。
- 删除表中的第 i 个学生。
- 根据学号在表中查找，若查找成功，则返回该学生在表中的位置，否则返回 -1。

实现上述操作的完整程序如下：

```
#define MAXLENGTH 30
typedef struct
{   int num;
    char name[10];
    int score;
} ElemType;

typedef struct
{   ElemType list[MAXLENGTH];
    int length;
} SeqList;

void InitList(SeqList *L)
{   /*构造一个空的线性表*/
    L->length=0;
}

int ListLength(SeqList *L)
{    /*求表长*/
    return(L->length);
}

ElemType GetElem(SeqList *L, int i)
{    /*取表的第 i 个元素*/
    if(i<1||i>L->length)
    {     printf("\n i 值非法");
          exit(1);
    }
    return(L->list[i-1]);
}
void InsertList(SeqList *L,int i, ElemType x)
{    /*在表的第 i 个位置插入一个元素 x */
     int j,n=L->length;
     if(i<1||i>n+1)
```

```
    {    printf("\n i 值非法");
         exit(1);
    }
    if(n >= MAXLENGTH)
    {    printf("\n 表空间溢出");
         exit(1);
    }
    for(j = n - 1; j >= i - 1; j - -)      L -> list[j + 1] = L -> list[j];
    L -> list[i - 1] = x;
    L -> length + +;
}

void DeleteList(SeqList * L,  int i,  ElemType * x)
{   /* 删除表的第 i 个元素 */
    int j,n = L -> length;
    if(i < 1 || i > n)
    {    printf("\n i 值非法");
         exit(1);
    }
    * x = L -> list[i - 1];
    for(j = i; j <= n - 1; j + +)
         L -> list[j - 1] = L -> list[j];
    L -> length - -;
}
void OutElement(ElemType x)
{   /* 输出一个学生信息 */
    printf("\n %10d %10s %10d",x.num,x.name,x.score);
}

void OutList(SeqList * L)
{   /* 输出所有的学生信息 */
    int i,n = L -> length;
    printf("\n %10s %10s %10s","num","name","score");
    for(i = 0; i < n; i + +)
```

```
            OutElement(L ->.list[i]);
        printf("\n");
    }

    int LocateElem(SeqList *L, ElemType x)
    { /*在表L中查找学号等于x.num的学生,查找成功,返回在表中的位置,否则返回-1*/
        int i=0,n=L->length;
        if(n==0)
        {   printf("\n Empty List");                    /*空表,返回*/
            exit(1);
        }
        while(i<n && x.num!=L->list[i].num) i++;  /*根据学号在表中查找*/
        if(i<n) return(i+1);                  /*查找成功,返回在表中的位置*/
        else return(-1);                                /*查找失败,返回-1*/
    }

    void InputElement(ElemType *x)
    {   /*输入学生信息*/
        printf("\n input num:");
        scanf("%d",&x->num);
        printf("\n input name:");
        scanf("%s",x->name);
        printf("\n input score:");
        scanf("%d",&x->score);
    }

    void main()
    {   SeqList seql,*L;
        int i,j,k,flag=1;
        const maxsize=30;
        ElemType x;
        L=&seql;
    InitList(L);
```

```
for(i=0; i<MAXLENGTH; i++) /*输入所有学生信息*/
{    InputElement(&x);
     L->list[i]=x;
     ++L->length;
}
while(flag)
{    printf("\n 1:输出所有学生信息        2:求学生人数);
     printf("\n 3:求第i个学生              4:在第i个位置插入一个学生);
     printf("\n 5:删除第i个学生            6:根据学号查找);
     printf("\n 7:退出\n");
     scanf("%d",&k);
     while(k<1||k>7)
     {    printf("\n k Error Input again");
          scanf("%d",&k);
     }
     switch(k)
     {   case 1:OutList(L);
                 break;
         case 2:i=ListLength(L);
                 printf("\n Length of List is %d",i);
                 break;
         case 3:printf("\n input i:");
                 scanf("%d",&i);
                 x=GetElem(L,i);
                 OutElement(x);
                 break;
         case 4:printf("\n input i:");
                 scanf("%d",&i);
                 InputElement(&x);
                 InsertList(L,i,x);
                 break;
         case 5:printf("\n input i:");
                 scanf("%d",&i);
                 DeleteList(L,i,&x);
```

```
                OutElement(x);
                break;
            case 6:printf("\n input x.num:");
                   scanf("%d",&x.num);
                   i = LocateElem(L,x);
                     printf("\n x is in the %d position",i);
                break;
            case 7:flag = 0;
                printf("\n 再见");
        }
    }
}
```

2.2.4 线性表的链式表示与实现

如前所述，线性表的顺序存储结构需要一块连续的存储空间，而且在实现插入、删除操作时需移动大量数据元素，为避免大量元素的移动，可以采用另一种存储结构——链式存储结构。

1. 单链表

链式存储结构是线性表的另一种表示方法，它不要求逻辑上相邻的元素在物理位置上也相邻。其存储单元既可以是连续的，也可以是不连续的，甚至是零散分布在内存中的任何位置上，是一种非顺序存储结构。为了能正确表示数据元素在线性表中的顺序，在存储每个元素值的同时，还应该存储表示元素之间逻辑关系的信息，通常存储一个指示其直接后继元素位置的信息。这两部分信息组成数据元素的存储映像，称为结点。结点结构如图 2.2.4 所示。

图 2.2.4 结点结构

图 2.2.4 中 *data* 域称为数据域，用来存放元素的值，*next* 域称为指针域或链域，用来存放直接后继元素的存储地址。指针域中存储的信息称为指针或链。链式存储结构正是通过每个结点的指针域将线性表的 n 个结点按其逻辑顺序连接在一起。用链接方式存储的线性表称为线性链表，简称为链表(Linked List)。这种链表的每个结点只包含一个指针域，故又称为单链表(Single Linked List)。

例如,图 2.2.5 是线性表(*A*,*B*,*C*,*D*,*E*)的单链表示意图。

线性表中第一个结点无前驱结点,通常设立头指针 *head*,表示链表中第一个结点的存储地址。最后一个结点没有后继,其指针域为空,在图示中常用符号"∧"表示,在 C 算法和 C 程序中用符号常量 NULL 表示。

这里所关心的只是结点间的逻辑顺序,并不关心每个结点的实际存储位置,因此通常用箭头表示链域中的指针。图 2.2.5 所示的单链表可以直观画成图 2.2.6 所示的形式。

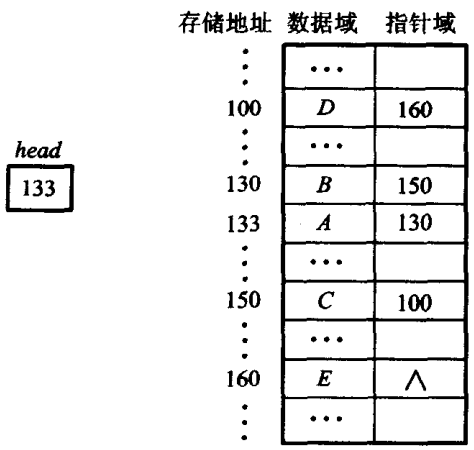


图 2.2.5 单链表示意图

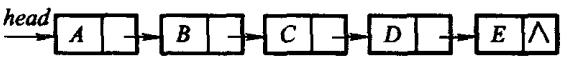


图 2.2.6 单链表逻辑状态

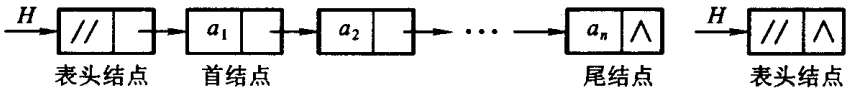
访问单链表时,通过头指针 *head*,可以找到单链表的第一个结点,通过第一个结点的指针可以找到第二个结点,依此类推,沿着结点的指针,可以访问到每一个结点。

单链表结构类型定义如下:

```
typedef struct node
{    ElemType data;
     struct node  * next;
} ListNode, * LinkList;
```

为了便于实现插入、删除结点的操作,通常在单链表的第一个结点之前增设一个表头结点。表头结点的结构与表中其他结点的结构相同,其数据域可以不存储任何信息,也可以存储如线性表的长度、表名等附加信息。表头结点的指针域存储线性表中第一个结点的地址。若线性表为空表,则表头结点的指针域为空。设 H 为单链表的头指针,带表头结点的单链表如图 2.2.7 所示,其中存储线性表第一个元素的结点称为首结点,存储最后一个元素的结

点称为尾结点。



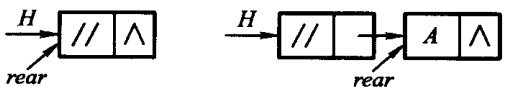
(a) 非空表 (b) 空表

图 2.2.7 带表头结点的单链表

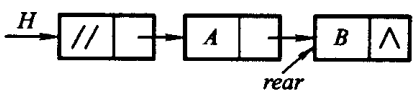
下面讨论用带表头结点的单链表作存储结构时,如何实现线性表的几种基本操作。在描述基本操作的 C 算法中,假设线性表中结点的数据类型是字符型。

(1) 建立单链表

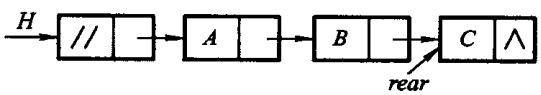
依新结点插入位置的不同,将生成的单链表分为先进先出表,后进先出表与有序表。先进先出表就是每次生成新结点后,总是将新结点插入到当前链表的表尾,新结点作为当前链表的尾结点。若以换行符'\n'作为输入结束标志,输入 A,B,C,则单链表的生成过程如图 2.2.8 所示。若遍历该链表,输出结点的值,结果为 A,B,C。



(a) 仅有表头结点 (b) 插入结点 A 之后



(c) 插入结点 B 之后



(d) 插入结点 C 之后

图 2.2.8 建立带表头结点的先进先出的单链表示意图

为方便新结点的插入,算法中设立一个尾指针 *rear*,它总是指向当前链表的尾结点,新结点总是插在尾结点之后,插入新结点后,修改尾指针,使其指向新结点。

建立带表头结点的先进先出的单链表算法如下:

```
LinkList CreateList_FFL( )
{
    LinkList H,p,rear;
    char ch;
```

```
    H = (LinkList) malloc(sizeof(ListNode)); /* 生成表头结点 */
    if(!H)
    {    printf("\n 存储分配失败");
         exit(1);
    }
    H -> next = NULL;
    rear = H;                              /* 链表初值为空,尾指针指向表头结点 */
    while((ch = getchar()) != '\n')
    {    p = (LinkList) malloc(sizeof(ListNode));/* 生成新结点 */
         if(!p)
         {    printf("\n 存储分配失败");
              exit(1);
         }
         p -> data = ch;
         p -> next = NULL;
         rear -> next = p;                 /* 新结点插入表尾 */
         rear = p;                         /* 尾指针指向新结点 */
    }
    return(H);                             /* 返回头指针 */
};
```

若生成新结点,总是将新结点插入到表头结点之后,新结点作为当前链表的首结点,这样建立的链表就是后进先出表。同样以换行符'\n'作为输入结束标志,输入 A,B,C,则单链表的生成过程如图 2.2.9 所示。若遍历该链表,输出每个结点的值,则结果为 C,B,A。

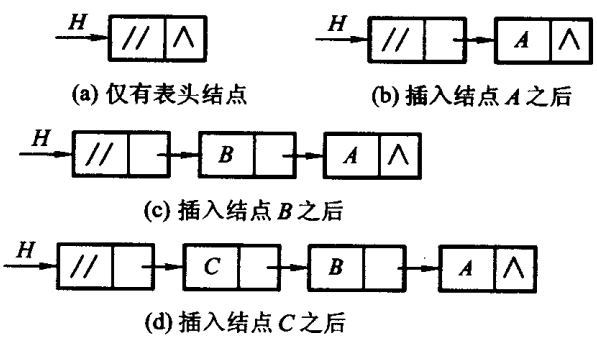
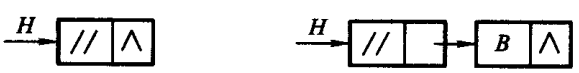


图 2.2.9 建立带表头结点的后进先出的单链表示意图

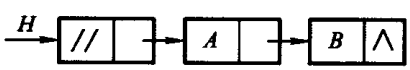
建立带表头结点的后进先出的单链表算法如下:

```
LinkList CreateList_FRL( )
{
    LinkList H,p;
    char ch;
    H = (LinkList) malloc( sizeof( ListNode) ) ;      /* 生成表头结点 */
    if( !H)
    {     printf( " \n 存储分配失败" ) ;
          exit( 1 ) ;
    }
    H -> next = NULL;
    while( ( ch = getchar( ) ) !  = '\n ')
    {     p = (LinkList) malloc( sizeof( ListNode) ) ;      /* 生成新结点 */
          if( ! p)
          {     printf( " \n 存储分配失败" ) ;
                exit( 1 ) ;
          }
          p -> data = ch;
          p -> next = H -> next;
          H -> next = p;                                  /* 新结点插入表头结点之后 */
    }
    return( H) ;                                          /* 返回头指针 */
}
```

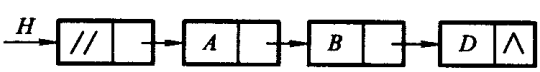


(a) 仅有表头结点

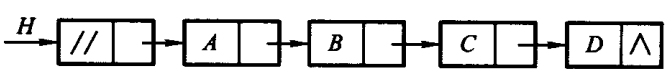
(b) 插入结点 *B* 之后



(c) 插入结点 *A* 之后



(d) 插入结点 *D* 之后



(e) 插入结点 *C* 之后

图 2.2.10　建立带表头结点的有序单链表示意图

有序单链表是指链表中结点的数据值按从小到大(或从大到小)顺序排列的链表。为了建立一个有序单链表,生成新结点后,必须找出该结点在链表中的合适位置,然后才能将它插入到链表中。如果以换行符'\n'作为结束标志,输入 B,A,D,C,则单链表的生成过程如图 2.2.10 所示。若遍历该链表,输出结果为 A,B,C,D。

由于引入了表头结点,在向链表中插入新结点时,不需要考虑空表以及新结点是插在表头、表中、表尾等多种情况,在所有位置上的插入操作都是一致的,不需要进行特殊处理。

建立带表头结点的有序单链表算法如下:

```
LinkList CreateList_ORL( )
{
    LinkList H,pre,cur;
    char ch;
    H = (LinkList)malloc(sizeof(ListNode));   /*生成表头结点*/
    if(!H)
    {    printf("\n 存储分配失败");
         exit(1);
    }
    H -> next = NULL;
    while((ch = getchar( ))!  ='\n')
    {    pre = H;                    /*pre 指向当前结点的前驱结点*/
         cur = H -> next;            /*cur 指向当前结点*/
     while(cur&&cur -> data < ch)            /*找出新结点在表中的合适位置*/
    {    pre = cur;
         cur = cur -> next;
    }
    cur = (LinkList)malloc(sizeof(ListNode));   /*生成新结点*/
    if(!cur)
    {    printf("\n 存储分配失败");
         exit(1);
    }
    cur -> data = ch;
    cur -> next = pre -> next;          /*新结点 cur 插在 pre 所指结点的后面*/
    pre -> next = cur;
}
return(H);
```

```
}
```

容易看出,以上 3 个算法的时间复杂度均为 $O(n)$。

(2) 查找结点

① 查找单链表中的第 i 个结点

在带表头结点的单链表中查找第 i 个结点,不能像在顺序表中那样根据序号直接访问顺序表中的元素,而只能采用顺序查找,即从头指针出发,沿结点的链域逐个向下搜索,直到搜索到第 i 个结点为止。

在带表头结点的单链表 H 中,查找第 i 个结点的算法如下:

```
LinkList GetElem(LinkList H,      int i)
{
    int j = 1;                          /* 计数器 j */
    LinkList p;
    p = H -> next;                      /* p 指向第一个结点 */
    while(p&&j < i)                     /* 顺指针向后查找 */
    {     p = p -> next;
          j + +;
    }
    if(j = = i&&p) return(p);           /* 查找成功,返回指向该结点的指针 */
    else return(NULL);                  /* 第 i 个结点不存在,返回 NULL */
}
```

② 查找具有给定值的结点

根据给定的数据项的值 x,在带表头结点的单链表中进行查找,若查找成功,则通过 i 返回首次找到的结点的位置,同时返回指向该结点的指针。若查找失败,则返回 NULL。

在带表头结点的单链表 H 中查找数据值为 x 的结点,算法如下:

```
LinkList LocateElem(LinkList H, ElemType x, int *i)
{
    LinkList p;
    P = H -> next; *i = 1;                  /* p 指向第一个结点 */
    While(p && p -> data! = x)              /* 顺指针向后查找 */
    {   p = p -> next;
        (*i) + +;
    }
    if(p -> data == x) return(p);
            /* 查找成功返回指向结点的指针,参数 i 返回结点的位置 */
```

```
    else return(NULL);                    /* 查找失败,返回 NULL */
}
```

上述两个算法的时间复杂度均为 $O(n)$。

(3) 插入

在链表中插入或删除一个结点不需要移动大量的元素,只需要修改结点的指针域即可。

① 在单链表的第 i 个位置插入一个结点

若单链表中有 n 个结点,插入位置 i,允许的取值范围为 $1 \leqslant i \leqslant n+1$。在第 i 个位置插入一个新结点,首先需要寻找到第 $i-1$ 个结点,然后将新结点插在第 $i-1$ 个结点之后。假定 *pre* 指向第 $i-1$ 个结点,*cur* 指向新结点,插入就是让新结点的指针域指向原来的第 i 个结点,即 cur -> next = pre -> next,并且修改第 $i-1$ 个结点的指针域,让其指向新结点,即 pre -> next = cur。插入过程如图 2.2.11 所示。

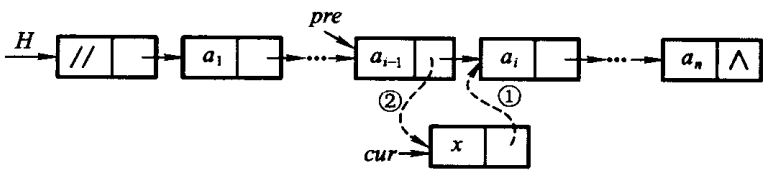


图 2.2.11 在单链表的第 i 个位置插入结点示意图

在带表头结点的单链表 H 中的第 i 个位置插入一个值为 x 的新结点,算法如下:

```
InsertList(LinkList H, int i, ElemType x)
{
    LinkList pre,cur;
    pre = GetElem(H,i-1);                 /* pre 指向第 i-1 个结点 */
    if(! pre)
    {     printf("\n i 值非法");
          exit(1);
    }
    cur = (LinkList)malloc(sizeof(ListNode));     /* 生成新结点 */
    if(! cur)
    {     printf("\n 存储分配失败");
          exit(1);
    }
    cur -> data = x;                              /* 新结点值为 x */
    cur -> next = pre -> next;
```

```
                /* 新结点指针域的值指向原来的第 i 个结点,如图 2.2.11① */
    pre -> next = cur;
                /* 第 i - 1 个结点的指针域的值指向新结点,如图 2.2.11② */
}
```

算法的时间主要耗费在查找操作 GetElem 上,故时间复杂度为 $O(n)$。

② 在一个有序的单链表 H 中插入一个值为 x 的结点,插入后所得单链表仍然有序。

生成新结点后,首先必须找出新结点在链表中的合适位置,然后将它插入到链表中。该算法与建立带头结点的有序单链表的算法思想是一致的。

在一个有序的单链表 H 中插入一个值为 x 的结点,算法如下:

```
InsertOrder( LinkList H,ElemType x)
{
    LinkList cur,pre = H;
    cur = H -> next;
    while( cur&&cur -> data < x)              /* 查找结点在表中的合适位置 */
    {   pre = cur;                            /* pre 指向当前结点的前驱结点 */
        cur = cur -> next;                    /* cur 指向当前结点 */
    }
    cur = ( LinkList) malloc( sizeof( ListNode) );   /* 生成新结点 */
    if( ! cur)
    {   printf( " \n 存储分配失败" );
        exit( 1 );
    }
    cur -> data = x;                  /* 新结点值为 x */
    cur -> next = pre -> next         /* 新结点 cur 插在 pre 所指结点的后面 */
    pre -> next = cur
}
```

注意算法中最后两个语句的次序不可颠倒,否则不能正确插入新结点。算法的时间复杂度为 $O(n)$。

(4) 删除

① 删除单链表的第 i 个结点

若单链表中有 n 个结点,删除第 i 个结点,i 允许的取值范围为 $1 \leqslant i \leqslant n$。类似地,首先需要找到第 $i-1$ 个结点,假定指针 *pre* 指向它,*cur* 指向第 i 个结点,要删除第 i 个结点,只要使第 $i-1$ 个结点的指针指向第 $i+1$ 个结点即可,即 pre -> next = cur -> next。删除过程如图 2.2.12 所示。

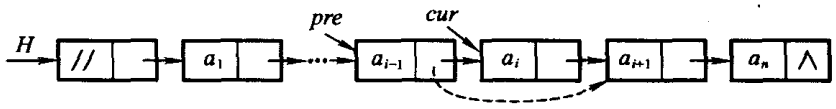


图 2.2.12 删除单链表的第 i 个结点示意图

在带表头结点的单链表 H 中删除第 i 个结点,算法如下:

```
DeleteList1(LinkList H,int i, ElemType *x)
{
    LinkList cur,pre;
    pre = GetElem(H, i-1);                    /*pre 指向第 i-1 个结点*/
    if(pre == NULL || pre -> next == NULL)    /*i 值是否合法*/
    {    printf("\n i 值非法");
         exit(1);
    }
    cur = pre -> next;                        /*cur 指向第 i 个结点*/
    pre -> next = cur -> next;                /*第 i-1 个结点的指针域修改为指向第 i+1
个结点*/
    *x = cur -> data;                         /*第 i 个结点的值由 x 返回*/
    free(cur);                                /*释放第 i 个结点所占空间*/
}
```

② 删除单链表中具有给定值的结点

在单链表中查找具有给定值的结点,找到后执行删除。

在带表头结点的单链表 H 中删除具有给定值的结点,算法如下:

```
DeleteList2(LinkList H, ElemType x)
{
    LinkList p,pre;
    p = H -> next; pre = H;                   /*p 指向第一个结点,pre 指向 p 的前驱*/
    While(p && p -> data! =x)                 /*顺指针向后查找*/
    {   pre = p;                              /*记住前一结点*/
        p = p -> next;                        /*查找下一个结点 */
    };
    if(p -> data == x)
```

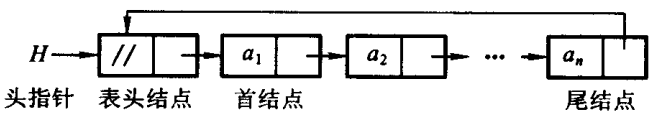
```
        {   pre -> next = p -> next;        /* 删除值为 x 的结点 */
            free(p);                        /* 释放结点所占空间 */
        }
}
```

显然，上述两个算法的时间复杂度也为 $O(n)$。

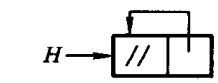
2. 循环链表

循环链表(Circular Linked List)的特点是表中最后一个结点的指针域指向表头结点，整个链表形成一个环。

在单链表中，若将表尾结点的指针域 NULL 改为指向表头结点，就得到了单循环链表。其特点是从表中任意结点出发均可以访问到表中所有的结点。设 H 为单循环链表的头指针，带表头结点的单循环链表如图 2.2.13 所示。



(a) 非空表



(b) 空表

图 2.2.13　带表头结点的单循环链表

在循环链表中没有 NULL 指针，故涉及遍历操作时，其循环中止条件就不再像普通链表那样判断 p -> next 是否为空，而是判断它们是否等于头指针。其他操作算法的实现和普通链表基本一致。

对于设立头指针的单循环链表，通过 H -> next 就可以找到结点 a_1，而要找到结点 a_n，需遍历整个链表。如果不设立头指针而设立尾指针 R(图 2.2.14)，则查找结点 a_1 与 a_n 就可以方便地表示为 R -> next -> next 与 R。因此，根据问题的需要，对于单循环链表，常设立尾指针而不设头指针，这样可以简化某些操作。

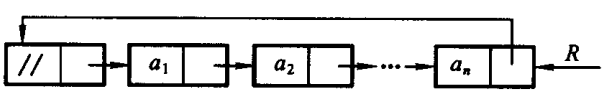


图 2.2.14　仅设尾指针的单循环链表

例如将图 2.2.15(a)所示的两个单循环链表合并成一个链表，合并后的链表如图

2.2.15(b)所示。

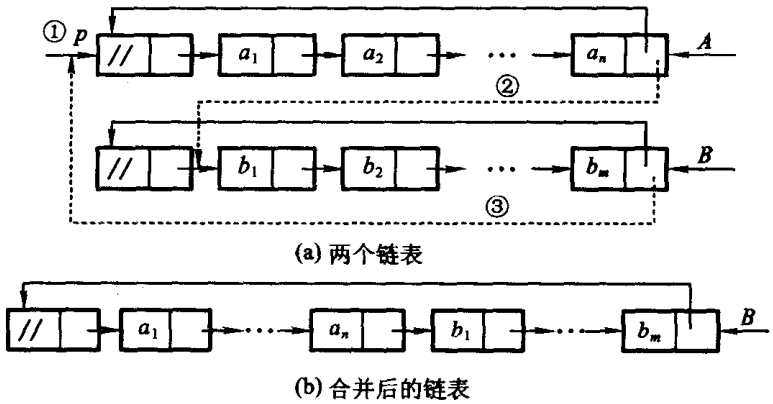


图 2.2.15 合并两个单循环链表

只需将一个表的表尾和另一个表的表头相接,释放掉多余的表头结点即可。下列语句就可完成这一工作。

```
p = A -> next;          /* p 指向 A 表的表头结点,如图 2.2.15(a)① */
A -> next = B -> next -> next;
                        /* A 表的表尾和 B 表的首结点相接,如图 2.2.15(a)② */
free(B -> next);        /* 释放 B 表的表头结点 */
B -> next = p;
                        /* 合并后尾结点的指针指向 A 表表头结点,如图 2.2.15(a)③ */
```

3. 双向链表

从单链表的结构特征可看出,查找一个结点的直接后继容易实现,如果要找一个结点的直接前驱,则需从表头开始扫描才能找到。

双向链表(Double Linked List)可以克服单链表的上述不足,双向链表的每个结点含有两个指针域,一个指向其直接后继,另一个指向其直接前驱。对于双向链表,既可以从表头向表尾方向搜索,也可以从表尾向表头搜索。其结点结构如图 2.2.16 所示。

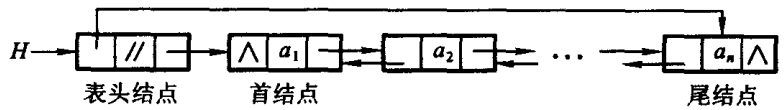
图 2.2.16 双向链表结点结构

类型定义如下:

```
typedef Struct DLNode
```

```
    ElemType data;
    Struct DLNode * prior, * next;
} * DLinkList;
```

带表头结点的双向链表如图 2.2.17 所示，其中(a)图为非空表，表头结点的 *next* 指针域指向首结点，*prior* 指针域指向尾结点，首结点的 *prior* 指针域以及尾结点的 *next* 指针域均为空指针，(b)图为空表，仅有表头结点，其 *prior* 指针域及 *next* 指针域均为空。

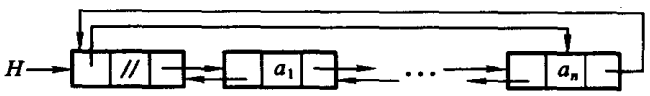


(a) 非空双向链表

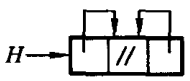
(b) 空双向链表

图 2.2.17　带表头结点的双向链表

双向链表也可以是循环链表，如图 2.2.18 所示。与图 2.2.17 不同的是，图 2.2.18(a)首结点的 *prior* 指针域及尾结点的 *next* 指针域均指向表头结点，图 2.2.18(b)的表头结点 *prior* 指针域及 *next* 指针域均指向表头结点自身。



(a) 非空双循环链表



(b) 空双循环链表

图 2.2.18　带表头结点的双循环链表

设指针 *p* 指向双向链表中的某个结点，该结点的直接前驱与直接后继均存在，根据双向链表中结点结构的特征，则有：

P -> prior -> next = P -> next -> prior = P

即该结点的直接前驱的 *next* 指针域与它的直接后继的 *prior* 指针域均指向该结点自身。

在双向链表中插入或删除一个结点是很方便的，但要注意需同时修改两个方向上的指针，其他操作算法的实现与单链表基本相同。

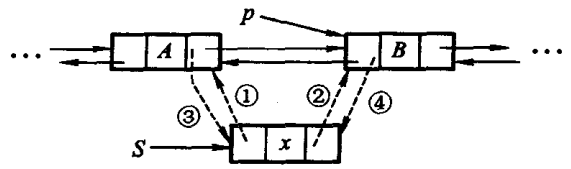


图 2.2.19 在双向链表中插入一个结点

图 2.2.19 表示在双向链表中插入一个结点时指针修改的情况。假定 p 指向双向链表中的结点 B,S 指向将要插入的新结点 x,x 插在结点 A、B 之间,新结点的指针域应是:

```
S -> prior = P -> prior;   /* 新结点的直接前驱指向 A,如图 2.2.19① */
S -> next = P;             /* 新结点的直接后继指向 B,如图 2.2.19② */
```

此时还应该修改结点 A 的直接后继指针与结点 B 的直接前驱指针,即:

```
P -> prior -> next = S;   /* 结点 A 的直接后继指向新结点,如图 2.2.19③ */
P -> prior = S;           /* 结点 B 的直接前驱指向新结点,如图 2.2.19④ */
```

注意,这两个语句的次序不能颠倒。

在带表头结点的双向链表中,在指针 p 所指的结点之前插入值为 x 的新结点,算法如下:

```
Void Insert_Dul(DLinkList  p,     ElemType     x)
{    DlinkList  S;
     S = (DlinkList)malloc(sizeof(DLNode));
     if(! s)
     {    printf("\n 存储分配失败");
          exit(1);
     }
     S -> data = x;
     S -> prior = p -> prior;
     S -> next = p;
     p -> prior -> next = S;
     p -> prior = S;
}
```

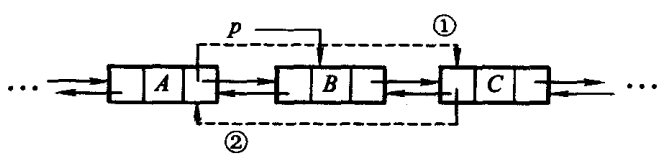


图 2.2.20 在双向链表中删除一个结点

在双向表中删除一个结点如图 2.2.20 所示。假定 p 指向结点 B，要删除该结点，需修改结点 A 的直接后继指针与结点 C 的直接前驱指针，即：

```
p -> prior -> next = p -> next;   /* 结点 A 的直接后继指向 C，如图 2.2.20① */
p -> next -> prior = p -> prior;  /* 结点 C 的直接前驱指向 A，如图 2.2.20② */
```

设 p 指向某结点，在双向表中删除该结点，算法如下：

```
Void Delete_Dul( DLinkList  p,     ElemType  &x)
{
    * x = p -> data;
    p -> prior -> next = p -> next;
    if( p -> next)                          /* 若待删结点不是尾结点 */
        p -> next -> prior = p -> prior;
    free( p);
}
```

4. 链式存储结构的特点

链式存储结构的特点是，链表中结点的存储空间是动态分配的，只要内存尚有空闲空间，就不会产生溢出，不需要预先估计线性表的长度分配连续的存储空间。此外，元素的插入、删除只需要修改相关结点的指针域，不需要移动大量的数据元素。因此，在线性表长度难以估计或者插入、删除较频繁时，宜采用链表作存储结构。如果表的插入和删除主要发生在表的首尾两端，则采用尾指针表示的单循环链表为宜。若希望克服单链表的单向性特点，则采用双向链表比较适宜。

链式存储结构克服了顺序存储结构的缺点，同时也失去了它的优点。它是一种非随机存储结构，对表中结点的访问必须从头指针开始沿着结点的链域往下搜索才能取得。此外，链表中的每个结点，除了数据域外，还要额外设置指针域，存储空间的利用率不如顺序存储结构高。

2.2.5　链式表应用举例

多项式的算术运算几乎已经成为表处理的一个经典问题。本节讨论多项式的相加运算。设一元多项式

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \qquad (式\ 2.2.7)$$

其中 $a_i(i=0,1,2,\cdots,n)$ 是系数。线性表 $(a_0,a_1,\cdots,a_n)$ 可唯一确定多项式 $P_n(x)$。

如果采用顺序存储结构存储该线性表，由于多项式中可能有多项的系数为 0，这种方式可能浪费大量的存储空间。一般采用单链表来表示多项式。在单链表中，每个结点设有 3 个域：系数域 coef，指数域 exp 和链域 next。结点结构如图2.2.21 所示。

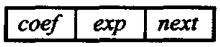


图 2.2.21 单链表结点结构

例如,多项式 $A(x)=3+2x+6x^9+7x^{15}$ 和 $B(x)=5x-6x^9+9x^{12}$ 的单链表形式如图 2.2.22 所示。

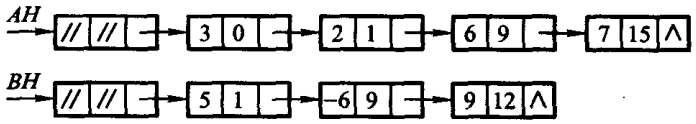


图 2.2.22 多项式 $A(x)$、$B(x)$ 的单链表表示

设 $C(x)=A(x)+B(x)$,根据多项式加法运算规则,若指数相同,系数相加,若和不为 0,则构成 $C(x)$ 中的一项,所有指数不同的项均作为 $C(x)$ 中的一项。多项式 $C(x)=3+7x+9x^{12}+7x^{15}$,其单链表形式如图 2.2.23 所示。

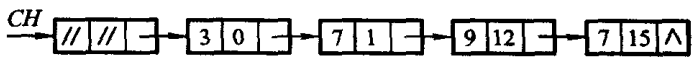


图 2.2.23 多项式 $C(x)$ 的单链表表示

两个多项式相加就变成由单链表 *AH*,*BH* 求单链表 *CH* 了。假设单链表是以指数值的递增次序排列的,指针 *pa*,*pb* 分别指向 *AH*,*BH* 当前正被考察的两个结点,比较两个结点的指数域,有下列 3 种情况:

- pa -> exp < pb -> exp,则将 *pa* 所指的结点插入到 *CH* 链表中,移动指针 *pa*,指向下一结点。
- pa -> exp > pb -> exp,则将 *pb* 所指的结点插入到 *CH* 链表中,移动指针 *pb*,指向下一结点。
- pa -> exp = = pb -> exp,则将两个结点中的系数相加,若和不为 0,则插入到 *CH* 链表中,移动指针 *pa*,*pb*,指向下一结点。

重复上述过程,直至 *pa* 或 *pb* 的值为 NULL。当其中一个为空时,说明有一个表的结点已经处理完,只要将另一个表的剩余部分链接在 *CH* 链表之后即可。

多项式相加程序如下:

```
#include "stdio.h"
#include "stdlib.h"
```

```
#define NULL 0

typedef struct node
{    float coef;
     int exp;
     struct node * next;
} ListNode, * LinkList;

LinkList InitList( )
{   /* 构造一个带表头的空表 */
    LinkList H;
    H = (LinkList) malloc( sizeof( ListNode) );
    if( ! H)
    {    printf( " \n 存储分配失败" );
         exit( 1 );
    }
    H -> next = NULL;
    return( H );
}

InsertTerm( LinkList H,    float c,    int e)
{    /* 在有序的多项式链表中,插入一项系数为 c,指数为 e 的新结点 */
     LinkList cur,pre = H;
     cur = H -> next;
     while( cur&&cur -> exp < e)                    /* 查找新结点应在的位置 */
     {    pre = cur;
          cur = cur -> next;
     }
     cur = (LinkList) malloc( sizeof( ListNode) );
     if( ! cur)
     {    printf( " \n 存储分配失败" );
          exit( 1 );
     }
     cur -> exp = e;
```

```
        cur -> coef = c;
        cur -> next = pre -> next;                    /* 新结点插在 pre 结点后面 */
        pre -> next = cur;
    }

    LinkList CreatePoly( )
    {    /* 输入 n 项的系数和指数,建立表示一元多项式的有序链表 */
        LinkList H,pre,cur;
        int n,i,exp;
        float coef;
        H = InitList( );
        printf( " \n enter n:" );
        scanf( " % d" ,&n);
        for( i = 1;i <= n;i + + )
        {     printf( " Enter coef and exp:" );
            scanf( " % f % d" ,&coef,&exp);
            InsertTerm (H,coef,exp);
        }
        return(H);
    }

    LinkList append( LinkList p, LinkList pc)
    {    /* 将结点 p 插到结点 pc 之后 */
        pc -> next = p;
        pc = p;
        return( pc);
    }

    LinkList Polyadd( LinkList A, LinkList B)
    {   /* 求和多项式 C(x) = A(x) + B(x) */
        LinkList C,pa,pb,pc,p;
        pa = A -> next;
        pb = B -> next;
        C = A;          /* 利用 A 链表的表头结点作为和多项式 C(x)的表头结点 */
```

```
pc = C;                                          /* pc 指向 C 链表中的尾结点 */
while(pa&&pb)
{   if(pa -> exp < pb -> exp)
    {   pc = append(pa,pc);
                /* 结点 pa 作为和多项式的一项,插入到结点 pc 之后 */
        pa = pa -> next;
    }
    else
    if(pa -> exp = = pb -> exp)                  /* 指数相同 */
    {   pa -> coef = pa -> coef + pb -> coef;
        if(pa -> coef)
        /* 和的系数非 0,作为和多项式的一项插入到结点 pc 之后 */
        {     pc = append(pa,pc);
              pa = pa -> next;
        }
        else                              /* 和的系数为 0,释放结点 pa */
        {   p = pa;
            pa = pa -> next;
            free(p);
        }
        p = pb;                  /* 不论和的系数是否为 0,都要释放结点 pb */
        pb = pb -> next;
        free(p);
    }
    else                              /* pa -> exp > pb -> exp */
    {   pc = append(pb,pc);
                /* 结点 pb 作为和多项式的一项插入到结点 pc 之后 */
        pb = pb -> next;
    }
}   /* while */
if(pa) pc = append(pa,pc);          /* 链接 A 中剩余结点 */
else pc = append(pb,pc);            /* 链接 B 中剩余结点 */
free(B);                            /* 释放 B 表表头结点 */
return(C);
```

```
}
void main()
{   LinkList A,B,C,p;
    printf("\n");
    A = CreatePoly();
    B = CreatePoly();
    C = Polyadd(A,B);
}
```

2.3 栈和队列

栈和队列是两种特殊的线性表,其基本操作是线性表操作的子集。对于线性表来说,可以在它的任意位置插入和删除元素,而栈和队列只允许在表的一端或两端进行插入或删除,它们是操作受限的线性表,故又称为限定性的数据结构。栈和队列在计算机的各种软件系统中有着广泛的应用。

2.3.1 栈的定义及基本操作

栈(Stack)是只允许在表的一端插入和删除元素的线性表。允许进行插入和删除的一端称为栈顶(top),而另一端称为栈底(bottom)。不含元素的栈称为空栈。插入一个新元素到栈中叫做进栈(push),删除栈顶元素叫做出栈(pop)。

假设将元素按 $a_1,a_2,\cdots,a_n$ 的次序进栈,如图 2.3.1 所示。其中 a_1 称为栈底元素,a_n 称为栈顶元素。出栈的次序是 $a_n,a_{n-1},\cdots,a_1$。栈中元素的进出原则是后进先出(Last In First Out),因此栈又被称为后进先出(LIFO)表。

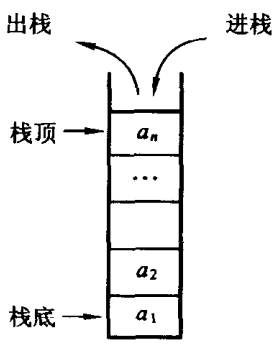


图 2.3.1 栈示意图

栈在日常生活中也可见到，如一叠盘子或一叠书，若规定从这叠物体中取出或者放入一件都只能在顶端进行，那它就是一个栈。

栈的基本操作有以下几种。

(1) InitStack(S)

构造一个空栈 S。

(2) StackEmpty(S)

判断栈 S 是否为空栈。若 S 为空栈，返回 1，否则返回 0。

(3) Push(S,x)

进栈。若栈 S 不满，则将元素 x 插入栈顶。

(4) Pop(S)

出栈。若栈 S 非空，则将栈顶元素删除，并返回该元素。

(5) GetTop(S)

读取栈顶元素。若栈 S 非空，返回栈顶元素，但并不删除该元素。

栈是一种线性表，因此线性表的顺序存储结构和链式存储结构都适用于栈，类似于顺序表和链表，栈也可分为顺序栈和链式栈。

2.3.2　栈的顺序存储结构

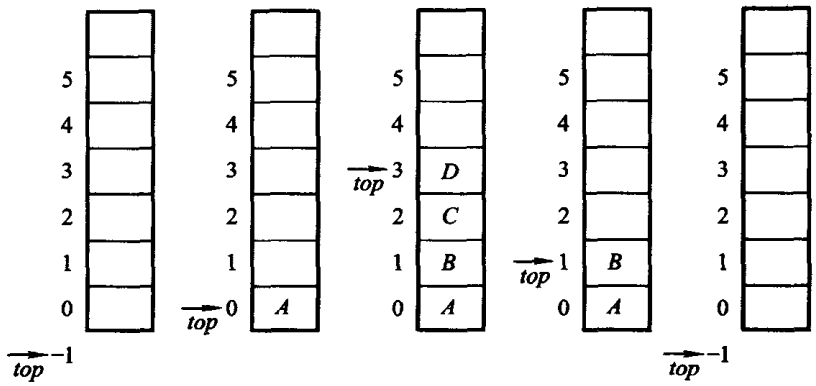
栈的顺序存储结构通常用一个一维数组和一个整型变量实现，利用数组存储栈中的所有元素，利用整型变量指示栈顶元素的下标位置。顺序栈的类型可定义如下：

```
#define MAXSIZE 100                    /* 设栈空间最多容纳 100 个元素 */
typedef char ElemType;                 /* 设 ElemType 为字符型 */
typedef Struct
{   ElemType Stack[MAXSIZE];
    int  top;
}SeqStack;
```

在顺序栈中，一般以 $top == 0$ 表示空栈，在 C 语言描述中，由于数组的下标约定从 0 开始，故 $top == -1$ 时，表示栈空。向栈中插入一个新元素(进栈时)，首先使 top 加 1，以指示新的栈顶位置，然后再把元素赋值到这个位置。删除栈顶元素(出栈)时，首先取出栈顶元素，然后使 top 减 1，指示新的栈顶位置。若 top 已经指向了 MAXSIZE - 1，表示栈满。向一个满栈插入元素和从一个空栈删除元素会产生上溢或下溢，上溢是一种出错状态，应该避免，下溢则常用来作为程序控制转移的条件。图 2.3.2 说明了在顺序栈中进栈和退栈时，栈中元素和栈顶指针的关系。

在顺序栈中实现栈的 5 种基本操作，具体算法有以下几种。

(1) 构造一个空栈



(a) 空栈 (b) A 进栈 (c) B、C、D 依次进栈 (d) D、C 出栈 (e) B、A 出栈

图 2.3.2 栈顶指针和栈中元素之间的关系

```
viod InitStack(SeqStack *S)
{    /* 初始化栈 S ,置为空栈 */
     S -> top = -1;
}
```

(2) 判断一个栈是否为空

```
int StackEmpty(SeqStack *S )
{    /* 判断 S 是否为空栈,若空,则返回 1,否则返回 0 */
     return S -> top = = -1;
}
```

(3) 进栈

```
viod Push(SeqStack *S, ElemType x)
{    /* 向栈 S 中插入元素 x */
    if(S -> top = = MAXSIZE - 1)
    {   printf ("\n 上溢");
        exit(1);
    }
    + +S -> top;                    /* 栈顶指针加 1 */
    S -> Stack[S -> top] = x;       /* x 进栈 */
}
```

(4) 出栈

```
ElemType pop(SeqStack *S)
{   /* 删除栈顶元素并将其返回 */
```

```
    if(StackEmpty(S))
    {   printf("\n 下溢");
        exit(1);
    }
    return S->Stack[S->top--];      /* 返回栈顶元素,栈顶指针减 1 */
}
```

(5) 取栈顶元素

```
ElemType GetTop(SeqStack *S)
{   /* 若栈 S 非空,返回栈顶元素,但并不删除该元素 */
    if(StackEmpty(S))
    {   printf("\n 空栈");
        exit(1);
    }
    return S->Stack[S->top];
}
```

如果一个程序同时使用多个顺序栈,为了不出现因上溢而出错,就必须给每个栈预先分配一个较大的空间,在有些情况下这样会造成系统空间的紧张,此外,还可能出现在某一个栈发生上溢的同时,而其余的栈未用的空间还很多的情形如果使多个栈共用一个足够大的数组空间,就可以相互调节,这样既节约了存储空间,又降低了发生上溢的概率,这就是栈的共用。

多栈共用中最常见的是两个栈的共用。假设一个程序中需设两个栈,可以令其共享一维数组空间 $S[m]$,可以将两个栈的栈底分别设在数组的两端,两个栈各自向中间延伸(图 2.3.3)。由于两个栈顶均为动态变化,可互补余缺,因此使得每个栈的最大空间均大于 $m/2$。只有当整个空间都被两个栈占满时,才会发生上溢。两个栈共享空间的示意图如图 2.3.3 所示。

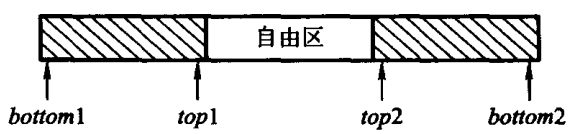


图 2.3.3 两个栈共享空间示意图

如果有两个以上的栈共享向量空间,这时必须为每个栈设置栈顶指针与栈底指针,当某个栈上溢时,如果其余栈中尚有未用空间,则需移动元素才能为产生上溢的栈腾出空间,故其效率较低。在这种情况下,采用顺序存储结构的动态分配或链式栈为宜。

2.3.3 栈的链式存储结构

栈的链式存储结构一般是通过单链表实现的，单链表中每个结点表示栈中的一个元素，由于只是在链表头部进行操作，故链式栈没有必要设置表头结点。栈顶指针就是链表的头指针，假设将元素按 $a_1,a_2,\cdots,a_n$ 的顺序进栈，则 a_1 为栈底元素，a_n 为栈顶元素，链式栈的示意图如图 2.3.4 所示。

向一个链式栈插入元素时，把该元素插入到栈顶，使该元素结点的指针域指向原来的栈顶结点，而栈顶指针则修改为指向该元素结点，使该结点成为新的栈顶结点。

从一个链式栈中删除元素时，取出栈顶元素后，使栈顶指针指向原栈顶结点的直接后继结点。链式栈的类型说明及 5 种基本操作说明如下。

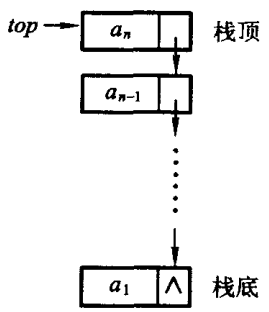


图 2.3.4 链式栈示意图

```
typedef char ElemType;
typedef struct stacknode
{    ElemType data;
     struct stacknode * next;
} StackNode;

typedef struct
{ StackNode   * top;              /* 栈顶指针 */
} LinStack;

void InitStack( LinStack * s)
{     /* 构造一个空栈 */
      s -> top = NULL;
```

```
}

int StackEmpty(LinStack *s)
{   /* 判断一个栈是否为空,若是,则返回1,否则返回0 */
    return s->top==NULL;
}

void Push(LinStack *s, ElemType x)
{   /* 进栈 */
    StackNode *p;
    p=(StackNode *)malloc(sizeof(StackNode));
    if(!p)
    {   printf("\n 存储分配失败 ");
        exit(1);
    }
    p->data=x;
    p->next=s->top;          /* 新结点插入链式栈栈顶 */
    s->top=p;                /* 新结点作为栈顶结点 */
}

ElemType Pop(LinStack *s)
{   /* 出栈 */
    ElemType x;
    StackNode *p;
    p=s->top;
    if(StackEmpty(s))
    {   printf("\n 下溢 ");
        exit(1);
    }
    x=p->data;               /* 保存栈顶结点数据 */
    s->top=p->next;          /* 删除栈顶结点,修改栈顶指针 */
    free(p);
    return x;
}
```

```
ElemType GetTop(LinStack *s)
{   /* 取栈顶元素 */
    if(StackEmpty(s))
    {   printf("\n 下溢");
        exit(1);
    }
    return s -> top -> data;
}
```

2.3.4 栈的应用举例

用计算机解决后进先出的有关问题时,很自然地就会使用栈。栈广泛应用于各种软件系统中。例如编译系统中表达式的计算以及函数的递归调用等都要利用栈来实现。下面讨论几个栈应用的典型例子。

1. 数制转换

由计算机基础知识可知,把一个十进制整数 n 转换为 r 进制数,转换方法是采用除基数 r 取余法。例如,若十进制整数为 83,把它转换为八进制数的过程如图 2.3.5 所示。

8	83	余数
8	10	3
8	1	2
	0	1

图 2.3.5 十进制数 83 转换为八进制数的过程

在转换过程中,由低到高得到 r 进制数中的每一位数字,输出时需要由高到低输出每一位。最后得到的八进制数为 $(123)_8$,所以此问题适合利用栈来解决。

十进制数转换为八进制数,程序如下:

```
#define MAXSIZE 100
typedef int ElemType;
typedef struct
{   ElemType stack[MAXSIZE];
    int top;
}SeqStack;
```

```
void InitStack(SeqStack *s)
{
    s->top=-1;
}

int StackEmpty(SeqStack *s)
{
    return s->top==-1;
}

void Push(SeqStack *s, ElemType x)
{
    if(s->top==MAXSIZE-1)
    {   printf("\n 上溢");
        exit(1);
    }
    ++s->top;
    s->stack[s->top]=x;
}

ElemType Pop(SeqStack *s)
{
    if(StackEmpty(s))
    {   printf("\n 下溢");
        exit(1);
    }
    return s->stack[s->top--];
}

void conversion(long num, int r)
{   /* 对一个非负十进制整数,输出与其等值的八进制数 */
    SeqStack s;
    int k;
    ElemType x;
```

```
    InitStack (&s);
    While (num)
    {   k = num % r;
        Push(&s,k);
        num/ = r;
    }
    printf("\n");
    while(! StackEmpty (&s))
    {   x = Pop(&s);
        printf(" %d",x);
    }
}

main()
{   long n;
    int r;
    printf("\n enter n,r");
    scanf("%ld,%d",&n,&r);
    conversion(n,r);
}
```

2. 算术表达式的计算

编译系统中表达式的计算是通过栈来实现的。为了叙述简便,只考虑算术表达式的计算,并假定表达式只含有 +、-、*、/ 4 种运算及圆括号。在表达式的两边虚设一个分界符"#",表示表达式的开始与结束。四则运算的规则是:先乘除后加减;同级别时从左算到右;先括号内,后括号外。

按此规则,下面表达式的求值顺序为:

1 +2 * (3 +4) -6/2
=1 +2 * 7 -6/2
=1 +14 -6/2
=15 -3
=12

为了使计算机产生正确的求值顺序,编译程序使用的方法之一是利用运算符之间的优先级关系对表达式进行处理。表 2.3.1 定义了 +、-、*、/、(、)和分界符"#"之间的优先级关系,$\theta 1$ 和 $\theta 2$ 表示表达式中相继出现的运算符。

表 2.3.1　运算符优先级关系表

θ1 \ θ2	+	-	*	/	(	)	#
+	>	>	<	<	<	>	>
-	>	>	<	<	<	>	>
*	>	>	>	>	<	>	>
/	>	>	>	>	<	>	>
(	<	<	<	<	<	=	
)	>	>	>	>		>	>
#	<	<	<	<	<		=

为了实现运算符优先算法，还需设置两个工作栈。一个称为操作数栈 OPND，用以存放操作数或运算结果；另一个称为运算符栈 OPTR，用以存放运算符。算法的基本思想是：

首先，置操作数栈为空栈，表达式分界符“#”为运算符栈的栈底元素。

然后，从左到右扫描表达式，若是操作数则进操作数栈，若是运算符，则和运算符栈的栈顶运算符比较优先级，若前者大，则该运算符进运算符栈，反之，则栈顶运算符退栈，并形成一条运算的指令。该指令中的运算符为出栈的运算符，操作数栈栈顶的两个元素出栈作为运算对象，运算结果进入操作数栈，如此处理，直到整个表达式求值完毕。

为问题简单起见，假定操作数都只有一位数字。另外，初始化栈、进栈、退栈、读取栈顶元素等函数已在前面讨论过，为节省篇幅，在此未列出。算法如下：

```
typedef struct
{     float stack [MAXLENGTH];
      int top;
}SeqStack1;                          /* 操作数栈 */
typedef struct
{     char stack [MAXLENGTH];
      int top;
}SeqStack2;                          /* 运算符栈 */

int op(char x)
{     int i;
      switch (x)
      {     case '+': i=0;     break;
```

```
            case '-': i=1;      break;
            case '*': i=2;      break;
            case '/': i=3;      break;
            case '(': i=4;      break;
            case ')': i=5;      break;
            case '#': i=6;      break;
            default : i= -1;
        }
        return(i);
}

char precede(char x1,char x2)
{    /* 完成运算符优先级的比较    */
     char result;
     int i,j;
     char pre[7][7]={   {'>','>','<','<','<','>','>'},
                        {'>','>','<','<','<','>','>'},
                        {'>','>','>','>','<','>','>'},
                        {'>','>','>','>','<','>','>'},
                        {'<','<','<','<','<','=',' '},
                        {'>','>','>','>',' ','>','>'},
                        {'<','<','<','<','<',' ','='}
                    };
     return(pre[op(x1)][op(x2)]);
}

float evalu_expre()
{
     char c,theta;
     float a,b,z;
     SeqStack1 opnd;
     SeqStack2 optr;
     InitStack2 (&optr);      Push2 (&optr,  '#');
     InitStack1(&opnd);       c=getchar();
```

```
while(c! ='#'||GetTop2(&optr)! ='#')
{   if(op(c)== -1)
    {   Push1(&opnd,c);      c=getchar();}      /* 操作数进操作数栈 */
    else
    switch (precede (GetTop2(&optr),c))
    {     case '<': Push2(&optr,c);   c=getchar();   break;
                                          /* 运算符进运算符栈 */
          case '=': c=Pop2(&optr);    c=getchar();   break;  /* 脱括号 */
          case '>': theta=Pop2(&optr);                       /* 出栈   */
                    b=Pop1(&opnd);      a=Pop1(&opnd);
                    switch(theta)
                    {     case '+': z=a+b;      break;
                          case '-': z=a-b;      break;
                          case '*': z=a*b;      break;
                          case '/': if(b)   { z=a/b;      break;  }
                                    else
                                    {     printf("\n 除数为0 ");
                                          exit(1);
                                    }
                    }
                    Push(&opnd,z); break;  /* 运算结果进操作数栈 */
    }
}
return GetTop1(&opnd);
}
```

利用上述算法对算术表达式 1+2*(3+4)-6/2 求值,操作过程如表 2.3.2 所示。

表 2.3.2　算术表达式 1+2*(3+2)-6/2 的求值过程

步骤	OPTR 栈	OPND 栈	输入字符	主要操作
1	#		1+2*(3+4)-6/2#	Push(OPND,1)
2	#	1	+2*(3+4)-6/2#	Push(OPTR,'+')
3	#+	1	2*(3+4)-6/2#	Push(OPND,2)

续表

步骤	OPTR 栈	OPND 栈	输入字符	主要操作
4	# +	1 2	* (3 + 4) - 6/2#	Push(OPTR,' * ')
5	# + *	1 2	(3 + 4) - 6/2#	Push(OPTR,'(')
6	# + * (	1 2	3 + 4) - 6/2#	Push(OPND,3)
7	# + * (	1 2 3	+ 4) - 6/2#	Push(OPTR,' + ')
8	# + * (+	1 2 3	4) - 6/2#	Push(OPND,4)
9	# + * (+	1 2 3 4	) - 6/2#	Operate(3 + 4)
10	# + * (	1 2 7	) - 6/2#	Pop(OPTR)
11	# + *	1 2 7	- 6/2#	Operate(2 * 7)
12	# +	1 14	- 6/2#	Operate(1 + 14)
13	#	15	- 6/2#	Push(OPTR,' - ')
14	# -	15	6/2#	Push(OPND,6)
15	# -	15 6	/2#	Push(OPTR,'/')
16	# - /	15 6	2#	Push(OPND,2)
17	# -	15 6 2	#	Operate(6/2)
18	# -	15 3	#	Operate(15 - 3)
19	#	12	#	return 12

3. 栈与递归的实现

递归是程序设计中一个强有力的工具,它可使问题的描述和求解变得简洁和清晰。如非负整数 n 的阶乘可递归定义为

$$n! = \begin{cases} 1, & n = 0 \\ n * (n-1)!, & n > 0 \end{cases}$$

相应的递归算法是:

```
int fact( int n)
{    if( n = = 0)      return 1;
     else              return   n * fact( n-1);
}
```

编译系统在调用一个函数时,会为被调用函数构造一个包括返回地址、参数表、局部变量等信息组成的活动记录(如图 2.3.6 所示),并将其压入到由系统提供的运行时刻栈的栈顶,然后将程序控制权转移到被调用函数,当被调用函数执行完毕时,系统将运行时刻栈栈

顶的活动记录退栈，并根据退栈的活动记录中所保存的返回地址将程序的控制权转移给调用者继续执行。递归函数的执行是一种严格的、嵌入式的后进先出的结构。使用运行时刻栈是实现递归函数的最简单、最自然的管理技术，每进入一层递归，就产生一个新的活动记录压入栈顶，每退出一层递归，就从栈顶弹出一个活动记录。栈顶活动记录是当前递归活动的工作记录。

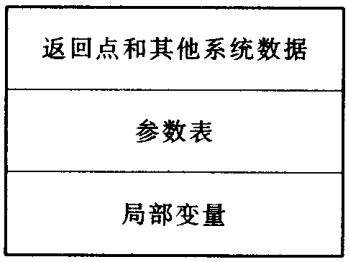


图 2.3.6 活动记录示意图

2.3.5 队列的定义及基本操作

队列(Queue)是一种只允许在表的一端进行插入而在另一端进行删除的线性表。允许插入的一端称为队尾(rear)，允许删除的一端称为队头(front)。不含元素的队列称为空队列。

假设将元素按 $a_1, a_2, \cdots, a_n$ 的顺序进入队列，如图 2.3.7 所示。退出队列也只能按照这个次序依次退出。这和日常生活中排队是一致的，新来的成员总是加入队尾，离开的成员总是队列头元素(不允许中途离队)。队列中元素的进出原则是先进先出(First In First Out)，因此队列又被称为先进先出(FIFO)表。

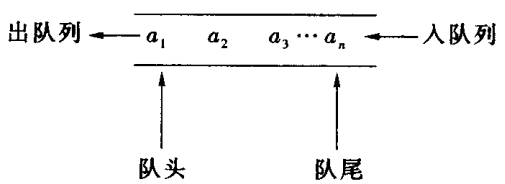


图 2.3.7 队列示意图

队列的基本操作有以下几种。

(1) InitQueue(Q)

构造一个空队列 Q。

(2) QueueEmpty(Q)

判断队列 Q 是否为空。若 Q 为空队列，则返回 1，否则返回 0。

(3) EnQueue(Q,X)

入队。若队列 Q 不满，则将元素 X 插入 Q 的队尾。

(4) DeQueue(Q)

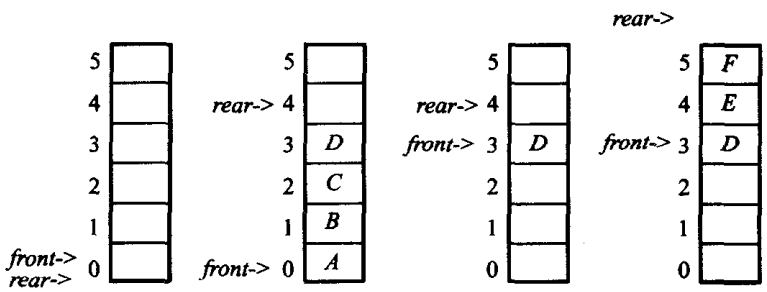
出队。若队列 Q 非空，则删去 Q 的队头元素，并返回该元素 。

(5) GetHead(Q)

读取队头元素。若队列 Q 非空，则返回队头元素，但并不删除该元素。

2.3.6 队列的顺序存储结构

队列的顺序存储结构称为顺序队列，和顺序栈类似，顺序队列通常用一个一维数组来存放队列中的元素。此外，尚需设置两个指针 *front* 和 *rear*，它们分别指示队头元素和队尾元素的位置，这里约定队列初始化时，它们的值均置为 0。入队时，将新元素插入 *rear* 所指的位置，然后将 *rear* 加 1；出队时，删除 *front* 所指的元素，然后将 *front* 加 1。在非空队列中，*front* 始终指向队头元素，而 *rear* 始终指向队尾元素的下一个位置，如图 2.3.8 所示。



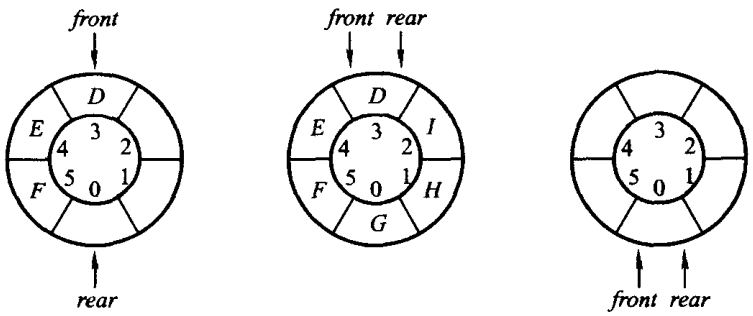
(a) 空队列 (b) A、B、C、D 入队 (c) A、B、C 出队 (d) E、F 入队

图 2.3.8 顺序队列中元素和头尾指针的关系

假设当前为队列分配的最大空间为 6，当队列处于图 2.3.8(d)所示的状态时，如果还有新元素请求入队，就会发生“溢出”。但这时队列的实际可用空间并未占满。为克服上述假溢出，可以将顺序队列想像为一个首尾相接的环状空间，如图 2.3.9(a)所示，称之为循环队列。在循环队列中进行出队、入队操作时，头尾指针仍要加 1，只不过当头尾指针指向上界时，其加 1 的操作结果是指向下界 0。假设当前循环队列最多能容纳 MAXSIZE 个元素，这种循环意义下的加 1 操作可以描述为：

```
i = (i + 1) % MAXSIZE;        /* i 表示 front 或 rear */
```

在如图 2.3.9(a)所示的循环队列中，如果相继插入新元素 G，H，I，则队列空间均被占满，如图 2.3.9(b)所示。此时亦存在关系式 *front* = *rear*，因此仅凭头尾指针是否相等无法判别队列空间是“空”还是“满”，解决此问题的方法之一是少用一个元素的空间，约定入队前，若尾指针加 1 后等于头指针，则认为队列满。



(a)一般情况 (b)队列满时 (c)空队列

图 2.3.9 循环队列示意图

下面采用这种方法实现循环队列上的几种基本操作。循环队列类型定义如下:

```
#define MAXSIZE 100                    /* 假设循环队列最多容纳 100 个元素 */
typedef char ElemType;                 /* 假设 ElemType 为字符型 */
typedef struct
{   ElemType data[MAXSIZE];
    int      front;
    int      rear;
}CirQueue;
```

在循环队列中实现 5 种基本操作,具体算法有以下几种。

(1) 构造一个空队列

```
void InitQueue(CirQueue *q)
{
    q->front = q->rear = 0;
}
```

(2) 判断队列是否为空

```
int QueueEmpty(CirQueue *q)
{
    if(q->rear = =q->front) /* 队列为空返回 1,否则返回 0 */
        return(1);
    else return(0);
}
```

(3) 入队

```
void EnQueue(CirQueue *q, ElemType x)
{
```

```
    if((q->rear+1)%MAXSIZE==q->front)
    {   printf("\n 上溢");                    /* 队满上溢 */
        exit(1);
    }
    q->data[q->rear]=x;                      /* 新元素插入队尾 */
    q->rear=(q->rear+1)%MAXSIZE;             /* 尾指针加1 */
}
```

(4) 出队

```
ElemType DeQueue(CirQueue *q)
{
    ElemType x;
    if(QueueEmpty(q))
    {   printf("\n 下溢");                    /* 队空下溢 */
        exit(1);
    }
    x=q->data[q->front];                     /* 取出队头元素 */
    q->front=(q->front+1)%MAXSIZE;           /* 头指针加1 */
    return(x);
}
```

(5) 读取队头元素

```
ElemType GetHead(CirQueue *q)
{
    ElemType x;
    if(QueueEmpty(q))
    {   printf("\n 下溢");                    /* 队空下溢 */
        exit(1);
    }
    x=q->data[q->front];                     /* 读取队头元素 */
    return(x);
}
```

2.3.7 队列的链式存储结构

队列的链式存储结构简称为链队列。由于需要在单链表的表头进行删除、在表尾进行插入,因此需要使用两个指针,队头指针和队尾指针。和线性表的单链表一样,为了操作方

便,可以给链队列增加一个表头结点,并令队头指针指向表头结点,队尾指针指向队列的最后一个结点。链队列的类型定义如下:

```
typedef char ElemType                /* 假设 ElemType 为字符型 */
typedef struct QNode                 /* 链队列中的结点类型 */
{    ElemType        data;
     struct QNode    * next;
} QNode, * QueuePtr;

typedef struct
{    QNode  * front;                 /* 队头指针 */
     QNode  * rear;                  /* 队尾指针 */
} LinQueue;
```

链队列示意图如图 2.3.10 所示,图中 q 为 LinQueue 型的指针。

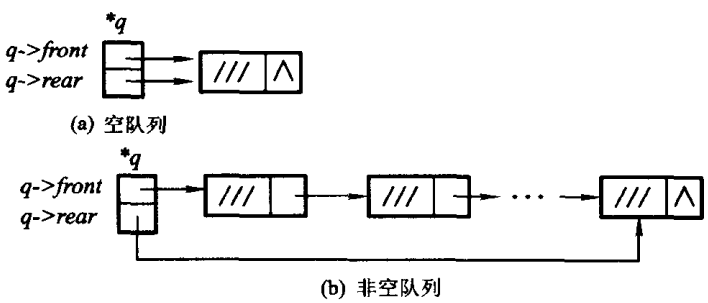


图 2.3.10　链队列示意图

在链队列中可实现 5 种基本操作,具体算法如下:

(1) 构造一个空队列

```
void InitQueue( LinQueue  * q)
{
     q -> front = q -> rear = ( QueuePtr) malloc( sizeof( QNode) );/ * 建立表头结点 */
     if ( ! q -> front)
     {    printf( " \n 存储分配失败" );
          exit( 1 );
     }
     q -> front -> next = NULL;
}
```

(2) 判断队列是否为空

```
int QueueEmpty(LinQueue *q)
{
    if(q->front==q->rear&& q->front->next==NULL)
        return(1);
    else return(0);
}
```

(3) 入队

```
void EnQueue(LinQueue *q, ElemType x)
{   QNode *p;
    p=(QueuePtr)malloc(sizeof(QNode));
    if(!p)
    {   printf("\n 存储分配失败");
        exit(1);
    }
    p->data=x;
    p->next=NULL;
    q->rear->next=p;                    /* 新结点插入队尾 */
    q->rear=p;                          /* 队尾指针指向新结点 */
}
```

(4) 出队

```
ElemType DeQueue(LinQueue *q)
{   QNode *p;
    ElemType x;
    if(QueueEmpty(q))
    {   printf("\n 队空下溢");
        exit(1);
    }
    p=q->front->next;
    x=p->data;                          /* 取出队头元素 */;
    q->front->next=p->next;             /* 修改队头指针 */
    if(q->rear==p)
        q->rear=q->front
                /* 若取出队头元素后,原队列为空,修改队尾指针 */
```

```
    free(p);
    return(x);
}
```

(5) 读取队头元素

```
ElemType GetHead(LinQueue *q)
{   QNode *p;
    ElemType x;
    if(QueueEmpty(q))
    {   printf("\n 队空下溢");
        exit(1);
    }
    p = q -> front -> next;
    x = p -> data;                    /* 取出队头元素 */
    return(x);
}
```

除了栈和队列之外,还有一种限定性数据结构是双端队列(Double_Ended Queue)。如果插入和删除操作能够在线性表的两端进行,则称这种线性表为双端队列。由于在两端进行插入和删除,因此需要设立两个端点,如图 2.3.11 所示。双端队列可以看成是栈底连在一起的两个栈,与前面讨论的两个栈共享存储空间不同的是,这里两个栈的栈顶指针是往两端延伸的。

双端队列可以采用顺序存储结构,也可以用双向链表作为双端队列的存储结构。允许在一端进行插入和删除,另一端只允许插入的双端队称为输出受限双端队;允许在一端进行插入和删除,另一端只允许删除的双端队称为输入受限双端队。

双端队列在实际中不及栈和队列应用广泛,故在此不作详细的讨论。

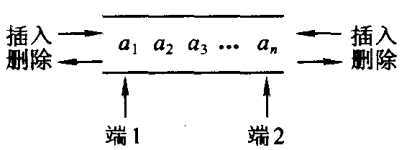


图 2.3.11 双端队列示意图

2.3.8 队列应用举例

队列的应用很广泛,只要问题满足先进先出原则,均可使用队列作为其数据结构。例如操作系统中的各种资源请求排队和各种数据缓冲区的管理以及各种应用系统中的文件规划、文件模拟等。

排队是现实生活中常见的现象,作为简单模拟,假定有一个服务窗口,该窗口在某个时刻只能接待一个顾客,假设为一个顾客服务所需的时间为定值 *sevetime*(由程序输入),在顾客人数众多时需要在窗口前排队,队列用 *q* 表示。顾客到达的时间是随机的,顾客的随机到达按如下方式处理:每单位时间内产生一个随机数 *arrive*(0 < *arrive* < 1),如果 *arrive* < *prob*(*prob* 由程序输入,0 < *prob* < 1),则计为有一人到达。顾客用整数表示,其值为该顾客进入队列时间。时间用 *time* 表示,其初值为 0,假设 *time* 每变化一个单位对应实际时间 1 分钟。当顾客到达时,若服务窗口空闲,顾客的等待时间为 0,否则顾客的等待时间为服务窗口开始为顾客服务的时间减去顾客到达的时间。

程序模拟在时间段 *simutime*(由程序输入)内总服务人数 *cus-num* 以及每个顾客平均排队所需的等待时间。程序中用 *busy* 表示服务窗口为队首顾客服务,在 *time* 时间时还需要多少时间才能完成服务工作。*sumwaits* 表示所有顾客总的等待时间。程序如下:

```
#include "stdlib.h"
#define MAXSIZE 100
typedef int ElemType;
typedef struct
{   ElemType data[MAXSIZE];
    int      front;
    int      rear;
}CirQueue;

main()
{   CirQueue q;
    int  time,servetime,simutime;
    int  busy=0, sumwaits=0, cus_num=0;
    float prob,arrive,aver;
    printf("\n 输入特定值 prob :");
    scanf("%f",&prob);
    printf("\n 输入一个顾客所需的服务时间:");
    scanf("%d",&servetime);
    printf("\n 输入模拟时间:");
    scanf("%d",&simutime);
    InitQueue(&q);
    for(time=0; time<=simutime; time++)
    {
```

```
        arrive = 1.0 * (rand()%10000)/10000;          /* 产生随机数 */
        if(arrive < prob) EnQueue(&q,time);
                          /* 若随机数小于 prob,则为顾客到达,入队 */
        if(busy >0) busy - -; /* 窗口有顾客,顾客所需的服务时间自减 */
        if(busy = =0 && ! QueueEmpty(&q))          /* 窗口空闲且队列非空 */
        {   busy = servetime;                      /* 窗口为顾客服务 */
            cus_num + +;                           /* 顾客数加 1 */
            sumwaits + =(time - DeQueue(&q));      /* 顾客等待时间累加 */
        }
    }
    if(cus_num = =0) aver = 0.0;
    else aver = 1.0 * sumwaits/cus_num;
    printf("\n %d 总服务人数,每个顾客平均等待%4.1f 分钟。", cus_num, aver);
}
```

程序中用的进队、出队、判断队列是否为空的函数前面已经讨论过,这里未列出。

2.4　树

树形结构是一类重要的非线性结构,简称为树。树具有层次关系,在树型结构中结点间的关系是前驱唯一而后继不唯一,即结点之间是一对多的关系。树型结构在客观世界中是大量存在的,例如行政组织机构,家谱等都可用树形象地表示。图 2.4.1 反映了一个工厂的行政结构。在计算机领域,树型结构也有着广泛的应用,例如在编译程序中,可用树来表示源程序的语法结构;在操作系统中用树表示文件目录。从实际应用出发,本节中重点讨论二叉树的概念、性质以及二叉树的存储、遍历和应用。

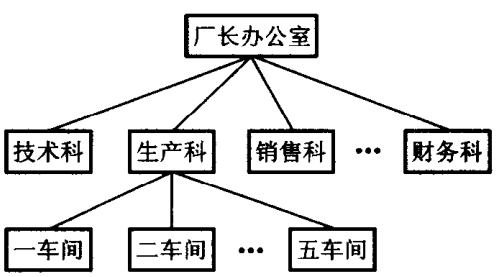


图 2.4.1　一个工厂的行政层次结构树

2.4.1 树的基本概念和术语

树(Tree)是 $n(n\geq 0)$ 个结点的有限集 T。当 $n=0$ 时,称为空树;当 $n>0$ 时,该集合满足如下条件:

- 有且仅有一个特定的称为根(root)的结点。
- 其余的结点可分为 $m(m\geq 0)$ 个互不相交的子集 $T1,T2,\cdots,Tm$,其中每个子集本身又是一棵树,并称其为根的子树(SubTree)。

树的递归定义刻画了树的固有特性,即一棵非空树是由若干棵子树构成的,而子树又可由若干棵更小的子树构成。

在树的图形表示中,通常用一个圆圈表示一个结点,结点的名字写在圆圈内或是圆圈旁边,以区别于其他的结点。在根结点与它的子树的根节点之间,加一条连线,表示它们之间的逻辑关系。

例如,在图 2.4.2 中,(a)是只有一个根结点的树,(b)是有 12 个结点的树,其中 A 是根,余下的 11 个结点分成 3 个互不相交的子集:$T1=\{B,E,F,J\}$,$T2=\{C\}$,$T3=\{D,G,H,I,K,M\}$。$T1$、$T2$、$T3$ 都是树,而且是根节点 A 的子树。对于树 $T1$,根结点是 B,其余的结点分成两个互不相交的子集:$T11=\{E\}$,$T12=\{F,J\}$。$T11$、$T12$ 也是树,而且是根结点 B 的子树。而在 $T12$ 中,F 是根,$\{J\}$ 是 F 的子树。

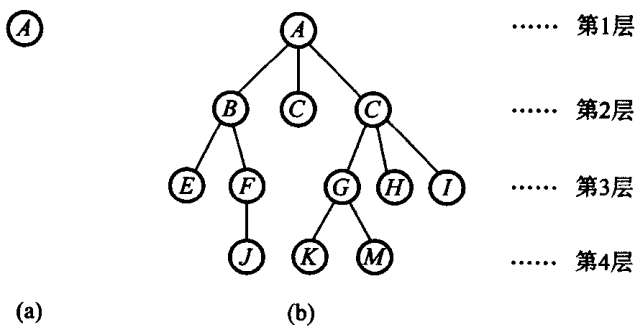


图 2.4.2 树的示例

下面介绍树结构中常用的基本术语。

(1) 结点的度

一个结点的子树数称为该结点的度(Degree)。在图 2.4.2(b)中,结点 A、D 的度为 3,结点 B、G 的度为 2,其余结点的度均为 0。

(2) 树的度

树中各结点的度的最大值。图 2.4.2(b)所示的树的度为 3,且称这棵树为 3 度树。

(3) 叶子

树中度为0的结点称为叶子(Leaf)或终端结点。图2.4.2(b)中,结点E、J、C、K、M、H、I都是叶子。

(4) 分枝结点

树中度不为0的结点称为分支结点或非终端结点。图2.4.2(b)中,结点A、B、D、F、G都是分支结点。

(5) 双亲和孩子

结点的子树的根称为该结点的孩子(Child),相应地,该结点称为孩子的双亲(Parent)。图2.4.2(b)中,A是B、C、D的双亲,而B、C、D都是A的孩子。对于B来说,它又是E、F的双亲,而E、F都是B的孩子。显然,对于一棵树来说,其根结点没有双亲,所有的叶子没有孩子。

(6) 结点的层

规定树的根结点的层(Level)为1,其余任一结点的层等于它的双亲的层加1。

(7) 树的深度

树中各结点的层的最大值称为树的深度(Depth)或高度。图2.4.2(a)中树的深度为1,图2.4.2(b)中树的深度为4。

(8) 兄弟和堂兄弟

同一双亲的孩子之间互称为兄弟(Sibling)。其双亲在同一层的结点互为堂兄弟。图2.4.2(b)中,E和F是兄弟,G,H,I是兄弟,但E、F和G、H、I是堂兄弟。

(9) 祖先和孙子

一个结点的祖先是指从树的根到该结点所经分支上的所有结点。一个结点的子树的所有结点称为该结点的子孙。图2.4.2(b)中,G的祖先是A、D,而G、H、I、K、M是D的子孙。

(10) 有序树和无序树

如果树中任一结点的各棵子树规定从左至右是有次序的,即不能互换位置,则称该树为有序树,否则称为无序树。在有序树中,根的最左边的子树称为根的第一棵子树,其余子树依次从左至右称为根的第2棵子树,第3棵子树……。对于有序树,交换任一结点的两棵子树的位置便构成不同的有序树。

(11) 森林

$n(n\geqslant 0)$棵互不相交的树的集合称为森林(Forest)。删去一棵树的根结点便得到一个森林;反过来,给一个森林加上一个结点,使原森林的各棵树成为所加结点的子树,便得到一棵树。例如,删去图2.4.2(b)中结点A便可得到一个具有3棵树的森林。

图2.4.3所示的是两棵不同的有序树,但如果将它们看作是无序树,则是同一棵无序树。

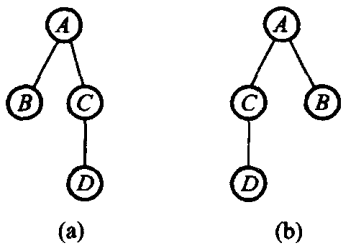


图 2.4.3 两棵不同的有序树

2.4.2 二叉树

二叉树是应用最广泛的一类树结构。它具有许多重要性质,它的存储结构比一般树的存储表示简单,同时任一棵树可以简便地转化为一棵二叉树。

1. 二叉树的定义

二叉树(Binary Tree)是 $n(n\geq 0)$ 个结点的有限集,它或者是空集($n=0$),或者是由一个根结点和两棵互不相交的且分别称为根的左子树和右子树的二叉树组成。这是二叉树的递归定义。若二叉树为空集,则称之为空二叉树,二叉树的每个结点只有两棵子树,且有左子树、右子树之分,不能互换位置,左、右子树可以是空二叉树。因此,二叉树共有 5 种基本形态,如图 2.4.4 所示。

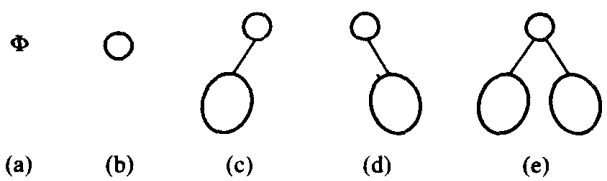


图 2.4.4 二叉树的 5 种基本形态

图 2.4.4 中,(a)为空二叉树、(b)为仅有一个根结点的二叉树、(c)为右子树为空的二叉树、(d)为左子树为空的二叉树、(e)为左、右子树均非空的二叉树。

由以上特点可以看出,在二叉树中,每一个结点的度最大为 2。二叉树和度为 2 的树是两个不同的概念。二叉树每个结点的度小于等于 2,其左子树和右子树是有序的;而度为 2 的树每个结点的度也小于等于 2,但一定至少有一个结点度为 2,可以不规定各子树之间的次序,即可以是有序树也可以是无序树。因此,图 2.4.5 中(a)和(b)是两棵完全不同的二叉树,但是如果把它们看作是一般的树结构,那么它们是相同的树。

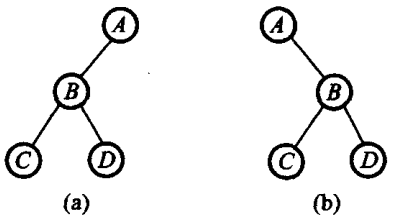


图 2.4.5　两棵不同的二叉树

2. 二叉树的性质

二叉树具有以下重要性质。

性质 1：在二叉树的第 i 层上，最多有 2^{i-1} 个结点（$i \geq 1$）。根据二叉树的特点，这个性质是显然的，利用归纳法容易证得此性质，这里从略。

性质 2：深度为 k 的二叉树最多有 $2^k - 1$ 个结点。

深度为 k 的二叉树是指二叉树共有 k 层。由性质 1，只要将第 1 层到第 k 层上最大的结点数相加，就可以得到整个二叉树中结点数的最大值，即

$$2^0 + 2^1 + \cdots + 2^{k-1} = 2^k - 1 \qquad (式\ 2.4.1)$$

性质 3：在任意一棵二叉树中，若度为 0 的结点（即叶子结点）数为 n_0，度为 2 的结点数为 n_2，则 $n_0 = n_2 + 1$。

设 n_1 为二叉树中度为 1 的结点数。因为二叉树中所有结点的度数均小于等于 2，所以结点总数为

$$n = n_0 + n_1 + n_2 \qquad (式\ 2.4.2)$$

另一方面，1 度结点有一个孩子，2 度结点有两个孩子，故二叉树中孩子结点的总数是 $n_1 + 2n_2$，树中只有根结点不是任何结点的孩子，故二叉树中的结点总数又可表示为

$$n = n_1 + 2n_2 + 1 \qquad (式\ 2.4.3)$$

由（式 2.4.1）和（式 2.4.2）得到

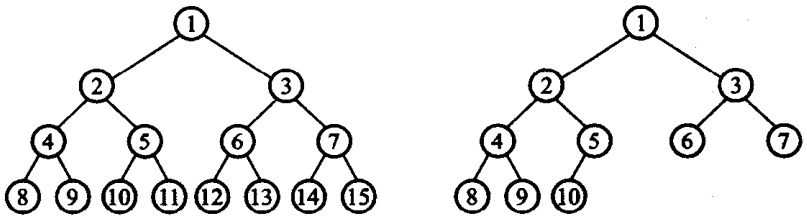
$$n_0 = n_2 + 1 \qquad (式\ 2.4.4)$$

满二叉树（Full Binary Tree）　一棵深度为 k 且具有 $2^k - 1$ 个结点的二叉树称为满二叉树。如图 2.4.6(a)所示是一棵深度为 4 的满二叉树。满二叉树中不存在度数为 1 的结点，每个分支结点均有两棵高度相同的子树，每一层上的结点数都达到最大值，且树叶都在最后一层上。

可以对满二叉树的结点进行顺序编号，约定编号从根结点起，自上而下，同层的结点从左至右。图 2.4.6(a)所示的就是用顺序编号法表示的满二叉树。

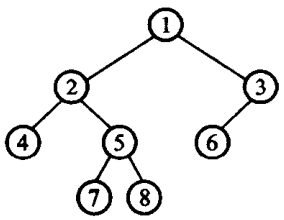
完全二叉树（Complete Binary Tree）　深度为 k，有 n 个结点的二叉树，当且仅当其每一个结点都与深度为 k 的满二叉树中编号从 1 至 n 的结点一一对应时，称之为完全二叉树。

如图 2.4.6(b)所示是一棵深度为 4 的完全二叉树。完全二叉树除最后一层外，每一层上的结点数都达到最大值，在最后一层上可能缺少右边的若干结点。显然满二叉树是完全二叉树，但完全二叉树不一定是满二叉树。图 2.4.6(c)所示的二叉树就不是一棵完全二叉树。



(a) 满二叉树

(b) 完全二叉树



(c) 非完全二叉树

图 2.4.6 特殊形态的二叉树

性质 4：具有 n 个结点的完全二叉树的深度为

$$\lfloor \log_2 n \rfloor + 1 \qquad (式\ 2.4.5)$$

设完全二叉树的深度为 k，由完全二叉树的定义，它的前 $k-1$ 层是深度为 $k-1$ 的满二叉树，由性质 2 知，一共有 $2^{k-1}-1$ 个结点，第 k 层上还有若干结点，因此该完全二叉树总的结点数 $n > 2^{k-1}-1$，同时 $n \leqslant 2^k - 1$，即

$$2^{k-1} - 1 < n \leqslant 2^k - 1 \qquad (式\ 2.4.6)$$

或

$$2^{k-1} \leqslant n < 2^k$$

取对数后有

$$k - 1 \leqslant \log_2 n < k \qquad (式\ 2.4.7)$$

因为 $k-1$ 和 k 是相邻的两个整数，故有

$$k - 1 = \lfloor \log_2 n \rfloor \qquad (式\ 2.4.8)$$

即

$$k = \lfloor \log_2 n \rfloor + 1 \qquad (式\ 2.4.9)$$

采用顺序编号的完全二叉树还具有以下性质

① 若 $i = 1$，则结点 i 是二叉树的根，无双亲；如果 $i > 1$，则该结点的父结点编号为 $\lfloor i/2 \rfloor$。

② 若 $2i \leqslant n$，则结点 i 的左孩子是结点 $2i$；若 $2i > n$，则结点 i 无左孩子。

③ 若 $2i+1 \leqslant n$，则结点 i 的右孩子是结点 $2i+1$；若 $2i+1 > n$，则结点 i 无右孩子。

根据完全二叉树的这些性质，很容易确定任一结点的双亲、左孩子和右孩子的位置。

3. 二叉树的存储结构

(1) 顺序存储结构

该方法是把二叉树的所有结点按从上至下、从左至右的顺序存放到一块地址连续的存储单元中。通常，用一维数组作存储结构。这种方法适宜表示完全二叉树。例如，一棵有 8 个结点的完全二叉树，其顺序存储结构如图 2.4.7 所示。为了清晰地反映完全二叉树中结点之间的逻辑关系，下标为 0 的空间通常不用，或是用来存放其他信息。

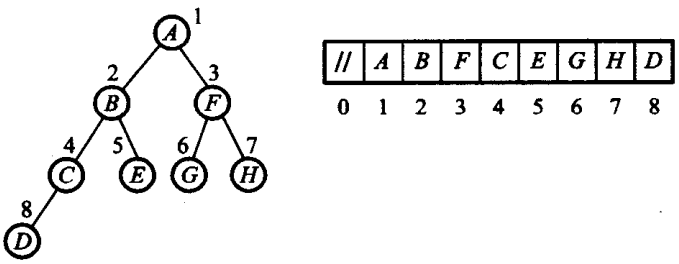


图 2.4.7　完全二叉树的顺序存储结构

对于一般的二叉树来说，为了能用结点在向量中的相对位置表示结点之间的逻辑关系，也必须按完全二叉树的形式来存储树中的结点。例如，图 2.4.8 所示是一棵非完全二叉树的顺序存储结构图。容易看出，一般的二叉树用顺序存储结构容易造成存储空间的浪费。在最坏的情况下，一个深度为 k 且只有 k 个结点的右单枝树需要 2^k-1 个结点的存储空间，而其中只有 k 个位置上存入了结点，空间利用率仅为 $k/(2^k-1)$。

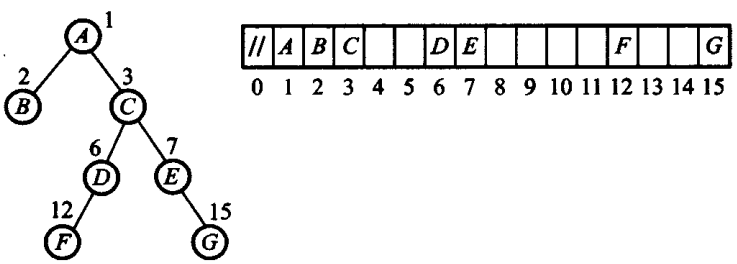


图 2.4.8　非完全二叉树的顺序存储结构

(2) 链式存储结构

通常用链式存储结构存储二叉树。由于二叉树的每个结点最多有两个孩子，因此，每个

结点除了存储结点本身的数据外，还应设置两个指针分别指向两个孩子。

结点的结构如图 2.4.9 所示。其中 *data* 域存储结点的值，*lchild* 域及 *rchild* 域分别存储指向左、右孩子的指针。若不存在左孩子或右孩子，则其相应的指针域为空指针。这样的存储结构称为二叉链表。此外，为每棵二叉树设立一个指向根结点的指针 *root*。若二叉树为空，则 *root* 为空指针。例如，图 2.4.10(a)所示二叉树的二叉链表表示如图 2.4.10(b)所示。

图 2.4.9 二叉链表结点的结构

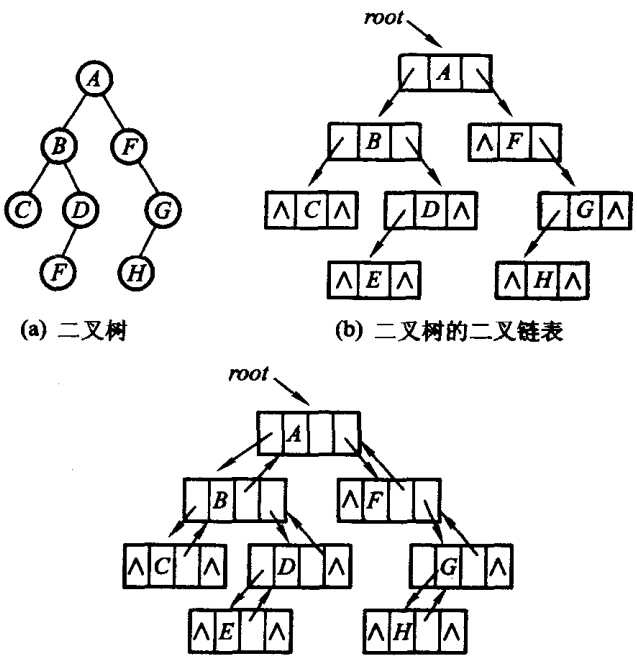


图 2.4.10 二叉树的链式存储结构

显然具有 n 个结点的二叉链表表示中，一共有 $2n$ 个指针域，其中只有 $n-1$ 个用来指示结点的左、右孩子，其余的 $n+1$ 个指针域为空。

通过结点的左、右孩子指针可以立即找到它的左、右孩子，但不能直接找到它的双亲。如果需要经常寻找二叉树中某个结点的双亲，可以在结点结构中增加一个指向双亲的指针域 *parent*。通常，将每个结点带 3 个指针域 *lchild*、*rchild*、*parent* 的链表称为二叉树的三叉链

表。结点结构如图 2.4.11 所示。

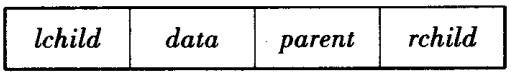


图 2.4.11 三叉链表的结点结构

例如,图 2.4.10(a)所示二叉树的三叉链表表示如图 2.4.10(c)所示。

二叉链表是二叉树最常用的存储结构,下面讨论的二叉树的遍历都是以二叉链表作为二叉树的存储结构。二叉链表中结点结构类型定义如下:

```
typedef   char   TelemType;      /* TelemType 类型由用户根据需要定义 */
typedef   struct BiTNode
{TElemtype      data;
    Struct      BiTNode      * lchild,    * rchild;
} BiTNode,    * BiTree;
```

2.4.3 遍历二叉树

在对二叉树的各种操作中,最基本的操作之一是遍历二叉树(Traversing Binary Tree)。遍历一棵二叉树,就是按照某种规则去访问二叉树的每个结点,而且每个结点仅被访问一次。

遍历一个线性结构很容易,只需从开始结点出发,依次访问当前结点的直接后继,直至终点为止。而二叉树是一种非线性结构,每个结点都可能有两个直接后继结点,这将导致存在多条遍历路线。因而需要寻找一种规则,以便不重不漏地访问到二叉树中的所有结点。从二叉树的递归定义可知,一棵非空的二叉树由根结点及左、右子树这 3 个基本部分组成。因此,在任一给定结点上,可以按某种次序执行 3 个操作:访问根结点,遍历这个根结点的左子树,遍历这个根结点的右子树。如果用 N 表示访问根结点,用 L、R 分别表示遍历这个根结点的左子树、右子树,则可以有 NLR、LNR、LRN、NRL、RNL、RLN 等 6 种不同的遍历规则。前 3 种遍历按先左子树后右子树的次序进行,后 3 种遍历按先右子树后左子树的次序进行。如果限定先左子树后右子树,则只有前 3 种情况。NLR、LNR、LRN 分别表示访问根结点的操作是发生在遍历其左右子树之前、之中和之后。根据访问根结点的次序,将这 3 种遍历分别称为先(根)序遍历、中(根)序遍历和后(根)序遍历。下面分别介绍这 3 种遍历的方法。

1. 先序遍历

先序遍历二叉树递归定义为:若二叉树为空,则遍历结束;否则,执行下列步骤,先访问根接点,然后遍历根的左子树,最后遍历根的右子树。并且,在遍历左、右子树时,仍然先访问根结点,然后遍历左子树,最后右子树。

例如,先序遍历图 2.4.10 (a)所示的二叉树,得到这棵二叉树的先序序列为:*A*、*B*、*C*、*D*、*E*、*F*、*G*、*H*。

先序遍历二叉树的递归算法如下：

```
void   PreorderTraverse( BiTree T)
                              /* 算法中的 1、2、3 是为说明执行过程加入的标号 */
{   if(T)
1     {  printf( "%c" ,T -> data) ;
2        PreorderTraverse( T -> lchild) ;
3        PreorderTraverse( T -> rchild) ;
      }
}
```

对图 2.4.12 所示的二叉树，递归算法 PreorderTraverse 遍历此二叉树的执行踪迹如图 2.4.13 所示，图中用 *POT* 表示执行算法 PreorderTraverse。从图 2.4.13 容易看出算法 PreorderTraverse 输出结点的次序为 *ABCDE*。

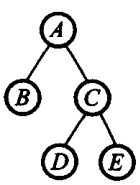


图 2.4.12 二叉树

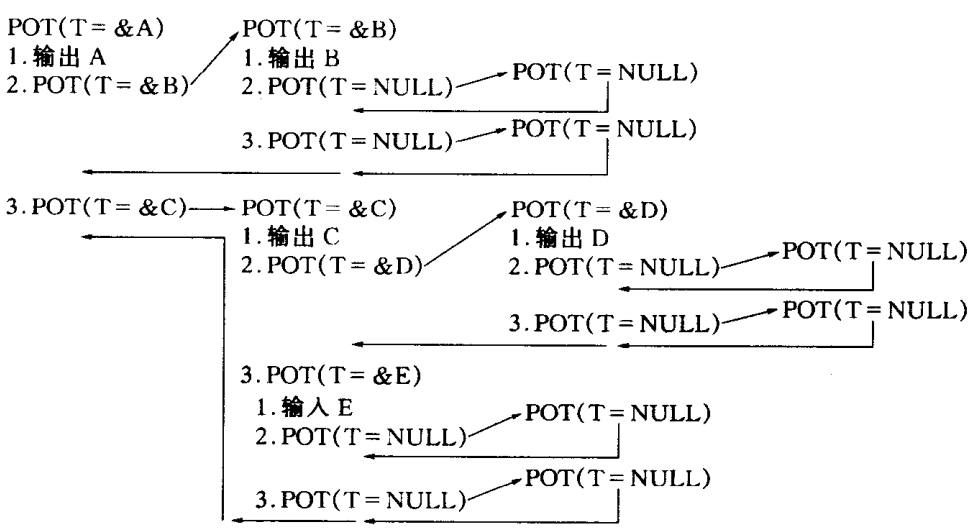


图 2.4.13 算法 POT(PreorderTraverse)执行踪迹示意图

2. 中序遍历

中序遍历二叉树递归定义为:若二叉树为空,则遍历结束;否则,执行下列步骤,先遍历根的左子树,然后访问根结点,最后遍历根的右子树。并且,在遍历左右子树时,仍然先遍历左子树,然后访问根结点,最后遍历右子树。例如,中序遍历图 2.4.10(a)所示的二叉树,得到这棵二叉树的中序序列为:*C*、*B*、*E*、*D*、*A*、*F*、*H*、*G*。

中序遍历二叉树的递归算法如下:

```
void    InorderTraversr( BiTree T)
{   if(T)
    {   InorderTraversr (T -> lchild);
        printf("%c",T -> data);
        InorderTraversr(T -> rchild);
    }
}
```

3. 后序遍历

后序遍历二叉树递归定义为:若二叉树为空,则遍历结束;否则,执行下列步骤,先遍历根的左子树,然后遍历根的右子树,最后访问根结点。并且,在遍历左、右子树时,仍然先遍历左子树,然后遍历右子树,最后访问根结点。例如,后序遍历图 2.4.10(a)所示的二叉树,得到这棵二叉树的后序序列为:*C*、*E*、*D*、*B*、*H*、*G*、*F*、*A*。

后序遍历二叉树的递归算法如下:

```
void PostorderTraversr( BiTree T)
{   if(T)
    {   PostorderTraversr(T -> lchild);
        PostorderTraversr(T -> rchild);
        Printf("%c",T -> data);
    }
}
```

上述二叉树的遍历算法,其时间复杂度均为 $O(n)$,其中,n 为树的结点数。算法的实现是对已存在的二叉树的二叉链表进行的操作,必须事先建立一棵二叉树的二叉链表。构造二叉链表的方法很多,这里介绍一个基于先序遍历的构造算法。算法的输入是二叉树的先序序列,为了指示空指针信息,在先序序列中用一个空格符表示一个空指针的位置。例如,对图 2.4.10(a)所示二叉树,按下列次序输入字符序列:ABCΦΦDEΦΦΦFΦGHΦΦΦ 可建立相应的二叉链表,其中 Φ 表示空格字符。算法如下:

```
CreateBiTree( BiTree  * T)
{   char ch;
```

```
    if ((ch = getchar()) == '')  *T = NULL;
    else
    {    *T = (BiTree) malloc(sizeof(BiTNode));   /*生成根结点*/
         (*T) -> data = ch;
         CreateBiTree(&(*T) -> lchild);           /*递归构造左子树*/
         CreateBiTree(&(*T) -> rchild);           /*递归构造右子树*/
    }
}
```

2.4.4 哈夫曼树及其应用

哈夫曼(Huffman)树,又称最优二叉树,是一类带权路径长度最短的二叉树,有着广泛的应用。在介绍哈夫曼树之前,先介绍几个基本术语。

路径与路径长度 从树中一个结点到另一个结点之间的分支构成这两个结点之间的路径,路径上的分支数目称为路径长度。

树的路径长度 从树根到树的各个结点的路径长度之和称为树的路径长度,记作 PL。

结点的权与带权路径长度 在实际应用中,常常给树中结点赋予一个具有某种实际意义的实数,该实数称为这个结点的权。结点的带权路径长度是该结点到树根之间的路径长度与结点的权的乘积。

树的带权路径长度 树中所有叶子结点的带权路径长度之和称为树的带权路径长度,记作

$$WPL = \sum_{i=1}^{n} W_i L_i \qquad (式\ 2.4.10)$$

其中,W_i 为叶子的权,L_i 为根结点到叶子之间的路径长度。

例如,图 2.4.14 中的 3 棵二叉树,都有 5 个叶子结点,分别带权 2,3,4,6,7,它们的路径长度及带权路径长度分别为:

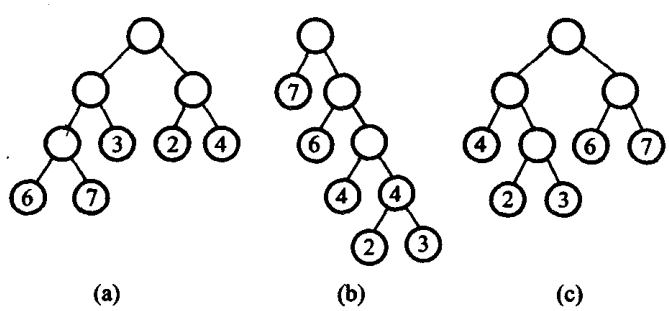


图 2.4.14 具有不同带权路径长度的二叉树

$PL(a) = 1 + 1 + 2 + 2 + 2 + 2 + 3 + 3 = 16$

$PL(b) = 1 + 1 + 2 + 2 + 3 + 3 + 4 + 4 = 20$

$PL(c) = 1 + 1 + 2 + 2 + 2 + 2 + 3 + 3 = 16$

$WPL(a) = 2 * (2 + 3 + 4) + 3 * (6 + 7) = 57$

$WPL(b) = 1 * 7 + 2 * 6 + 3 * 4 + 4 * (2 + 3) = 51$

$WPL(c) = 2 * (4 + 6 + 7) + 3 * (2 + 3) = 49$

1. 哈夫曼树

在权值为 $W_1, W_2, \cdots, W_n$ 的 n 个叶子所构成的所有二叉树中，带权路径长度最小的二叉树称为哈夫曼树。图 2.4.14(c)所示的二叉树就是一棵哈夫曼树。

如何构造权集合 $\{W_1, W_2, \cdots, W_n\}$ 的哈夫曼树，哈夫曼提出了一个算法，叙述如下：

(1) 给定的 n 个权值为 $W_1, W_2, \cdots, W_n$ 构成有 n 棵二叉树的森林 $F = \{T_1, T_2, \cdots, T_n\}$，其中每棵二叉树 T_i 只有一个权值为 W_i 的根结点，其左、右子树均为空。

(2) 森林 F 中至少还有两棵二叉树时，重复下列步骤：

① 从森林 F 中选出两棵根结点的权值最小的二叉树 T_1 和 T_2，如果这样的二叉树不止两棵，可以从中任选两棵，构造一棵新的二叉树 T，使 T_1 和 T_2 分别为 T 的左子树和右子树，T 的根的权值为 T_1，T_2 的根的权值之和，然后从森林中删去 T_1 和 T_2。

② 将新二叉树 T 插入森林 F 中。

用哈夫曼算法构造出来的二叉树一定是最优二叉树。从构造过程可以看出，权值最大的叶子离根最近，权值最小的叶子离根最远，所得二叉树必然具有最小的带权路径长度。

例如，设给定的权值为 6,7,2,3,4，构成森林 F 并排序后，如图 2.4.15(a)所示；用根为 2 和 3 的二叉树生成根为 5 的二叉树，并重新排序后，如图 2.4.15(b)所示；重复进行下去，直到 F 中剩下一棵二叉树为止，便得到哈夫曼树，如图 2.4.15(e)所示。

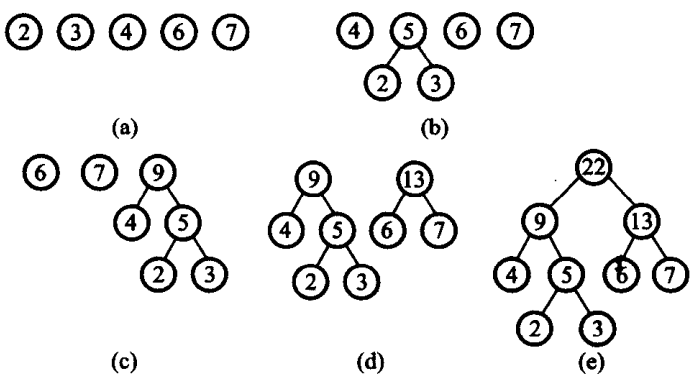


图 2.4.15　哈夫曼树的构造过程

算法的具体描述和实际问题所采用的存储结构有关，这里从略。

2. 哈夫曼编码

哈夫曼树在通信、编码和数据压缩等技术领域有着广泛的应用。下面讨论在数据编码中的一个应用，即数据的最小冗余编码问题。

在通讯业务中，可以用 0 和 1 所组成的编码来表示一个字符，一个字符序列可以用一个编码序列来表示。假定文本中出现的字符集为 26 个字母，由于 $2^4 < 26 < 2^5$，因此最简单的编码方案是每个字母都用固定长度为 5 的二进制编码来表示。这种编码不考虑字符在文本中的出现频度。

现在讨论另一种编码方案。在实际应用中，各个字符的出现频度或使用次数是不相同的。有些字符经常出现，比如 A 和 E，有些字符的使用率很低，比如 Z 和 Q。设计编码的目标是，希望用短的编码来表示那些出现频度大的字符，用长的编码来表示出现频度小的字符，从而使得要表示的字符序列（文本）的编码序列的总长度最小，所需总空间量最少。这就是最小冗余编码问题。

假定要用到的字符集为 $\{A,E,R,T,F,D\}$，各字符在文本中的出现次数为 $\{8,4,5,3,1,1\}$，现在为这些字符设计最优编码。

首先用 $\{8,4,5,3,1,1\}$ 作权构造一棵哈夫曼树，如图 2.4.16 所示。叶子中的权为字符的出现频度。然后在各分支结点发出的左分支上标上 0，右分支上标上 1。于是，从根结点到叶子的路径上的 0 和 1 所组成的序列就是该叶子所对应的字母编码，称这样的树为哈夫曼编码树，由此所得的编码称为哈夫曼编码。本例中各字母的哈夫曼编码如表 2.4.1 所示。

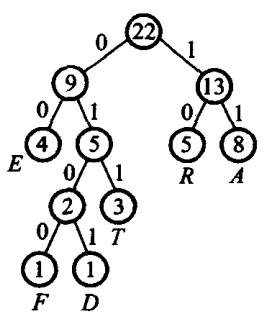


图 2.4.16 哈夫曼编码树

表 2.4.1 哈夫曼编码表

字母	A	E	R	T	F	D
编码	11	00	10	011	0100	0101

其中，出现次数较多的字母 A、E、R，具有最短的编码，长度均为 2；出现次数较少的字母 F、D 具有最长的编码，长度均为 4。

哈夫曼编码具有下述两个优点：

- 对给出的文本具有最短的编码序列。
- 任一字符的编码不会是另一字符编码的前缀（词头），即使两个字符的出视频度相同，其编码也不相同。

例如，上面 A 的编码 11 与其他字符编码的前 2 位都是不相同的。这样，两个字符间不需要加分隔符，但两个单词之间仍需要加空白。

一个任一长度的编码序列可被唯一地翻译为一个字符序列（单词）。依次取出编码序列中的 0 或 1，从哈夫曼编码树的根开始去寻找一条路径，若为 0，则沿着左分支往下走，若为 1，则沿着右分支往下走。每到达一个叶子时，就译出一个相应的字母。然后再回到哈夫曼编码树的根结点，依次译出余下的字母，最后得到一个单词。

例如，对于编码序列 1110011，对照图 2.4.16 中的哈夫曼编码树，依次取出 1，1，从根 22 走向右孩子 13，再走向右孩子 8，这已是一个叶子，故译出字母 A；然后由 10 译出 R；最后由 011 译出 T。这样，1110011 的译码为单词 ART。又如，编码序列 1101000110010 的译码为单词 $AFTER$。

由于哈夫曼码使得编码序列的总长度最短，因此对于计算机来说，所需的总存储量最少。

哈夫曼树还有其他应用，这里不再列举。

2.5 查 找

查找是指在一个给定的数据结构中查找某个指定的元素。通常，根据不同的数据结构，采用不同的查找方法。查找是数据处理领域中的一个重要内容，查找的效率将直接影响到数据处理的效率。

在查找算法中，常使用关键字这个术语。关键字（Key）是指数据元素（或记录）中某个数据项的值，用它可以标识一个数据元素（或记录）。

关键字的选定依赖于具体的应用。例如，学生登记表由 n 个记录组成，每个记录有学号、姓名、班级、成绩等数据项。如果指定一个学号，查找对应学生的姓名等信息，学号就是学生记录的关键字。

假定由 n 个记录组成一个表 $F=(R_1,R_2,\cdots,R_n)$，记录 R_i 的关键字是 $Keyi(i=1,2,\cdots,n)$。现在给定一个关键字 K，在 F 中确定一个记录 R_i，使得它的关键字 $Keyi=K$，这个过程称为查找（Searching）。

如果确定了关键字为 K 的记录 R_i，则称这次查找是成功的。这时可以对该记录进行访问、修改、删除等方面的操作。如果没有找到关键字为 K 的记录，则称这次查找失败。如果需要，可以把关键字为 K 的记录插入到被查的表中。

查找可以分为静态查找和动态查找。静态查找仅访问某个记录，查找结束后不改变表的长度，也不修改记录的内容。而动态查找可能要插入一个新记录到表中，或者是修改、删除一个原有的记录。

由于查找运算的主要操作是关键字的比较，所以通常把查找过程中对关键字需要执行的平均比较次数（也称为平均查找长度）作为衡量一个查找算法效率优劣的标准。平均查找长度 ASL（Average Search Length）定义为

$$ASL = \sum_{i=1}^{n} p_i c_i \qquad (式\ 2.5.1)$$

其中，n 是记录的个数，p_i 是查找第 i 个记录的概率，以后若不特别声明，均认为每个记录的查找概率相等，即 $p_i = 1/n$，c_i 是找到第 i 个记录所需进行的比较次数。

2.5.1 顺序查找

顺序查找（Sequential Search）是一种最简单的查找方法。它的基本思想是：从表的一端开始，逐个进行记录的关键字和给定值的比较，若某个记录的关键字和给定值相等，则查找成功；若找遍所有的记录，仍未找到某个记录的关键字和给定值相等，则查找失败。

顺序查找方法既适用于线性表的顺序存储结构，也适用于线性表的链式存储结构。在单链表中顺序查找的算法已在 2.2.4 节中讨论过，下面只讨论线性表采用顺序存储结构时，查找算法的实现过程。

假设 n 个记录已存入一维数组 R 中的 $R[1..n]$ 的区域中，$R[0]$ 作监视哨，存放要查找的给定值 K。算法从最后一个记录开始，依次与 K 进行比较，若直到第一个记录，其关键字和给定值都不等，则查找失败，返回 0；否则查找成功，返回记录的位置。

类型说明及算法如下：

```
#define N 100
typedef int KeyType;                    /* 假设关键字为整型 */
typedef struct
{   KeyType key;
    InfoType otherinfo;                 /* otherinfo 类型依赖于具体应用 */
} Element;
typedef Element SqList[N+1];            /* 0 号单元作监视哨 */
```

```
int Seq_Search(SqList R, KeyType K)
{   int i;
    R[0].key = K;                          /* 设置监视哨 */
    for(i = N;  R[i].key! = K; i - - );     /* 从表尾往前找 */
        return i;        /* 若 i 为 0,查找失败,否则返回记录所在的位置 i */
}
```

算法中设立监视哨是为了避免在查找的过程中每一步都要检测下标是否越界的条件$i \geq 1$,节省比较的时间。即使查找失败,由于监视哨中的值为给定值 K,循环也能终止。

对 n 个记录进行顺序查找,若查找成功,则所需比较关键字的次数最少为 1,即 $R[n]$就是要找的记录;最多为 n,即 $R[1]$是要找的记录; 若要查找记录 $R[i]$,则需比较 $n-i+1$ 次。假定每个记录的查找概率相同,查找成功时的平均查找长度为

$$ASL = \sum_{i=1}^{n} p_i c_i = \frac{1}{n}\sum_{i=1}^{n}(n-i+1) = (n+1)/2 \qquad (式\ 2.5.2)$$

即查找成功时,平均比较次数约为表长的一半。若查找失败,则需进行 $n+1$ 次比较。

顺序查找法的优点是算法简单,对查找表无任何特殊要求,其缺点是查找效率低。当 n 较大时不宜采用顺序查找。

在实际问题中,表中各记录的查找概率常常不等,可以把查找概率大的记录尽量放在表的尾部,以提高顺序查找的效率。

2.5.2　折半查找

折半查找(Binary Search)又称为二分查找。折半查找要求查找表采用顺序存储结构,且表中记录按关键字大小次序排列。假定用 $R[low..high]$存放 n 个递增有序记录,折半查找的基本思想是将线性表中间位置

$$mid = \lfloor (low + high)/2 \rfloor \qquad (式\ 2.5.3)$$

记录的关键字与给定值 K 进行比较,有下面 3 种情况:

(1) 若 $K = R[mid].key$,查找成功,$R[mid]$就是所要找的记录。

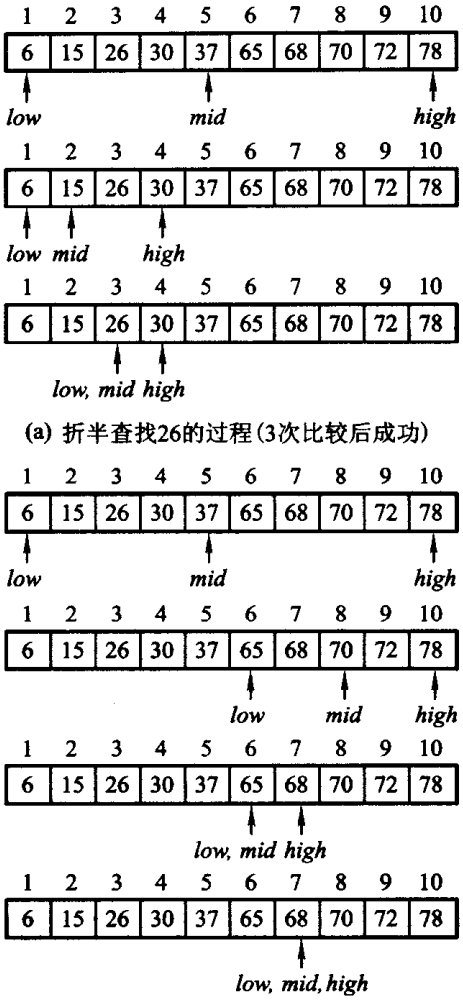
(2) 若 $K < R[mid].key$,当存在关键字值为 K 的记录时,那么待查的记录必定在表的前半部分 $R[low..mid-1]$,此时再对前面的子表进行折半查找。

(3) 若 $K > R[mid].key$,当存在关键字值为 K 的记录时,那么待查的记录必定在表的后半部分 $R[mid+1..high]$,此时再对后面的子表进行折半查找。

这个过程重复进行,直到查找成功或者子表长度为 0。

例如,已知有 10 个数据元素的有序关键字序列为(6,15,26,30,37,65,68,70,72,78),

要折半查找关键字为 26 和 67 的数据元素，查找过程分别如图 2.5.1(a)、(b)所示。



(a) 折半查找26的过程(3次比较后成功)

(b) 折半查找67的过程(4次比较后失败)

图 2.5.1　折半查找过程示意图

由此可知，每进行一次查找，或者查找成功，或者查找范围缩小一半，不像顺序查找，需要对表中记录逐一进行比较。折半查找的效率比顺序查找高。

折半查找算法如下：

```
int Bin_Search(SqList R, KeyType K)
{   /* 在有序表R[1..n]中进行折半查找，成功时返回记录的位置，失败时返回0 */
    int low = 1, high = n, mid;                    /* 置区间初值 */
    while (low <= high)                            /* 当前查找区间非空 */
```

```
    {   mid = (low + high)/2;
        if(R[mid].key = = K) return mid;
                                        /* 查找成功,返回记录所在的位置 mid */
        if(R[mid].key > K) high = mid - 1;        /* 继续在前半部分查找 */
        else low = mid + 1;                       /* 继续在后半部分查找 */
    }
    return 0;                                     /* 查找失败,返回 0 */
}
```

折半查找过程可用一个称为判定树的二叉树描述,判定树中每一结点对应表中一个记录,但结点值不是记录的关键字,而是记录在表中的位置序号。根结点对应当前区间的中间记录,左子树对应前一子表,右子树对应后一子表。例如,对于上述 10 个元素的有序表,折半查找过程的判定树如图 2.5.2 所示。

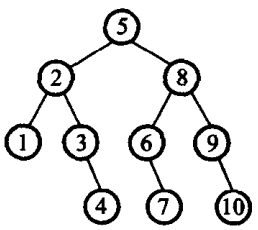


图 2.5.2　折半查找过程的判定树($n=10$)

显然,若要找的结点是表中第 5 个结点,则只需进行 1 次比较;若要找的结点是表中第 2 个或第 8 个结点,则需进行 2 次比较;若要找的结点是表中第 1、3、6、9 个结点,则需进行 3 次比较;若要找的结点是表中第 4、7、10 个结点,则需进行 4 次比较。由此可见,找到有序表中任一记录的过程,对应判定树中从根结点到与该记录相应的结点的路径,而所做比较的次数恰为该结点在判定树上的层次数。因此,折半查找成功时,关键字比较次数最多不超过判定树的深度。由于判定树的叶子结点所在层次之差最多为 1,故 n 个结点的判定树的深度与 n 个结点的完全二叉树的深度相等,均为$\lfloor \log_2 n \rfloor + 1$。这样,折半查找成功时,关键字比较次数最多不超过$\lfloor \log_2 n \rfloor + 1$。如果在图 2.5.2 所示判定树中所有结点的空指针上加一个指向一个方形结点的指针,通常称这些方形结点为外部结点,如图 2.5.3 所示。那么,折半查找失败的过程对应判定树中从根结点到某个外部结点的路径。因此,折半查找失败时,关键字比较次数最多也不超过判定树的深度$\lfloor \log_2 n \rfloor + 1$。

假设每个记录的查找概率相等,可以证明,折半查找成功时的平均查找长度为

$$ASL = \frac{n+1}{n}\log_2(n+1) - 1$$

折半查找方法的优点是比较次数少,查找速度快,平均性能好;其缺点是要求待查表为

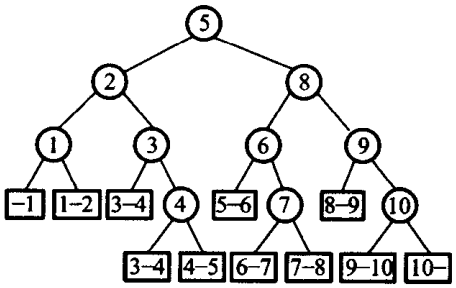


图 2.5.3　加上外部结点的判定树($n=10$)

顺序有序表,且插入删除困难。因此,折半查找方法适用于不经常变动而查找频繁的有序表。

2.5.3　分块查找

分块查找(Blocking Search)又称索引顺序查找。它是一种性能介于顺序查找和折半查找之间的查找方法。它要求将查找表组织成以下索引顺序结构。

首先将查找表分成若干个块(子表)。一般情况下,块的长度相等,最后一块可以不等。每块中元素任意排列,即块内无序,但前一块中的最大关键字必须小于后一块中的最小关键字,即块与块之间有序。

然后构造一个索引表。其中每个索引项对应一个块,并记录每块的起始位置,以及每块中的最大关键字。索引表按关键字有序排列。

图 2.5.4 所示为一个索引顺序表,其中包括 3 个块。第 1 个块的起始地址为 1,块内最大关键字为 30;第 2 个块的起始地址为 5,块内最大关键字为 45;第 3 个块的起始地址为 9,块内最大关键字为 60。

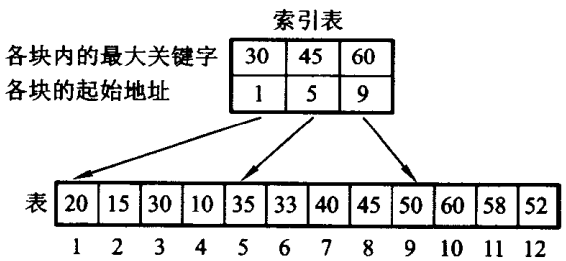


图 2.5.4　分块查找示意图

分块查找的基本思想是:首先查找索引表,由于索引表是有序表,故可采用二分查找或顺序查找,以确定待查的记录在哪一块;然后在已确定的块中进行顺序查找。例如,假设查找给定值 $K=33$ 的记录,首先查找索引表,由于 $30<33<45$,所以关键字为 33 的记录如果存在的话,则必定在第二块中;而第二块的起始地址是 5,结束地址是 8,故从查找表中第 5

个记录起顺序查找，本例中，第 6 个记录的关键字是 33，查找成功。如果在本块中不存在所要查找的记录，则查找失败。

由于分块查找实际上是两次查找过程，故整个查找过程的平均查找长度，是两次查找的平均查找长度之和。若以 L_B 表示查找索引表时的平均查找长度，L_W 表示在相应块内顺序查找记录时的平均查找长度，则有：$ASL_{bs} = L_B + L_W$

假设将长度为 n 的表分成 b 块，每块含有 s 个记录，即 $b = \lfloor n/s \rfloor$；又假设表中每个记录的查找概率相等，则每个索引项的查找概率为 $1/b$，块中每个记录的查找概率为 $1/s$。

若以顺序查找来确定块，则查找成功时的平均查找长度

$$ASL_{bs} = L_B + L_W = \frac{b+1}{2} + \frac{s+1}{2} = \frac{b+s}{2} + 1 = \frac{1}{2}\left(\frac{n}{s} + s\right) + 1$$

此时的平均查找长度与 n、s 有关，在 n 给定的前提下，容易证明，当 $s = \sqrt{n}$ 时，ASL_{bs} 取最小值 $\sqrt{n} + 1$。

若以折半查找来确定块，则分块查找成功时的平均查找长度

$$ASL_{bs} = L_B + L_W \approx \log_2(b+1) + \frac{s+1}{2} \approx \log_2\left(\frac{n}{s} + 1\right) + \frac{s}{2}$$

由此可见，分块查找算法的效率介于顺序查找和二分查找之间。

2.5.4　二叉排序树查找

前面介绍的查找方法适用于静态查找表，若要对动态查找表进行高效率的查找，可采用本节介绍的二叉排序树作为查找表的组织形式。

1. 二叉排序树

二叉排序树（Binary Sort Tree）或者是一棵空树，或者是满足下列性质的二叉树：

① 若它的左子树非空，则左子树上所有结点的值均小于根结点的值。

② 若它的右子树非空，则右子树上所有结点的值均大于根结点的值。

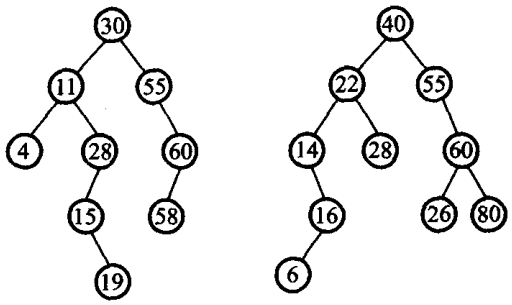
③ 左、右子树也分别为二叉排序树。

例如，图 2.5.5（a）所示的二叉树是一棵二叉排序树。对一棵二叉排序树进行中序遍历，所得的结点序列一定是递增有序的。如对图 2.5.5（a）所示二叉树进行中序遍历，可得递增有序序列：4，11，15，19，28，30，55，58，60。图 2.5.5（b）所示的二叉树不是一棵二叉排序树，它的中序遍历序列为：14，6，16，22，28，40，55，26，60，80。

如此定义的二叉排序树，各结点关键字是唯一的，实际应用中，数据元素的关键字可能相同，这时可将性质①中的“小于”改为“小于等于”，或将性质②中的“大于”改为“大于等于”。

2. 二叉排序树的生成

给定 $n(n \geq 1)$ 个元素的序列，可按下列步骤生成一棵二叉排序树：



(a) 二叉排序树　(b) 非二叉排序树

图 2.5.5　二叉排序树与非二叉排序树

① 若二叉排序树为空,则该元素为根结点。

② 若元素值小于根结点值,则将元素插入到左子树中。

③ 若元素值大于根结点值,则将元素插入到右子树中。

无论是插入到左子树还是插入到右子树,同样按照上述步骤进行。

例如,依次读入元素序列(30,11,55,4,28,15,19,60,58)生成一棵二叉排序树时,其过程如图 2.5.6 所示。

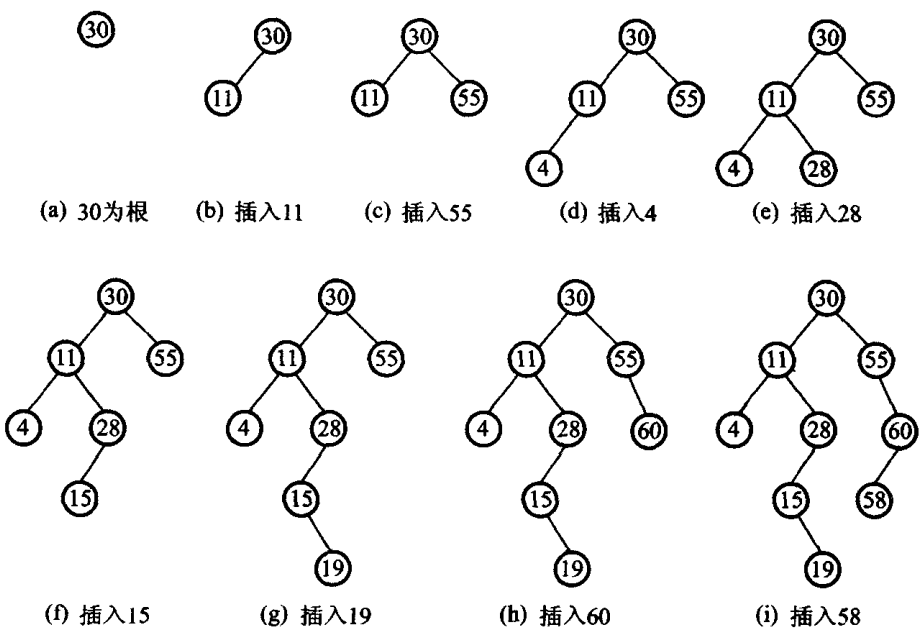


图 2.5.6　二叉排序树生成过程示例

二叉排序树的生成是一个连续插入新结点的过程。新结点总是作为叶结点插入到二叉

排序树中。假设二叉排序树用二叉链表作存储结构，插入新结点到二叉排序树中的算法如下：

```
void InsertBST(BiTree *T, ElemType e)
{  /* 若二叉排序树 *T 中没有关键字为 e 的结点，则插入，否则返回 */
   BiTree f,p = *T;                    /* p 的初值指向根结点 */
   while(p)                            /* 查找插入位置 */
   {  if(p->data = =e) return;         /* 树中已有结点值为 e，无须插入 */
      f = p; /* f 保存当前被查找的结点，是下次要查找结点的父结点 */
      if(e < p->data) p = p->lchild;   /* 在左子树中查找 */
      else p = p->rchild;              /* 在右子树中查找 */
   }
   p = (BiTree)malloc(sizeof(struct BiTNode));      /* 生成新结点 */
   p->data = e;
   p->lchild = p->rchild = NULL;
   if(! *T) *T = p;                    /* 原树为空，新结点为二叉排序树的根
                                          结点 */
   else if(e < f->data) f->lchild = p;
                                       /* 原树非空，新结点为 f 的左孩子或右
                                          孩子 */
      else f->rchild = p;
}
```

二叉排序树的生成是从空的二叉排序树开始，每输入一个元素数据，就调用一次插入算法。生成二叉排序树的算法如下：

```
BiTree CreateBST()
{  int i,e;
   BiTree T = NULL;          /* 初始时，T 为空树 */
   scanf("%d",&e);
   for(i=0;i<N;i++)          /* 插入 N 个结点 */
   { InsertBST(&T,e);
      scanf("%d",&e);
   }
   return T;                 /* 返回二叉排序树的树根 */
}
```

3. 二叉排序树的查找

由二叉排序树的生成可知,构造二叉排序树的过程实际上将无序序列变成了有序序列。在二叉排序树中查找关键字为 K 的结点,与折半查找类似,也是一个逐步缩小查找范围的过程。首先将给定值 K 与根结点的关键字进行比较,若相等,则查找成功,否则根据给定值 K 和根结点的关键字之间的大小关系,分别在左子树或右子树中继续进行查找。例如,在图 2.5.6(i)所示的二叉排序树中查找关键字 $K=28$ 的记录,因为 $K<30$,则查找以 30 为根的左子树,此时左子树非空,且 $K>11$,则查找以 11 为根的右子树,这时 K 和右子树根的关键字相等,查找成功。又如在图 2.5.6(i)中查找关键字 $K=52$ 的记录,和上述过程类似,在与关键字 30、55 比较之后,查找以 55 为根的左子树,此时左子树为空,说明该树中不存在关键字 $K=52$ 的记录,查找失败。算法描述如下:

```
BiTree SearchBST(BiTree T,ElemType k)
{  /* 在根指针 T 所指的二叉排序树中递归地查找某关键字等于 K 的数据元素,若查找成功,则返回指向该数据元素结点的指针,否则返回空指针 */
    if(T==NULL||k==T->data) return T;                    /* 查找结束 */
    if(k<T->data) return SearchBST(T->lchild,k);    /* 在左子树中继续查找 */
    else return SearchBST(T->rchild,k);                 /* 在右子树中继续查找 */
}
```

由上可知,在二叉排序树中查找关键字为 K 的结点,如果查找成功,则是走了一条从根结点到该结点的路径的过程;如果查找不成功,则是从根结点出发走了一条从根到某个叶子结点的路径。与折半查找类似,与给定值 K 比较的关键字的个数不超过树的深度。然而,折半查找长度为 n 的有序表,其判定树是唯一的,而含有 n 个结点的二叉排序树却不唯一。对于同样一组结点,由于结点插入的先后次序不同,所生成的二叉排序树的形态和深度可能不同。例如,对于与图 2.5.6 相同的一组结点,若读入元素序列为(28,15,55,19,11,30,4,60,58),则生成的二叉排序树如图 2.5.7(a)所示,如果读入元素序列为:(4,11,15,19,28,30,55,58,60),则生成的二叉排序树如图 2.5.7(b)所示。假设每个元素的查找概率相同,查找成功时,它们的平均查找长度分别是:

- 对于图 2.5.6(i)所示的二叉排序树:$ASL=(1+2+2+3+3+3+4+4+5)/9=27/9=3$。
- 对于图 2.5.6(a)所示的二叉排序树:$ASL=(1+2+2+3+3+3+3+4+4)/9=25/9\approx 2.78$。
- 对于图 2.5.6(b)所示的二叉排序树:$ASL=(1+2+3+4+5+6+7+8+9)/9=45/9=5$。

显然,在二叉排序树上进行查找,平均查找长度和二叉排序树的形态有关。最好的情况是二叉排序树的形态比较均匀,与折半查找的判定树相似,树中任一结点左、右子树深度之差的绝对值不超过 1,此时,它的平均查找长度大约是 $\log_2 n$。最坏的情况是二叉排序树蜕化为一棵深度为 n 的单支树,平均查找长度是 $(n+1)/2$。如果考虑把 n 个结点按各种可能的次序插入到二叉排序树中,则有 $n!$ 棵二叉排序树(其中有的形态相同),对这些二叉排序

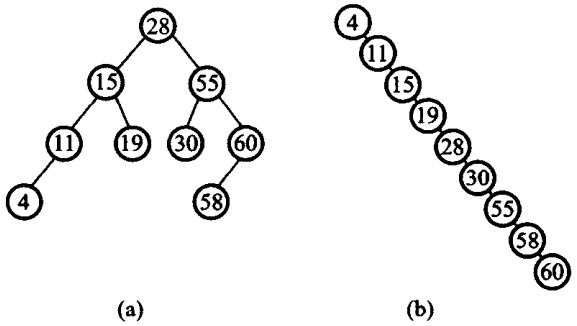


图 2.5.7 由同一组关键字构成的形态不同的二叉排序树

树的查找长度进行平均,可以证明,得到的平均查找长度仍然是 $O(\log_2 n)$。

就平均性能而言,二叉排序树上的查找和折半查找相差不大;就维护表的有序性而言,二叉排序树更为有效,因为在二叉排序树上插入结点十分方便,无需移动大量结点。因此,对于需要经常做插入、删除、查找运算的表,宜采用二叉排序树结构。在实际应用中,为了提高查找效率,应尽量使二叉排序树保持形态均匀。

2.6 排 序

2.6.1 排序的基本概念

排序(Sorting)又称分类,是指将一个无序序列整理成按关键字递增(或递减)的有序序列的处理过程。

假设有 n 个记录的序列$\{R_1, R_2, \cdots, R_n\}$,其相应关键字的序列是$\{K_1, K_2, \cdots, K_n\}$,则排序是确定一个排列 $p_1, p_2, \cdots, p_n$,使得 $K_{p1} \leqslant K_{p2} \leqslant \cdots \leqslant K_{pn}$,从而得到一个按关键字有序的序列$\{R_{p1}, R_{p2}, \cdots, R_{pn}\}$。

当待排序记录中的关键字 $K_i(i = 1, 2, \cdots, n)$ 都不相同时,则任何一个记录的无序序列经排序后得到的结果是唯一的;若待排序的序列中存在两个或两个以上具有相同关键字的记录,则排序所得到的序列不唯一。假设 $K_i = K_j(1 \leqslant i \leqslant n, 1 \leqslant j \leqslant n, i \neq j)$,且在排序前的序列中 R_i 领先于 R_j(即 $i < j$),若在排序后的序列中 R_i 仍领先于 R_j,则称所用的排序方法是稳定的;反之,称所用的排序方法是不稳定的。例如,给定无序序列(10,25,4,22,45,25*,30,18),序列中存在两个 25,其中带"*"的 25 处在不带"*"的 25 位置之后,经过排序,若得到的有序序列为(4,10,18,22,25,25*,30,45),则称此排序方法是稳定的;若得到的有序序列为(4,10,18,22,25*,25,30,45),则称此排序方法是不稳定的。

根据排序过程中所涉及的存储器不同，可将排序方法分为两类：内部排序和外部排序。若整个排序过程完全在内存中进行，称为内部排序；若待排序记录数量很大，以致不能全部放在内存中，尚需对外存进行访问的排序过程称为外部排序。排序的方法很多，本节仅讨论一些常用的内部排序方法。

评价一个排序算法优劣的标准有：对 n 个记录排序所需比较关键字的次数；对 n 个记录排序所需移动记录的次数；排序过程中所需的辅助存储空间的大小。

为了讨论方便，约定待排记录以顺序表作为存储结构，记录从下标为 1 的位置开始存放，下标为 0 的位置作为监视哨或中间变量，记录的关键字均为整数，并假定均是按关键字递增的次序排序。数据类型定义如下：

```
#define  n  100
typedef  int  KeyType;                    /* 设关键字类型为整型 */
typedef  struct
{   KeyType key;
    InfoType otherinfo;                   /* 类型 InfoType 依赖于具体应用 */
}RedType;
typedef RedType Sqlist[n+1];              /* 0 号单元作监视哨 */
```

2.6.2 冒泡排序

冒泡排序(Bubble Sort) 的基本思想是：通过对无序序列中相邻记录关键字的比较和位置的交换，使得关键字较小的记录向一头漂移，关键字较大的记录向另一头下沉。从而使无序序列变成有序序列。

设有 n 个待排序的记录，第一遍排序，将第一个记录的关键字和第二个记录的关键字进行比较，若为逆序，则将两个记录交换，否则，不进行交换，然后比较第二个记录与第三个记录的关键字。依此类推，直至第 $n-1$ 个记录和第 n 个记录的关键字进行过比较为止。经过这样一遍处理之后，其中关键字最大的记录移到了第 n 个位置上。然后，对前面的 $n-1$ 个记录进行第二遍排序，第二遍排序之后，前 $n-1$ 个记录中关键字最大的记录移到了第 $n-1$ 个位置上。继续进行下去，直到不需要再交换记录为止。

例如，对(25,89,4,25*,15,38)进行冒泡排序，其过程如图 2.6.1 所示。

初始关键字：	25	89	4	25*	15	38
第 1 遍排序后：	25	4	25*	15	38	89
第 2 遍排序后：	4	25	15	25*	38	
第 3 遍排序后：	4	15	25	25*		
第 4 遍排序后：	4	15	25			

图 2.6.1　冒泡排序示例

此例共计进行4遍排序。第1遍排序进行5次比较和4次交换。因25<89,不交换;因89>4,交换89和4的位置;因89>25*,交换89和25*的位置;因89>15,交换89和15的位置;因89>38,交换89和38的位置。

第2遍进行4次比较和2次交换。第3遍进行3次比较和1次交换。第4遍仅进行比较,没有交换,说明已得到有序文件,排序结束。

冒泡排序算法如下:

```
void BubbleSort(Sqlist R)
{   int i=1,j,swap;                          /* swap为交换标志 */
    do
    {   swap=0;                              /* 每遍排序前swap初值为0 */
        for(j=1;j<=n-i;j++)
              if(R[j].key>R[j+1].key)        /* 逆序,则交换记录 */
              {   R[0]=R[j];
                  R[j]=R[j+1];
                  R[j+1]=R[0];
                  swap=1;                    /* 交换过记录,交换标志swap置为1 */
              }
        i++;
    }while(i<n&&swap);   /* 若排序共进行了n-1遍,排序结束;若在某遍
}                                排序中,未交换过记录,排序也应结束 */
```

在算法中,使用了交换标志*swap*。每遍排序之前,置*swap*为0,在排序过程中,若进行了交换,则将*swap*改为1。该遍排序结束之后,判断*swap*,若*swap*==1,则表示刚进行过交换,需要继续进行排序;若*swap*==0,则排序结束。当然,若共计进行了$n-1$遍排序,则排序也应结束。

显然,在最好情况时,记录的原始排列为正序,只需要进行一遍排序,共计比较$n-1$次,不交换记录。在最坏情况下,记录的原始排列为反序,要经过$n-1$遍排序,第1遍比较$n-1$次,第2遍比较$n-2$次,…,第$n-1$遍比较1次。所以,算法所需总的比较次数最多为

$$max=(n-1)+(n-2)+\cdots+1=n(n-1)/2$$

若每比较一次关键字就要交换一对记录,则移动记录的次数最多为$3n(n-1)/2$。

容易看出,冒泡排序是稳定的。冒泡排序的时间复杂度为$O(n^2)$。

2.6.3 插入排序

插入排序(Insertion Sort)的基本思想是:每次将一个待排序的记录按其关键字的大小插入到已排好序的有序表中,使得插入之后得到的表仍然是有序表,如此反复,直到全部记录插入完成为止。

插入一个记录,首先要对有序表进行查找,以确定这个记录的插入位置,按查找方式的不同,插入排序又可以分为直接插入排序和折半插入排序,前者使用顺序查找,后者使用折半查找。

1. 直接插入排序

直接插入排序的基本思想是:首先将记录 R_1 看作有序子表。第 1 遍,将 R_2 插入这个有序子表中,若 R_2 的关键字小于 R_1 的关键字,则 R_2 插在 R_1 之前,否则 R_2 插在 R_1 的后面。第 2 遍,将记录 R_3 插入前面已有 2 个记录的有序子表中,得到 3 个记录的有序子表。依此类推,依次插入 $R_4,\cdots,R_n$。最后得到 n 个记录的有序表。例如,对(25,4,89,15,25*,38)的排序,其过程如图 2.6.2 所示。

	R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]
初始关键字:		25	4	89	15	25*	38
第 1 遍排序后:	4	(4	25)	89	15	25*	38
第 2 遍排序后:		(4	25	89)	15	25*	38
第 3 遍排序后:	15	(4	15	25	89)	25*	38
第 4 遍排序后:	25*	(4	15	25	25*	89)	38
第 5 遍排序后:	38	(4	15	25	25*	38	89)

图 2.6.2　线性插入排序示例

设($R_1,R_2,\cdots,R_{i-1}$)是已排序的有序子表,现在插入 R_i,若 R_i 的关键字 K_i 大于等于记录 R_{i-1}的关键字 K_{i-1},则位置 i 就是 R_i 的插入位置;若 R_i 的关键字小于记录 R_{i-1}的关键字,算法中首先将 R_i 存放到 R_0 中,然后将 $K_0(K_i)$从右向左依次与前面记录的关键字 K_{i-1},$K_{i-2},\cdots,K_1$ 进行比较。若 $K_0<K_j(j=i-1,i-2,\cdots,1)$,则将 R_j 后移一个位置,若 $K_0\geqslant K_j(j=i-1,i-2,\cdots,1)$,则查找过程结束,$j+1$ 即为 R_i 的插入位置。

直接插入排序算法如下:

```
void  InsertSort(Sqlist R)
{   int i,j;
```

```
    for(i=2; i<=n; i++)          /* 依次插入 R2,…,Rn */
    if(R[i].key<R[i-1].key)      /* 若记录 Ri 的关键字大于等于当前有序子表
中所有记录的关键字,位置 i 就是 Ri 的插入位置 */
    {   R[0]=R[i];               /* Ri 放入监视哨 */
        j=i-1;                   /* 从右向左查找 Ri 的插入位置 */
        while(R[0].key<R[j].key)
        {   R[j+1]=R[j];         /* 将关键字大于 Ri 的记录后移 */
            j--;
        }
        R[j+1]=R[0];             /* 插入记录 Ri */
    }
}
```

算法中 $R[0]$ 既作为中间变量存放 $R[i]$，又起“监视哨”的作用，防止下标 j 出界，一旦 $j=0$，循环结束，避免了循环中每次都要判断 $j\geq 1$。

直接插入排序所需比较关键字的次数与记录的原始排列有关。最好的情况是待排记录的原始序列是递增的（正序），这时所需比较关键字的次数达最小值 $n-1$，记录不需移动；反之，最坏的情况是待排记录的原始序列是递减的（反序），这时所需比较关键字的次数达最大值 $\sum_{i=2}^{n} i=(n+2)(n-1)/2$，记录移动的次数也达最大值 $\sum_{i=2}^{n}(i+1)=(n+4)(n-1)/2$。若待排序记录是随机排列的，可取上述最小值和最大值的平均值，比较和移动记录的次数约为 $n^2/4$。直接插入排序的时间复杂度为 $O(n^2)$。算法需要一个监视哨的辅助存储空间，直接插入排序是稳定的。

2. 折半插入排序

在插入排序过程中，前面插入的记录已构成一个有序子序列，因此在插入 R_i 时，可以对这个有序子序列采用折半查找来确定 $R_i(i=2,3,\cdots,n)$ 的插入位置，以减少比较次数。这种插入排序称为折半插入排序（Binary Insertion Sort）。

折半插入排序算法如下：

```
void BinsertSort(Sqlist R)
{   int i,j,low,mid,high;
    for(i=2; i<=n; i++)
        if(R[i].key<R[i-1].key)
        {   R[0]=R[i];               /* 将 R[i]暂存到 R[0] */
            low=1; high=i-1;
            while(low<=high)         /* 折半查找 R[i]的插入位置 */
```

```
        {   mid = (low + high)/2;
            if(R[0].key < R[mid].key) high = mid - 1;
            else low = mid + 1;
        }
        for(j = i - 1; j >= low; j - - ) R[j + 1] = R[j];   /* 记录后移 */
        R[low] = R[0];                                      /* 插入 */
    }
}
```

折半插入排序仅减少了关键字之间的比较次数，所需移动记录的次数和辅助空间与直接插入排序相同。

2.6.4 选择排序

选择排序(Selection Sort)的基本思想是：每一遍从待排序的 $n-i+1(i=1,2,\cdots,n-1)$ 个记录中选取关键字最小的记录作为有序序列中的第 i 个记录。下面介绍简单选择排序(Simple Select Sort)。

第一遍排序，从 n 个记录中进行 $n-1$ 次关键字的比较，选出关键字最小的记录作为有序序列中第 1 个记录；第二遍排序，从剩下的 $n-1$ 个记录中进行 $n-2$ 次关键字的比较，选出关键字最小的记录作为有序序列中第 2 个记录；第 i 遍排序，从余下的 $n-i+1$ 个记录中进行 $n-i$ 次关键字的比较，选出关键字最小的记录作为有序序列中第 i 个记录，直到所有记录排序完成为止。例如，对(25,89,15,25',4,38)进行简单选择排序，其过程如图 2.6.3 所示。

简单选择排序算法如下：

```
void SelectSort(Sqlist R)
{   int i,j,k;
    for(i = 1; i < n; i + + )
    {   k = i;
        for(j = i + 1; j <= n; j + + )
                        /* 在当前 n - i 个记录中选关键字最小的记录 */
        if(R[j].key < R[k].key) k = j;
                    /* 记下目前找到的关键字最小的记录位置 */
        if(k! = j)                          /* 交换 R[i]与 R[k] */
        {   R[0] = R[i];
            R[i] = R[k];
                R[k] = R[0];
```

```
    }
  }
}
```

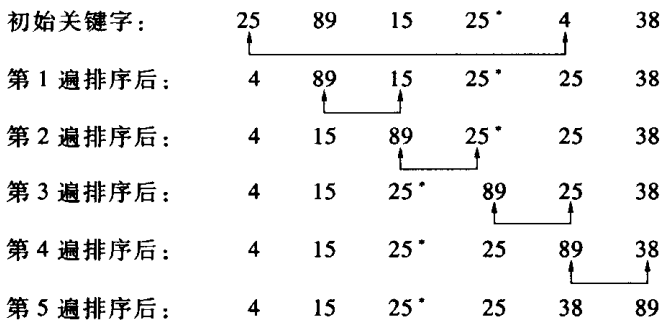


图 2.6.3　简单选择排序示例

显然，简单选择排序所需比较关键字的次数与记录的原始排列无关，在任何情况下，算法所需的比较次数都为 $\sum_{i=1}^{n-1}(n-i)=n(n-1)/2$。而记录的移动次数与记录的原始排列有关。在记录的原始排列为正序时，不需要移动记录；如果记录的原始排列为反序，由于每遍排序均要交换记录，共需要交换 $n-1$ 次，由于每次交换实际上要移动 3 次记录，故移动记录数为 $3(n-1)$。简单选择排序的时间复杂度也是 $O(n^2)$，算法需要一个记录的辅助存储空间，简单选择排序是不稳定的。

2.6.5　快速排序

快速排序的基本思想是：首先在待排序的记录中，确定一个记录 R_i，经过比较和移动，将 R_i 放到“中间”某个位置上，使得 R_i 左边所有记录的关键字小于等于 R_i 的关键字，R_i 右边所有记录的关键字大于等于 R_i 的关键字。于是，以 R_i 所在的位置为界，将记录序列划分为左、右两个子序列。然后，用同样的方法分别对这两个子序列进行划分，得到 4 个更小的子序列。继续进行下去，直到每个子序列只有一个记录为止。

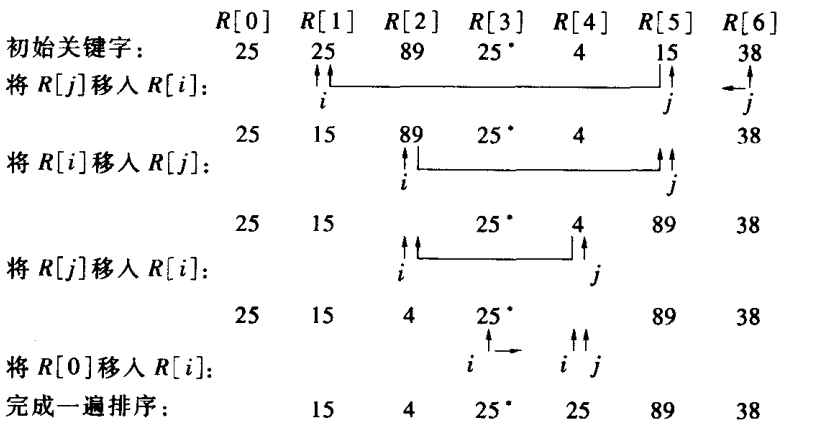
排序时，可以从待排序列中的第 1 个记录、中间一个记录与最后一个记录中选取关键字值居中的一个记录，用它来划分序列。为简单起见，这里选用第 1 个记录对序列进行划分。使用指针 i、j 从序列的左、右两端向中间方向扫描。首先令 i 指向左端第 1 个记录，j 指向右端的最后一个记录。将 i 所指的记录存入 $R[0]$ 中，然后反复进行如下两个过程，直至 $i=j$ 时为止。

(1) 指针 j 由右向左扫描，直到 $R[j].key < R[0].key$，将 $R[j]$ 移入 i 所指的位置。

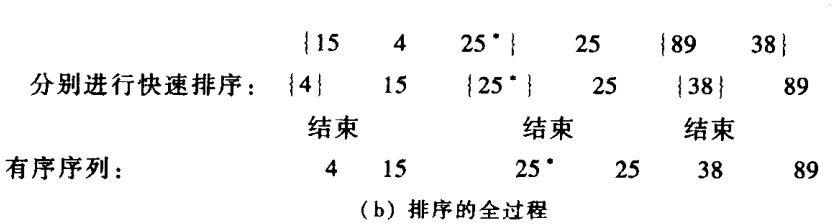
(2) 指针 i 由左向右扫描，直到 $R[i].key > R[0].key$，将 $R[i]$ 移入 j 所指的位置。

当 $i=j$ 时，i、j 所指的位置便是存放于 $R[0]$ 中的记录应该存放的位置，将 $R[0]$ 中的记录移入该位置上，便完成了对序列的一次划分。此时，以 $R[i]$ 为界，将序列划分为左、右两个子序列，左边所有记录的关键字均小于等于 $R[i].key$，右边所有记录的关键字均大于等于 $R[i].key$。之后，用同样的办法分别递归处理左、右两个子序列，直到每个子序列只有一个记录为止。

例如，对(25,89,25*, 4, 15,38) 进行快速排序，排序过程如图 2.6.4 所示。



(a) 一遍快速排序过程



(b) 排序的全过程

图 2.6.4 快速排序示例

选用第 1 个元素 25 进行划分。指针 i 指向 25，j 指向 38。将 25 移入 $R[0]$ 中。首先，指针 j 从 38 开始扫描。$38>25$，j 继续向左扫描，因 $15<25$，j 停止扫描，并将 15 移入 i 所指的位置，即 25 所在的初始位置。然后，指针 i 从 89 开始向右端扫描。因 $89>25$，停止扫描，并将 89 移入 j 所指的位置，即 15 所在的初始位置。因 $i<j$，故 j、i 继续扫描。现在，j 从 4 开始扫描。因 $4<25$，停止扫描，并将 4 移入 i 所指的位置，即 89 所在的初始位置。现在，i 从 25* 开始扫描，因 $25^*=25$，继续向右扫描，因 $i=j$，停止扫描，将存入 $R[0]$ 中的 25 移入 i、j 所指的位置，一遍快速排序完成。此时，以元素 25 为界，将原序列划分为左子序列(15,4,25*)和右子序列(89,

38)。然后再对两个子序列分别进行快速排序,如图 2.6.4(*b*)所示,最后得到有序序列。

对序列进行一次划分,即一遍快速排序的算法如下:

```
int Partition(Sqlist R,int i,int j)
{   R[0]=R[i];                          /* 用第一个记录划分序列 */
    while(i<j)                          /* 从序列两端交替地向中间扫描 */
    {   while(i<j&&R[j].key>=R[0].key) j--;     /* 从右向左扫描 */
        if(i<j) R[i++]=R[j];
              /* R[j].key<R[0].key ,将 R[j]移入 i 所指的位置,i 加 1 */
        while(i<j&&R[i].key<=R[0].key) i++;     /* 从左向右扫描 */
        if(i<j) R[j--]=R[i];
              /* R[i].key>R[0].key ,将 R[i]移入 j 所指的位置,j 减 1 */
    }
    R[i]=R[0];                          /* 划分结束,将 R[0]中的记录移入 R[i]中 */
    return i;
}
```

整个快速排序的过程可递归进行,若待排序列中只有一个记录,显然已有序,否则对划分所得的两个子序列再继续进行划分。算法如下:

```
void QSort(Sqlist R,int low,int high)
{   int pivotpos;
    if(low<high)                                    /* 子序列长度大于 1 */
    {   pivotpos=Partition(R,low,high);             /* 将 R[low..high]一分为二 */
        QSort(R,low,pivotpos-1);                    /* 对左子序列递归排序 */
        QSort(R,pivotpos+1,high);                   /* 对右子序列递归排序 */
    }
}
```

对原始序列排序,只需调用算法 Qsort,即可完成。

```
void QuickSort(Sqlist R)
{ Qsort(R,1,n);
}
```

就平均速度而言,快速排序是已知内部排序方法中最好的一种排序方法。这是由于对排序序列不断一分为二所带来的效益。

在最好情况时,每次划分所得到的两个子序列长度大致相等,其时间复杂度为 $O(n\log_2 n)$。

在最坏情况时,待排序的原始记录序列已按关键字有序或基本有序,在这种情况下,长

度为 n 的序列经一次划分后得到的两个子序列的长度分别是 0 和 $n-1$，也就是说不能进行有效的、一分为二的划分，从而失去了快速排序的优势，时间复杂度上升为 $O(n^2)$。这时宜选用插入排序或冒泡排序。可以证明，当待排序的原始记录随机排列时，其时间复杂度仍为 $O(n\log_2 n)$。快速排序采用递归算法，系统内部需要一个栈作辅助空间，用来实现递归处理左、右子序列，空间复杂度为 $O(\log_2 n)$。快速排序是不稳定的。

2.6.6 归并排序

归并排序(Merge Sort)的基本思想是：把 $K(K\geqslant 2)$ 个已排序的子序列组合在一起，形成一个有序的序列。同时归并 K 个有序子序列为一个有序序列的排序过程叫做 K 路归并排序。为简单起见，这里仅讨论二路归并排序(简称归并排序)。

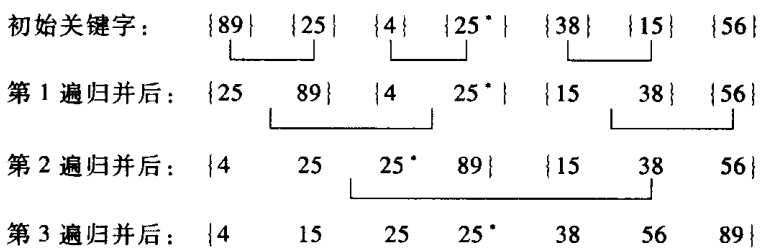


图 2.6.5　二路归并排序示例

假设初始序列含有 n 个记录，首先将这 n 个记录看成 n 个有序的子序列，每个子序列的长度为 1，然后两两归并，得到$\lceil n/2\rceil$个长度为 2(n 为奇数时，最后一个子序列的长度为 1)的有序子序列，在此基础上，再进行两两归并，如此重复，直至得到一个长度为 n 的有序序列为止。图 2.6.5 为二路归并排序的一个例子。

归并排序的基本操作是将两个位置相邻的有序子序列 $R[low..mid]$、$R[mid+1..high]$ 归并为一个有序子序列 $T[low..high]$。算法如下：

```
void Merge(Sqlist R,Sqlist T,int low,int mid,int high)
{   int i,j,k;
    k=i=low; j=mid+1;   /* i,j,k 置初值,分别指向 3 个记录区的起始位置 */
    while(i<=mid&&j<=high)   /* 两个子序列非空时,取关键字值小者放入 T 中 */
    {   if(R[i].key<=R[j].key)   T[k]=R[i++];
        else T[k]=R[j++];
        k++;
```

```
        }
        while(i <= mid) T[k + +] = R[i + +];  /* 第一个子序列非空,复制剩余记录到 T 中 */
        while(j <= high) T[k + +] = R[j + +];  /* 第二个子序列非空,复制剩余记录到 T 中 */
    }
```

设在某遍归并中,各有序子序列长度为 *length*,在将相邻的有序子序列两两进行归并时,子序列的个数可能是奇数,这时,最后一个子序列无须和其他子序列归并。另外,如果最后一个子序列长度小于 *length*,那么,上界就是原始序列的上界 n。对 $R[1..n]$ 做一遍归并排序,算法如下:

```
void MergePass(Sqlist R,int length)
{   int i = 1;
    Sqlist T;
    while(i + 2 * length - 1 < n)  /* 两两归并长度均为 length 的有序子序列 */
    {   Merge(R,T,i,i + length - 1,i + 2 * length - 1);
        i + = 2 * length;
    }
    if(i + length - 1 < n) /* 归并长度为 length 和长度不足 length 的最后两个有序子序列 */Merge(R,T,i,i + length - 1,n);
    else while(i <= n) { T[i] = R[i]; i + +;}     /* 复制最后一个子序列 */
    for(i = 1;i <= n;i + +) R[i] = T[i];        /* 将 T[1..n] 复制到 R[1..n] */
}
```

二路归并排序,算法如下:

```
void MergeSort(Sqlist R)
{   int length;
    Sqlist T;
    for(length = 1;length < n;length * = 2)
        MergePass(R,length);
}
```

归并排序是一种稳定的排序方法。对长度为 n 的序列,需进行$\lceil \log_2 n \rceil$遍二路归并,每遍待归并的记录的数量是 n,在任何情况下其时间复杂度均为 $O(n\log_2 n)$。此外,归并排序需要和待排记录等数量的辅助空间,空间复杂度为 $O(n)$。

2.6.7 排序方法的比较

排序的方法很多，各有其优缺点。从算法的时间性能、空间性能及稳定性方面考虑，对上述各种排序方法加以归纳，如表 2.6.1 所示。

表 2.6.1 各种排序方法的性能比较

排序方法	最好时间	平均时间	最坏时间	辅助空间	稳定性
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	√
插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	√
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	×
快速排序	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$	×
归并排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	√

选择排序方法时，不仅要考虑实际问题对时间复杂度、空间复杂度以及对排序稳定性的要求，还需要考虑待排序的记录个数 n、记录本身数据量的大小及关键字的分布情况。

如果待排序的记录个数 n 较小，则可选用插入排序或冒泡排序。此外，若待排序的记录序列按关键字基本有序，选用插入、冒泡排序，时间复杂度能达到 $O(n)$。

如果记录本身数据项较多，宜选用排序过程中平均移动记录次数较少的选择排序。

如果待排序的记录个数 n 较大，一般选用快速排序或归并排序。

排序方法可以综合使用。例如，在记录个数 n 较大时，采用快速排序，在划分子区间的长度小于某个设定值时调用插入排序。

根据待排序序列的规模以及对数据处理的要求，可以采用不同的排序方法。

本章小结

本章给出了数据结构的基本概念，讨论了线性表、栈、队列、树等数据结构的逻辑特性、存储表示及其主要操作，介绍了查找和排序常用的各种方法。

如前所述，数据结构研究数据的逻辑结构、存储结构及数据的操作。

数据的逻辑结构可概括为两大类：线性结构和非线性结构。线性结构的逻辑特征是：若结构非空，则有且仅有一个开始结点和一个终端结点，并且所有结点最多只有一个直接前驱和一个直接后继。线性表就是一个典型的线性结构。栈和队列是受限的线性表。栈只能在表的一端插入和删除，队列只允许在表的一端插入，而在另一端删除。非线性数据结构的特征是一个结点可能有多个直接前驱和直接后继。树是一种非线性结构，在树型结构中结点间的关系是前驱唯一而后继不唯一。

关于数据的存储结构，介绍了顺序和链式两种基本的存储方式，各有特点。顺序存储的优点是占用的存储空间少，存取元素的效率高，其缺点是需要预先根据线性表的长度分配一段连续的存储空间，每插入或删除一个数据元素，平均需要移动表中一半的元素，数据移动量大。链式存储的优点是可以充分利用内存空间，只要内存尚有空闲，就不会产生溢出，不需要预先分配连续的存储单元；此外，元素的插入，删除只需要修改相关结点的指针域，不需要移动大量的数据元素。链式存储的缺点是链表中的每个结点，除了数据域外，还要额外设置指针域，存储空间的利用率不如顺序存储结构高；此外，对表中结点的访问必须从头指针开始沿着结点的链域往下搜索才能获得。同一种逻辑结构可以采用不同的存储方式而得到不同的存储结构。采用不同的存储结构，其数据处理的效率是不同的。因此，在进行数据处理时选择合适的存储结构是很重要的。

数据的操作是在数据的逻辑结构上定义的运算，如插入、删除、修改、查找、排序等，存储结构不同，算法的实现和效率是不同的。

查找分为静态查找和动态查找。顺序查找、折半查找、分块查找都是静态查找方法，二叉排序树查找是动态查找方法。效率较高的折半查找只能在有序顺序表上进行；效率次之的分块查找要求查找表是索引顺序表；顺序查找效率最低，对查找表无要求。

排序方法介绍了冒泡排序、插入排序、选择排序、快速排序和归并排序。每种方法都各有特点，应根据实际应用灵活选择。

讨论以上各问题的主要目的是为了提高数据处理的效率。所谓提高数据处理的效率，主要包括两个方面：一是提高数据处理的速度，二是尽量节省在数据处理过程中所占用的存储空间。

实际问题往往是复杂的，需要分析研究数据的特性，以便为应用所涉及的数据选择适当的逻辑结构、存储结构及其相应的操作算法。

习 题 二

1. 简述下列术语：数据，数据元素，数据类型，数据对象，数据结构，逻辑结构，存储结构。
2. 数据元素之间的逻辑关系有哪几类？
3. 描述以下 3 个概念的区别：头指针，表头结点，首结点。
4. 线性表有两种存储结构：顺序表和链表。它们各有哪些主要优缺点？
5. 给定一个带表头结点的单链表，试写出计算此单链表长度的算法。
6. 编写一个连接两个单链表的算法。设有链表 $A=(a_1,a_2,\cdots,a_n)$，$B=(b_1,b_2,\cdots,b_m)$，其中 m,n 均大于或等于零，连接后产生一个新链表 $C=(a_1,a_2,\cdots,a_n,b_1,b_2,\cdots,b_m)$。
7. 分别在不同存储结构上实现将两个有序表合并成一个有序表的运算。

8. 按照运算符优先数法，画出对下列算术表达式求值时操作数栈和运算符栈的变化过程：$A+B*C+(E-D)/F*G$。

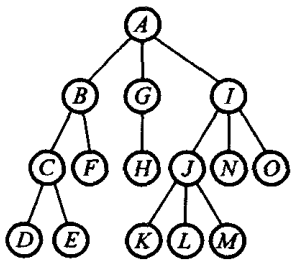


图 1 树

9. 对于图 1 所示的树，回答下列问题：

（1）树的根是哪个结点？哪些是叶子结点？哪些是非终端结点？

（2）树的度是多少？各个结点的度是多少？

（3）树的深度是多少？各个结点的层数是多少？以结点 B 为根的子树的深度是多少？

（4）对于结点 J，它的双亲是哪个结点？它的祖先是哪些结点？它的孩子是哪些结点？它的兄弟和堂兄弟分别是哪些结点？

10. 对于图 2 所示的二叉树，画出它的顺序存储结构及二叉链表表示。

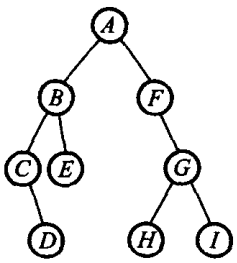


图 2 二叉树

11. 举例说明，一棵度为 2 的树与一棵二叉树有何区别？

12. 试分别画出具有 3 个结点的树和 3 个结点的二叉树的所有不同形态。

13. $n(n>1)$个结点的各棵树中，其中深度最小的那棵树的深度是多少？它共有多少叶子和多少非终端结点？其中深度最大的那棵树的深度是多少？它共有多少叶子和多少非终端结点？

14. 对于 n 个结点的完全二叉树，用 1 到 n 的连续整数顺序编号，试回答下列问题：

（1）它共有多少层？各层的结点数分别是多少？

（2）各层最左边的结点的编号分别是多少？各层最右边的结点的编号分别是多少？

（3）对于编号为 $i(1\leqslant i\leqslant n)$的结点，它的层是多少？它的双亲（若存在的话）的编号是多少？它的左孩子（若存在的话）和右孩子（若存在的话）的编号分别是多少？

15. 试用数列 30,5,18,32,38,20,15 生成一棵二叉排序树,然后进行前序、中序、后序遍历。

16. 对有序表(2,3,10,15,20,30,40,60)进行折半查找,若要查找值为 3 的元素,则将依次与哪些元素进行比较?若要查找值为 45 的元素,则又将依次与哪些元素进行比较?

17. 给定一组关键字(19,02,26,92,87,11,43,87,21),分别用下列方法排序,试列出每一遍排序后关键字的排列次序:

(1) 冒泡排序。

(2) 插入排序。

(3) 选择排序。

(4)快速排序。

(5)归并排序。

18. 从 100 000 个数中求出 3 个最大的数,试问:应该采用什么样的方法最好?为什么?如果是求出 50 000 个最大的数,又应该采用什么样的方法最好?为什么?

第三章　操作系统及应用

随着计算机技术的发展，计算机应用日益普及。计算机在科学研究、人类活动以及社会生活的各个领域得到了广泛的应用，从办公自动化到数据、图像处理，从工业控制到科学计算，从远程教育到电子商务……计算机应用无处不在，无处不有！当前，大量的计算机应用并不局限在单机系统上，而是更多地运行在并行计算机、局域网络、甚至广域网的环境上。人们越来越多地与复杂的应用环境打交道，越来越多地与计算机打交道。那么，人们如何与计算机交往？计算机如何根据应用的需要去管理繁杂的数据和各类资源？它又是如何处理大量的用户和应用提出的请求？这些都涉及到计算机系统资源的管理、多用户多任务活动的处理。这些与系统有关的、大量的工作都是由配置在计算机上的一个重要系统软件——操作系统来解决的。现代的计算机几乎没有不配置操作系统的。那么，操作系统是什么，有什么特点，能做些什么，它在计算机系统中的地位如何，它的功能是如何实现的？这些问题是本章讨论的主要内容。

3.1　操作系统概述

3.1.1　计算机系统的组成与操作系统的位置

1. 计算机系统的组成

计算机系统是一个整体概念，不论是大型机、小型机还是微型机，都是由两大部分组成的：计算机硬件部分和软件部分。

硬件指的是组成计算机的任何机械的、磁性的、电子的装置或部件。硬件也称为硬设备，它由中央处理机（CPU）、存储器、输入/输出（I/O）设备等组成。CPU 还包含若干个寄存器，用来存储一些暂时的结果和其他控制信息。每一个寄存器都具有特定的功能，其中，有一个很重要的寄存器称为程序计数器（PC），它指示下一条应该执行的指令的地址。

存储器是计算机存储程序和数据的部件。它分为主存储器（简称主存或内存）和辅助存储器（简称辅存或外存）两类。主存储器是中央处理机可以直接读、写信息的存储器，是计算机处理信息的工作场所。主存储器现在一般用集成电路芯片组成，它的存取速度比较快。辅助存储器能保存大量的数据信息，它容量大、价格较主存储器便宜，但存取速度较慢。辅存上的信息不能被中央处理机直接存取。如要使用必须先送入主存，之后才能被 CPU

使用。

输入/输出设备则是完成信息传输任务的设备。当某一问题需要计算机处理时，必须先给定程序和初始数据，这些程序和数据通过输入设备进入计算机。在程序运行过程中或结束后，必要的数据信息或计算结果要通过输出设备传送给用户。为了满足不同应用的需要，输入/输出设备的品种是多种多样的。

这些部件一般采用总线结构组织在一起，典型的单处理机系统结构已在 1.2.2 节中的图 1.2.1 中描述。由这些硬部件构成的机器称为裸机，它是计算机系统的物质基础和最基本的硬件环境。

任何应用或用户使用计算机时，面对的绝不是裸机这种工作环境，而是配置了操作系统和其他应用软件的计算环境。因为裸机上没有任何一种可以协助它们解决问题的手段，只提供最低级的机器语言。为了对硬件的性能加以扩充和完善，为了方便用户上机，在裸机外添加了能实现各种功能的软件程序。在这些软件中有一个很重要的软件系统称为操作系统，它管理系统中所有的硬、软设备，并组织整个计算机的工作流程。

软件一般可以分为以下几类。

- 系统软件：操作系统、编译系统、程序设计语言、连接装配程序、系统实用程序等。
- 工具软件：各种诊断程序、检查程序、引导程序等。
- 应用软件：应用程序、软件包（如数据统计软件包、运筹计算软件包等）。

通常，将与计算机系统密切相关的程序称为系统程序，这样，也可以将工具软件划归到系统软件范畴。而应用程序则是为各种应用目的而研制的程序。

依上所述，计算机系统的组成由图 3.1.1 说明。

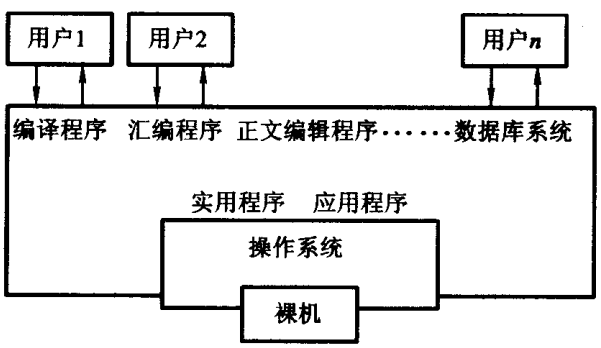


图 3.1.1　计算机系统的组成

裸机是计算机系统的物质基础，没有硬件就不能执行指令和实施最原始、最简单的操作，软件也就失去了效用；而若只有硬件，没有配置相应的软件，计算机就不能发挥它潜在的能力，这样硬件也就没有活力。因此，硬件和软件这二者是互相依赖、互相促进的。只有软

件和硬件有机地结合在一起的系统，才能称得上是一个计算机系统。操作系统将系统中的各种软、硬资源有机地组合成一个整体，使计算机真正体现了系统的完整性和可利用性。

2. 操作系统在计算机系统中的位置

计算机系统是由硬件和软件两大部分组成的一个完整的系统。由所有硬件组成的裸机处于系统的最底层，裸机的外面是软件部分。软件是相当丰富的，根据它们的功能和使用特性，又可以分为两大类：系统软件和应用软件。应用软件是在系统软件的支持下完成各项工作的，所以它应在系统软件的外层。系统软件中操作系统处于核心地位，它负责整个系统的管理和控制，是其他系统软件和应用软件运行的基础，所以，在所有软件中操作系统处于最内层并直接与裸机相接，与计算机硬件的关系最为密切。

图 3.1.1 描述了计算机系统的组成，为了更清楚地说明操作系统在计算机系统中的位置，现用图 3.1.2 说明。

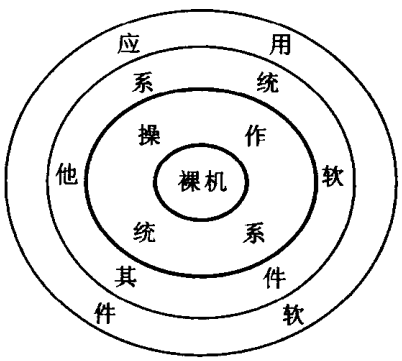


图 3.1.2 操作系统在计算机系统中的位置

3. 操作系统与硬件及其他软件的关系

在计算机系统中，操作系统位于裸机之上、所有软件最内层，所以，它与计算机硬件和其他软件有着密切的联系。

首先，操作系统是裸机上扩充的第一层软件，负责所有硬部件的分配、控制工作。如负责中央处理机(CPU)、主存储器、磁盘、打印机、显示器、键盘等设备的分配，以及设备的驱动和存取，使它们在操作系统控制之下正常、有效地工作。

其次，操作系统与其他系统软件和应用程序是一种管理与被管理、调用与被调用的关系。所有软件都在操作系统的管理和控制之下工作。另外，操作系统与其他系统软件，如各种程序设计语言及相应的编译程序、连接装配程序、各种服务性程序和标准程序一起为应用程序提供一个运行环境。

计算机硬件、操作系统和其他系统软件及应用软件共同工作，为用户提供了良好的工作环境。在这 3 个部分中，操作系统是系统的控制中心，它管理和控制硬件的工作，控制和调

度各种系统软件。但在操作系统之上运行的还必须有其他系统软件和应用软件，当系统提供丰富的系统软件和应用软件时，用户才能更方便、直观地使用计算机。

裸机是操作系统运行的基础和环境，它的体系结构对操作系统实现技术和方法有一定的约束和制约；而用户或应用软件对操作系统又会提出各种服务要求，这些因素对如何确定操作系统功能、如何实现操作系统的功能将产生极大的影响。

3.1.2 多道程序设计技术与分时技术

操作系统在现代计算机中起着相当重要的作用。它是因客观需要而产生，是随着计算机技术的发展和计算机应用的日益广泛而逐渐发展和完善的。它的功能由弱到强，在计算机系统中的地位也不断提高，以至成为系统的核心。研究操作系统的形成和发展要用一种历史的观点去分析，以便从中体会到操作系统产生的必然性和促使它发展的根本原因。

1. 操作系统发展的阶段

第一台电子计算机于 1946 年问世。此后，计算机在其运算速度、存储容量、外部设备的功能和种类等方面都有了惊人的发展和提高。人们通常按照元件工艺的演变把计算机的发展过程分为 4 个阶段：

1946 年 ~ 1955 年　第一代计算机（电子管时代）。

1955 年 ~ 1965 年　第二代计算机（晶体管时代）。

1965 年 ~ 1980 年　第三代计算机（集成电路时代）。

1980 年至今　第四代计算机（大规模集成电路时代）。

计算机现在正向着巨型化、微型化、网络化、智能化几个方向发展。与计算机硬件发展同步，为适应客观应用的需要，操作系统也经历了如下的发展阶段：

第一代计算机　手工操作阶段（无操作系统）。

第二代计算机　批处理（早期）、执行系统。

第三代计算机　操作系统形成——批处理操作系统、分时操作系统、实时操作系统。

第四代计算机　具有图形化操作界面的个人计算机操作系统、网络操作系统、分布式操作系统。

在手工操作阶段，无任何软件，人们通过物理地址编程、人工操作方式使用计算机。当主机的速度提高后，由于人工操作的慢速度严重影响了计算机效率的发挥，为解决人—机矛盾出现了批处理系统。当主机速度不断提高，硬件产生了通道技术、中断技术后，又出现了能支持 CPU 与外部设备并行操作的批处理系统。但不久发现，这种并行是有限度的，并不能完全消除中央处理机对外部传输的等待。因为，在单道程序工作时，在其进行 I/O 处理时，处理机必然处于空闲状态，其原因是程序的输入/输出与本道程序相关。为了充分挖掘计算机的效率，必须在计算机系统主存中存放多道程序，使其同时运行，这就是多道程序设计技术。在批处理系统中采用多道程序设计技术就出现了多道批处理系统、分时系统。这

两类操作系统的出现标志着操作系统的形成。

多道程序设计技术是现代操作系统实现技术的基础。现代操作系统都提供多用户、多任务的运行环境，它的核心是具备支持多个程序同时运行的机制。在操作系统发展过程中，多道程序系统是十分关键的发展阶段，因为，它提出了多道程序设计的概念和解决多道运行的一套方法，使操作系统向着具备多任务并发和资源共享特征的方向发展。

2. 多道程序设计技术

(1) 什么是多道程序设计技术

为了了解多个程序同时进入计算机系统时的运行情况，首先讨论单道程序的运行。当单道程序进入计算机系统后，它需要在中央处理机(CPU)上运行，也可能需要进行 I/O 工作(例如输入一批数据)，这时需请求操作系统服务。操作系统帮助启动输入设备，进行数据输入工作，这时输入设备忙碌而中央处理机空闲等待。当数据输入完成后，由操作系统进行 I/O 完成处理后返回用户程序继续计算。这一过程可由图 3.1.3 描述。

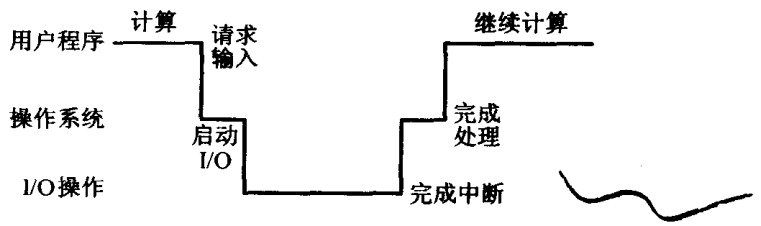


图 3.1.3 单道程序工作示例

从图 3.1.3 中可以看出，当外部设备进行传输工作时，CPU 处于空闲等待状态，反之，当 CPU 工作时，I/O 设备又无事可做。这说明，计算机系统各部件的效能没有得到充分的发挥，其原因在于主存中只有一道程序。在计算机价格十分昂贵的 20 世纪 60 年代，提高设备的利用率是首要目标。为此，人们设想能否在系统中同时存放几道程序，这就引入了多道程序设计的概念。

多道程序运行情况由图 3.1.4 来说明。用户程序 A 首先在处理机上运行，当它需要从光标记阅读机输入新的数据而转入等待时，系统帮助它启动光标记阅读机进行输入工作，并让用户程序 B 开始计算，直到程序 B 需要进行输入或输出处理时，再启动相应的外部设备进行工作。如果此时程序 A 的输入尚未结束，也无其他用户程序需要计算，则处理机就处于空闲状态，直到程序 A 在输入结束后重新执行。若当程序 B 的 I/O 处理结束时，程序 A 仍在执行，则程序 B 继续等待，直到程序 A 计算结束请求输出时，才转入程序 B 的执行。从图 3.1.4 中可以看出，在有两道程序执行的情况下，CPU 的效率已大大提高。因此，当有多道程序工作时，CPU 将几乎始终处于忙碌状态。

多道程序设计是一种软件技术，该技术使同时进入计算机主存的几个相互独立的程序，

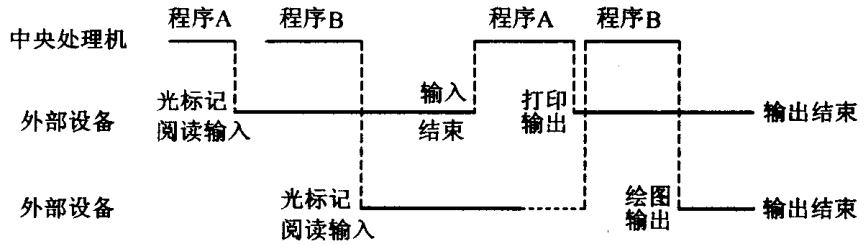


图 3.1.4　多道程序工作示例

在管理程序控制之下相互穿插地运行。当某道程序因某种原因不能继续运行下去时(如等待外部设备传输数据),管理程序便将另一道程序投入运行。这样可以使中央处理机及各外部设备尽量处于忙碌状态,从而大大提高计算机的使用效率。

(2) 多道运行的特征

多道程序设计技术使得几道程序在系统内同时工作。那么,几道程序怎么能在系统内同时执行呢? 应该看到,计算机系统中除中央处理机外,还有各种不同的输入设备、输出设备,虽然对中央处理机而言,一个时刻只能有一道程序在上面运行,但从整个计算机系统来看,CPU、输入设备、输出设备是有可能同时操作的。例如,正在处理机上运行的程序 A 因为要进行输入操作而让出 CPU 给程序 B 运行,当程序 B 运行一段时间后又要求做输出操作,这时,CPU 让程序 C 运行。若程序 A 的输入操作、程序 B 的输出操作没有结束,从整个计算机系统来看,程序 A 正在做输入工作,程序 B 正在做输出工作,程序 C 的计算工作正在进行。从宏观上说,这几道程序都处于执行状态,称这几道程序在并发执行。当然,就 CPU 而言,这几道程序实际上是在轮流使用它,当一道程序运行不下去时,CPU 才让另一道程序运行。

综上所述,多道运行的特征是:

① 多道。即计算机主存中同时存放几道相互独立的程序。

② 宏观上并行。同时进入系统的几道程序都处于运行过程中,即它们先后开始了各自的运行,但都未运行完毕。

③ 微观上串行。从微观上看,主存中的多道程序轮流地或分时地占用处理机,交替执行。

3. 分时技术

让操作员(用户)通过控制台(终端)直接操作、控制自己程序运行的操作方式称为联机工作方式。用户十分欢迎这种工作方式。因为,在这种方式下,一方面操作员(用户)可以通过控制台(终端)向计算机发出各种控制命令,使系统按自己的意图控制程序的运行;另一方面,系统在运行过程中输出一些必要的信息,如给出提示符、报告运行情况和操作结果,

以便让用户根据此信息决定下一步的工作，这样，用户和机器可直接采用问答方式来完成他的作业，即在人机之间进行“会话”。分时系统采用这种联机工作方式。在分时系统中，一个计算机和许多终端设备连接，每个用户可以通过终端向系统发出命令、请求完成某项工作，而系统则分析从终端设备发来的命令、完成用户提出的要求；之后，用户又根据系统提供的运行结果，向系统提出下一个请求，这样重复上述“会话”过程，直到用户完成预期的全部工作为止。

当一台计算机同时为多个用户服务时，如何能保证各用户发出的命令，计算机都能及时响应，使每个用户好像独占计算机一样？为达到这一目标，系统必须采用分时技术。

所谓分时技术，是指把处理机时间划分成很短的时间片（如几百毫秒）轮流地分配给各个联机用户使用。如果某个用户程序在分配给它的时间片用完之前还未完成，则中断该用户程序的执行，等待下一轮继续计算，此时处理机让给另一个用户使用。这样，每个用户的各次要求都能得到及时、快速的响应，好像他独占一台计算机一样。

3.1.3 操作系统的定义

如图 3.1.5 所示，计算机系统拥有丰富的硬件、软件资源，操作系统的主要功能是要管理这些资源。在高档微型机、小型机或大机器上配置的操作系统大多为多用户、多任务操作系统，其资源管理的功能非常复杂。因为多个用户或多任务共用一个计算机系统时，会产生资源共享的问题，而共享必将导致对资源的竞争。资源竞争是多个计算任务对计算机系统资源的争夺。例如，当某计算机系统配置好后有这样一些部件：一台处理机，两台输入机，一台打印机，一个硬盘驱动器。假定该系统有 4 个用户，当这些用户程序同时投入运行时，它们都要用 CPU 计算，都要输入数据，都要打印结果……。因此，必然会出现竞争局面，即竞争占用 CPU 的时间，竞争主存空间，竞争 I/O 设备和存储设备，竞争使用公用子程序等。这种局面是为了充分利用系统资源而必然会出现的。为了使这些用户程序能正常运行和对资源争而不乱，必须有办法将系统资源有效地管起来，并协调各用户程序之间的关系和组织整个工作流程，这些工作就是由操作系统来实现的。

一方面，操作系统有效地管理系统资源以充分发挥它们的作用，另一方面，还极大地方便了用户。如果没有配置操作系统，让用户直接使用裸机，用户将会束手无策。比如，用户要在裸机上运行自己的程序，并在打印机上输出一串字符，那么用户就得自己编写输入/输出程序。为此用户要了解这台设备的命令寄存器、数据寄存器的使用方法、设备的启动地址以及如何发启动命令等问题，这些细节对于用户而言是十分麻烦的，用户要这样来用机是不可能的，而且在多用户的情况下也是绝对不允许，因为如果每个用户都能发启动设备的命令，将造成系统混乱。所有这些工作只能由操作系统来做，当配置了操作系统后，用户通过操作系统使用计算机。操作系统是用户和系统的界面，系统内部虽然非常复杂，但这些复杂性是不显现在用户面前的。计算机通过操作系统可向用户提供一个功能很强的系统；用户

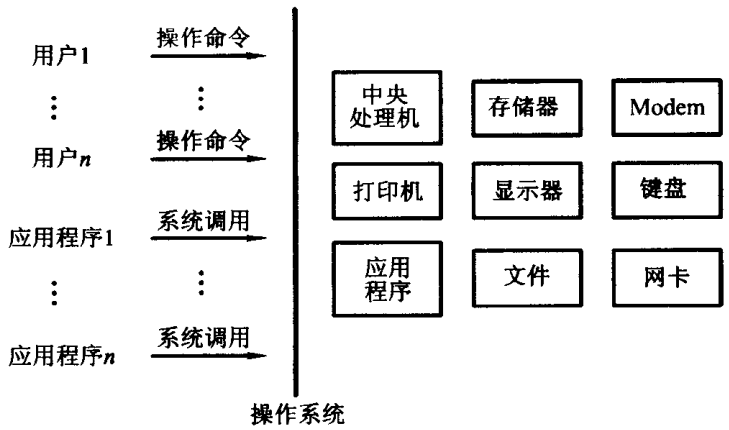


图 3.1.5 操作系统的作用

可以使用操作系统提供的命令,简单、方便地把自己的意图告诉系统,以完成所需完成的工作。正是由于操作系统卓越的工作,才能既充分地利用系统的资源,又使用户能方便地使用计算机。

综上所述,操作系统是一个大型的程序系统,它负责计算机的全部软、硬资源的分配、调度工作;控制并协调多个任务的并发活动;提供用户接口,使用户获得良好的工作环境。操作系统使整个计算机系统实现了高度自动化、高效率、高利用率和高可靠性,操作系统是整个计算机系统的核心。

3.1.4 操作系统的功能

操作系统的首要任务是对系统资源的管理。在单用户系统中,资源管理任务相对来说比较简单,而在多用户(多任务)系统中,资源管理的任务要复杂一些,因为它要解决资源共享的策略和方法问题。在本节中,将从多用户(多任务)共用一个计算机系统的角度来讨论操作系统的资源管理功能,因为单用户(单个任务)是所讨论问题的一个特例。

当多用户(多任务)共享系统资源时,提出了一些新的问题:竞争 CPU 时间,竞争主存空间,竞争 I/O 设备和存储设备,竞争软件资源,操作系统必须有相应的功能解决这些矛盾。下面讨论操作系统的资源管理功能。

1. 处理机管理

计算机系统中最重要的资源是中央处理机,没有它,任何计算都不可能进行。在处理机管理中,最关心的是 CPU 时间。如何使用处理机时间,多数情况下,计算机为了等待 I/O 操作,而使 CPU 时间浪费几乎一半。为了提高 CPU 的利用率,现代操作系统都支持多用户、多任务的共同运行。对众多用户的算题任务(或称为一道作业),操作系统首先要确定选择

哪几个作业进入系统，让它们获得使用处理机的权利，这是处理机的高级调度；而当这些作业进入系统后，它们对应的程序什么时候能真正在处理机上运行，还需要进行处理机的低级调度。这就是处理机的调度层次问题，一般系统具有处理机的两级调度。在进行处理机的低级调度时，一个时刻只能有一个程序在处理机上运行，即采用“微观上串行”的方法，此时，需要解决 CPU 先分给哪个用户程序？它占用多长时间？下一个又该轮到哪个程序执行的问题？另外还需解决如何让选中的用户程序真正得到处理机的控制权的问题？

处理机管理的功能包括：处理机的调度层次；确定调度策略、给出调度算法；具体实施 CPU 的分派工作。

2. 存储器管理

在任何一个计算机系统中，第二位缺乏的资源是主存，对于小型和微型计算机也是如此。若系统采用了多道程序设计技术，则要求存储管理应具备以下几个方面的功能。

（1）存储分配和存储无关性

若有多个用户程序同时进入系统，它们各自都需要占用一定的存储空间。这些程序和数据分别安置在主存的什么位置？各占多大区域？这就是存储分配问题。在多用户系统中，为了提高主存空间的利用率，需要实现动态的主存分配，同时，也方便用户的程序开发。因为任何应用程序都无法预知存储管理部件（模块）会将其分配到主存什么地方，而且程序员也希望摆脱存储地址、存储空间大小等细节问题。为此，用户编程时不用关心具体的物理地址，而是用虚地址或称逻辑地址编程；对系统而言，应采用动态分配方式，即根据当时主存的使用情况确定用户程序在主存中的位置，这就是“存储无关性”的概念。存储管理部件应具有“地址重定位”的能力，也就是要提供动态地址映像机构。

（2）存储保护

在多道运行的情况下，主存中可同时存放几道用户程序。为了防止某道程序干扰、破坏其他用户程序或系统程序，存储管理必须保证：每个用户程序只能访问各自的存储空间，不能存取、破坏任何其他范围内的信息。这就要求系统提供存储保护的手段。存储保护必须由硬件提供支持，具体保护措施有基址、界限寄存器法、存储键和锁等方法。

（3）存储扩充

现代操作系统将系统资源分为两个概念：①物理资源，或称为实资源；②逻辑资源，或称为虚资源。这样处理，既有利于资源的动态分配，又方便用户使用计算机系统的资源。当对主存区分为存储空间和虚拟空间后，操作系统在硬件支持下，实现了主存和辅存之间信息的动态调度，即通过磁盘、磁鼓等辅助存储器去扩充主存空间，实现这种功能的软件技术称为“虚拟存储器”。而系统要提供这一功能，就必须具备“存储无关性”，即用户编程时，与实际的存储空间无关。

3. I/O 管理

I/O 管理是操作系统中最庞杂、琐碎的部分，其原因是：① 这部分涉及很多实际的物理

设备,它们的品种繁多、用法各异;② 各种外部设备都能和主机并行操作,而且有的设备可被多个程序所共享;③ 主机和外部设备以及各类外部设备之间的速度极不匹配,级差很大 。

基于这些原因,设备管理主要解决以下问题。

(1) 设备分配

各个用户程序在其运行的开始、中间或结束阶段都可能有输入或输出工作,因此需要请求使用 I/O 设备。在一个系统中配置的设备种类和台数总是有限的,一般少于使用者的个数,那么,制定设备分配的策略和实施分配是很重要的。对于设备分配通常有 3 种基本技术:独享、共享和虚拟技术。

(2) 设备的传输控制

设备管理模块要完成用户提出的传输要求,就必须实现设备的传输控制,包括组织使用设备的有关信息,设备启动、中断处理、结束处理等工作。

(3) 设备无关性

设备品种繁多,用法各异,为了方便用户的使用,操作系统应屏蔽其复杂性,为用户提供使用方便的 I/O 系统调用。用户可以根据自己的习惯给所需使用的设备指定一个逻辑名字,然后指出要进行的操作、数据传送的源(或目的地),操作系统帮助用户完成所需的传输工作。这里涉及到"设备无关性",即用户在程序中或资源申请命令中使用设备的逻辑名,而与实际设备无关,这称为"设备无关性"。这一特征不仅为用户使用设备提供了方便,而且也提高了设备的利用率。

4. 软件资源管理

软件资源就是程序和数据以及文档资料的集合。程序又分为系统程序和用户程序,系统程序包括操作系统的功能模块、系统程序库和实用程序。这些系统程序是以文件形式组织、存放、提供给用户使用的。为了实现多个用户对系统程序的有效存取,这些程序必须是可重入的,这比创建多个文件副本要好得多。而用户为完成自己的算题任务,也需要将程序和数据组织成文件的形式存储起来。

软件资源管理(也就是文件系统)要解决的问题是:为用户提供一种简便的、统一的存取和管理信息的方法,负责对信息的组织,实现文件共享、数据的存取控制和保密等问题,并负责磁盘空间的分配和管理工作。

综上所述,操作系统的主要功能是管理系统的软、硬件资源。这些资源按其性质来分,可以归纳为 4 类:处理机、存储器、外部设备和软件资源。针对这 4 类资源,操作系统就有相应的资源管理程序:处理机管理、存储管理、设备管理和软件资源管理程序。这一组资源管理程序就组成了操作系统这一程序系统。分析这些资源管理程序的功能和实现方法就是操作系统的资源管理观点。

3.1.5 操作系统的类型

在操作系统发展过程中，为了满足不同应用的需要而产生了不同类型的操作系统。根据使用环境和对计算任务的处理方式的不同，操作系统的类型主要有以下几种：批量操作系统；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. 批量操作系统

批量操作系统是操作系统的一种类型，它将用户提交的作业成批地送入计算机，然后，由作业调度程序选择作业运行。这样能缩短作业之间的交接时间，减少处理机的空闲等待时间，从而提高了系统的效率。计算中心大都会配置这类操作系统，若一个计算机上配置了批量操作系统，则称该系统为批处理系统。

批处理系统的主要特征是“批量”。当用户要使用计算机时，必须事先准备好自己的作业，然后交给计算中心。由计算中心的操作员将一批作业送入系统，计算结果也是成批进行输出。作业的执行采用“多道”形式，在作业执行过程中，用户不能直接进行干预。

批量操作系统的特点是：系统吞吐量高，若能合理搭配作业，选择好的调度算法，可以大大提高系统资源的利用率。但也存在着周转时间长、用户使用不方便等缺点。在批处理系统中，周转时间通常是几小时或几天。有的用户作业很短，只需几分钟便可做完，但他却要等较长的时间才能得到结果，这对用户来说是不合适的。另外，当用户把作业交给系统后；就不能直接控制作业的运行，了解其运行情况，而且在提交作业之前要对作业的处理进行周密考虑，例如要考虑遇到意外情况时的处置问题，这对用户来说也是比较麻烦和不方便的。

2. 分时操作系统

分时操作系统是操作系统的另一种类型，它采用时间片轮转的策略，使一台计算机同时为多个终端用户服务。对每个用户都能保证足够快的响应时间，并提供交互会话功能。分时系统通过给每个用户提供一个“个人计算机”的方法来提高整个系统的效率。它与批量系统之间的主要差别在于，它采用的是联机工作方式，每个用户通过各自的终端使用计算机。

用户要使用计算机之前，首先进行呼叫，当呼叫成功后，终端即和主机连接上，在终端设备上应有一条引导信息，告诉用户：“终端设备与系统已连接好”。这时，终端用户应打入一条“录入”命令，向系统申请录入一个作业。一般录入命令应给出以下参数：用户名、作业名、口令、资源需求等。系统接收命令，检查口令是否符合，资源是否能够满足。如检查通过了，系统在终端上显示“允许录入”，用户就可以进行通信工作了。有的分时系统比较简单，只要输入用户名和口令，当系统检查这两项正确时，即可进行通信工作。在通信这一阶段，用户可以从终端输入作业的有关程序和数据，输入各种键盘命令和系统进行直接的“对话”，以控制程序的运行。最后全部操作完成，用户输入“告辞”命令，退出计算机。

分时系统具有以下特点：

① 同时性。即众多联机用户可以同时使用一台计算机。

② 独占性。分时操作系统采用时间片轮转的办法使一台计算机同时为众多终端用户服务,因此,客观效果是这些用户彼此之间都感觉不到别人也在使用这台计算机,好像只有自己独占计算机一样。一般分时系统在3秒之内响应用户要求,用户就会感到满意,因为此时用户在终端上感觉不到需要等待。

③ 交互性。用户与计算机之间进行"会话",用户从终端打入命令,提出计算要求,系统收到命令后分析用户的要求并完成之,然后把运算结果通过屏幕或打印机告诉用户,用户可以根据运算结果提出下一步要求,这样一问一答,直到全部工作完成。

3. 实时操作系统

早期的计算机基本上用于科学和工程问题的数值计算。50年代后期,计算机开始用于生产过程的控制,形成实时系统。当计算机进入集成电路时期后,由于机器性能提高、计算机系统的功能增强,使得计算机的应用领域越来越宽广。这些应用有:炼钢、化工生产的过程控制,航天和军事防空系统中的实时控制。在信息管理方面的应用有:仓库管理、医疗诊断、教学系统、气象、地质勘探直到图书检索、飞机订票、银行储蓄、出版编辑管理等。

实时操作系统是操作系统的又一种类型,它对外部输入的信息能够在规定的时间内处理完毕并作出反应。"实时"二字的含义是指计算机对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作出快速反应。实时系统对响应时间的要求比分时系统更高,一般要求秒级、毫秒级甚至微秒级的响应时间。

电子计算机应用到实时控制中,配置实时操作系统,组成各种不同的实时系统。实时系统按其使用方式不同分为两类:

(1) 实时控制

在生产过程、武器控制系统、医疗控制等实时应用中,都有被控制的对象。实时控制系统通过传感器或特殊的外围设备获取被控对象产生的信号(如温度、压力、流量等的变化),然后对获得的数字或模拟信号进行处理、分析,作出决策,激发一个改变可控过程活动的信号,以达到控制的目的。

实时响应指的是,从实时信号来临,到计算机对该事件加工处理完毕,控制信号到达被控对象这段时间间隔。这一响应时间要足够快,并且要能预测在系统响应、处理过程中的主要开销,这样才能满足实时处理的需要,不然,一旦发生失控,就会出现生命、财产的巨大损失。

(2) 实时信息处理

计算机还有一类很重要的实时性应用,即组成实时信息处理系统。比如,自动订购飞机票系统、情报检索系统等。这一类应用大多数用于服务性工作,如预订一些飞机票、查阅一种文献资料。用户可以通过终端设备向计算机提出某种要求,而计算机系统处理后将通过终端设备回答用户。

实时系统主要为联机实时任务服务，特点如下：

① 系统对外部实时信号必须及时响应，响应的时间间隔要足以能够控制发出实时信号的那个环境。

② 实时系统要求有高可靠性和安全性，系统的效率则是放在第二位。

③ 系统的整体性强。实时系统要求所管理的联机设备和资源，必须按一定的时间关系和逻辑关系协调工作。

④ 实时系统没有分时系统那样强的交互会话功能，通常不允许用户通过实时终端设备去编写新的程序或修改已有的程序。实时终端设备通常只能作为执行装置或询问装置。

实时系统大部分是为特殊的实时任务设计的，这类任务对系统的可靠性和安全性要求很高。所以，系统的所有部分通常是采用双工方式工作的。

4. 个人计算机操作系统

20 世纪 80 年代以来，由于微电子技术、计算机技术、计算机体系结构的迅速发展和用户使用计算机的需求不断增长，使得操作系统沿着个人计算机、视窗操作系统、网络操作系统、分布式操作系统方向发展。

随着微电子技术的发展，使个人计算机的功能越来越强、价格越来越便宜；另一方面计算机应用的日益广泛，渗透到各行各业、个人和家庭。在个人计算机上配置的操作系统称为个人计算机操作系统。

在个人计算机（或工作站领域）有两种主流操作系统：一种是 Microsoft 公司的磁盘操作系统和具有图形用户界面的 Windows 系统，另一种是 UNIX 系统。Windows 操作系统具有图形用户界面、使用直观方便，Microsoft 公司的另一个操作系统是 Windows NT，它满足工作站平台、局域网超级服务器的需要。而 UNIX 系统是一个多用户分时操作系统，自 1969 年问世以来十分流行，它运行在高档个人计算机到大型机等各种不同处理能力的机器上，提供良好的工作环境，具有可移植性、安全性，提供很好的网络支持功能，大量被用于网络服务器。

5. 网络操作系统

一些独立自治的计算机，利用通信线路相互连接形成的一个集合体称为计算机网络。这里要求计算机是独立自治的，即计算机网络中的各个计算机是平等的，任何一台计算机都不能强制性地启动、停止或控制另一台计算机。互连指的是两台计算机之间能彼此交换信息，这种连接不一定必须经过导线，也可以采用激光、微波和地球卫星来实现。

在计算机网络中，每个主机都有操作系统，它为用户程序运行提供服务。当某一主机联网使用时，该系统就同网中更多的系统和用户交往，这个操作系统的功能要扩充，以适应网络环境的需要。网络环境下的操作系统既要为本机用户提供简便、有效地使用网络资源的手段，又要为网络用户使用本机资源提供服务。为此，网络操作系统除了具备一般操作系统应具有的功能模块（如系统核心、设备管理、存储管理、文件系统等）之外，还要增加一个网

络通信模块。该模块由通信接口、中断处理程序、通信控制程序以及各级网络协议软件组成。

网络操作系统提供的功能包括:允许用户访问网络主机中的各种资源;对这种访问进行控制,仅允许授权用户访问特定的资源;使远程资源的利用同本地资源一样;提供全网络统一的记账办法;联机提供最近的网络说明资料。

网络操作系统以命令(或称原语)形式向用户或上层软件提供服务。这些原语可分为用户通信原语、作业迁移原语、数据迁移原语和控制原语4类。

6. 分布式操作系统

(1) 分布式系统

一组相互连接并能交换信息的计算机形成了一个网络。这些计算机之间可以相互通信,任何一台计算机上的用户可以调用网络上其他计算机的资源。但是,计算机网络并不是一个一体化的系统,它没有标准的、统一的接口。网上各结点的计算机有各自的系统调用命令、数据格式等。若一个计算机上的用户希望使用网上另一台计算机的资源,他必须指明是哪个结点上的哪一台计算机,并以那台计算机上的命令、数据格式来请求,才能实现共享。另外,为完成一个共同计算任务,分布在不同主机上的各合作进程的同步协作也难以自动实现。因此,计算机网络的功能对用户来讲是不透明的。它存在的问题之一是,在网络上的不同类型计算机中,为某一类计算机所编写的程序如何在另一类计算机上运行;存在的另一个问题是,如何在具有不同数据格式、字符编码的计算机之间实现数据共享。另外,还需要解决分布在不同主机上的诸合作进程如何自动实现紧密的合作。

大量的实际应用要求一个完整的、一体化的系统,而且又具有分布处理能力。如在分布事务处理、分布数据处理、办公自动化系统等实际应用中,用户希望以统一的界面、标准的接口去使用系统的各种资源,去实现所需要的各种操作。这就导致了分布式系统的出现。

分布式系统又称为分布式计算机系统或分布式数据处理系统,简称分布式系统。分布式系统是由多个相互连接的处理单元组成的计算机系统。这些处理单元能够在整个系统的控制下合作完成一个共同的任务,最少依赖集中的程序、数据或硬件。这些处理单元可以是物理上相邻的、也可以是在物理上分散的。

构成分布式系统的处理单元就是一个个独立的计算机系统,这些计算机都有自己的局部存储器和外部设备。它们既可独立工作(自治性),亦可合作。在这个系统中,各机器可以并行操作且有多个控制中心,即具有并行处理和分布控制的功能。分布式系统的主要特征是:逻辑上它是单一系统,为用户提供一个透明的用户接口,使用户感觉不到系统是由多台计算机构成的事实。用户需要存取资源时,只要提出需要哪种服务,而不用指明由哪些资源,在哪儿为他服务,用户像使用单机一样地使用分布式系统。这就要求分布式系统具有任务自动划分、任务全局调度、全局资源分配能力。

分布式系统的基础可以是一个计算机网络,也可以是由特殊的互连结构相互连接而成。

关键是在分布式系统中一定含有多个处理单元,能进行并行操作,处理部件之间利用消息通信来进行相互合作。分布式系统不仅仅是一个物理上的松散耦合系统,同时又是一个逻辑上紧密耦合的系统。

目前,分布式应用中有许多采用计算机网络作为硬件结构。分布式系统和计算机网络的区别在于前者具有多机合作和坚强性。多机合作是指自动地实施任务分配和协调。而坚强性表现在,当系统中有一个甚至几个计算机或通路发生故障时,其余部分可自动重构成为一个新的系统,该系统可以工作,甚至可以继续完成其失效部分的部分或全部工作,这叫做优美降级。当故障排除后,系统自动恢复到重构前的状态。这种优美降级和自动恢复就是系统的坚强性。人们研制分布式系统的根本出发点和目的就是追求多机合作和坚强性。正是由于多机合作,系统才取得短的响应时间,高的吞吐量;正是由于优美降级,才获得了高可用性和高可靠性。

(2) 分布式操作系统

分布式系统是一个一体化的系统,在整个系统中有一个全局的操作系统称为分布式操作系统,它负责全系统的资源分配和调度、任务划分、信息传输、控制协调等工作,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的操作和使用系统的资源。至于操作是在哪一台计算机上执行或使用哪个计算机的资源则是系统的事,用户是不用知道的,也就是系统对用户是透明的。

分布式操作系统研究的问题很多,主要包括以下几个方面:分布式操作系统模型与层次结构;分布式资源管理模型、全局资源分配策略和算法;分布式资源存取控制;全局处理机分配及处理机负荷平衡;进程通信机制;数据安全问题;分布式活动的一致性问题。

还有分布式命名、分布式死锁检测、系统容错与故障处理等研究课题,在这里不一一列举。

分布式操作系统研究的内容非常丰富,有许多是当前研究的热点。许多学者及科学工作者正在进行深入研究,并不断取得研究成果。由于计算机应用的迫切需要,分布式操作系统与分布式系统将日益完善和实用化。

3.2 操作系统用户界面

3.2.1 运行一个用户程序的过程

1. 计算机处理应用程序的步骤及作业的概念

任何应用程序要在计算机上运行,都必须通过以下 3 个步骤:

① 用某种语言(如 C 语言)编制一个程序,这个程序被称为源程序。

② 将源程序和初始数据记录在某种输入介质(如磁盘)上。一般在终端设备(包括键盘、显示器)上直接编辑源程序。

③ 按照一定要求来控制计算机工作,并经过加工最后算出结果。

控制计算机工作的最简单的办法是,由操作员通过控制台(或用户在终端设备上)输入一条条命令。例如,用户可先将源程序通过编辑建立在磁盘上,接着发出"编译"命令,操作系统接到这条命令后,将编译程序调入主存并启动它工作。编译程序将记录在磁盘上的源程序进行编译,并产生浮动目标程序模块。然后,用户再发出"连接"命令。操作系统执行该命令,将生成一个完整的、可执行的主存映像程序。最后发出"运行"命令,由操作系统启动主存映像程序运行,从而计算出结果。

从这个简单的例子可以看到,计算机好像一个加工厂,当把原材料(源程序和数据)以及加工要求(先对源程序编译、再连接、最后运行等)交给工厂后,它就生产出成品来。这样一个加工过程称为作业。更确切地说,作业是要求计算机系统按指定步骤对初始数据进行处理,并得到计算结果的加工工作。在多道程序运行环境下,一个作业是一个单位,是一个用户的计算任务区别于其他用户的计算任务的一个单位。从这个观点看,作业是对算题任务进行处理的一个动态过程,但从静态观点看,作业有其对应的程序和数据。若说将某作业装入主存,指的是将该作业的程序和数据装入主存。

2. 作业步及其相互关系

对源程序和数据的加工过程一般可分为若干个步骤。通常把加工工作中的一个步骤称为作业步。对作业的处理一般有这样几个作业步:编辑(修改)、编译、连接、运行。

(1) 作业步

① 编辑(修改)。编辑是建立新文件或对原有文件进行修改的工作步。在单用户操作系统或分时系统中,采用联机工作方式。用户可以调用自己熟悉的编辑程序来完成这一工作。现在流行的编辑程序非常丰富。一般,采用全屏幕编辑程序,如 Turbo C、VC(Visual C)、VB(Visual Basic)等。也有采用行编辑程序的,如 UNIX 系统中的 ED 等。

② 编译。编译是将源程序翻译成浮动目标程序的过程。翻译好的浮动目标程序存放在磁盘上。

③ 连接。连接是将主程序模块和其他所需要的子程序和例行程序连接装配在一起,成为一个可执行的完整的主存映像文件的过程。

④ 运行。将主存映像文件调入主存,并启动之,最后给出计算结果。

(2) 作业步之间的关系

要完成一个应用程序,必须经过上述 4 个作业步。这些作业步是相互关联、顺序地执行的。作业步之间的关系表现为:

① 每个作业步运行的结果产生下一个作业步所需要的文件。

如图 3.2.1 所示,某用户通过编辑作业步,用 C 语言编写了一个名为 user.c 的源程序,

它是第 2 个作业步(编译)的对象;而编译作业步产生的目标模块(user. obj)则是第 3 个作业步的连接对象,经过第 3 个作业步连接装配后形成的主存映像文件(user. exe)是第 4 个作业步运行的处理对象;在第 4 个作业步中,运行该主存映像文件,最终得到运行结果。

② 一个作业步能否正确地执行,依赖于前一个作业步是否成功地完成。

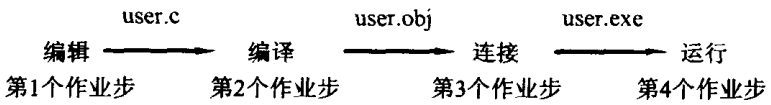


图 3.2.1 作业步之间的关系

若源程序中有错,编译作业步一定不会成功,系统会列出编译中的错误。这时,需返回到编辑作业步,进行修改,修改正确后再进行编译。只有当编译的出错信息为零时才能进行下一个作业步,而只有当连接成功后程序才能投入运行。所以,只有当前一个作业步得出正确的结果后才能进行下一个作业步的工作。

3.2.2 什么是用户界面

用户要想把某一算题任务交给计算机完成,最关心的问题是:系统提供什么手段使用户能方便地描述和解决自己的问题。现代的计算机系统,都是在裸机之上配置了操作系统后,以操作系统虚拟机的面貌呈现在用户面前的。用户通过操作系统来使用计算机,那么,操作系统又以什么样的界面和用户打交道呢?

操作系统的用户界面(或称接口)是操作系统提供给用户与计算机打交道的外部机制。用户能够借助这种机制和系统提供的手段来控制用户所在的系统。

操作系统的用户界面分为两个途径(或称两个类型),一个是操作界面(或称命令界面),用户使用这个操作界面来组织自己的工作流程和控制程序的运行。操作界面有作业控制语言、键盘命令和图形化的操作界面 3 种形式。操作系统的另一个用户界面是操作系统的程序界面(或称功能服务界面)。任何一个用户程序在其运行过程中,都可以使用操作系统提供的功能调用来请求操作系统的服务(如申请主存、使用各种外设、创建线程等)。

任何一个操作系统都具有程序级与操作级两类界面,而操作界面与操作系统类型有密切的关系,比如,批处理系统提供的操作界面为作业控制语言,因为这类操作系统采用的是脱机处理方式;而分时系统、个人计算机提供的操作界面则是键盘命令或者是图形化的界面,因为这类操作系统采用的是联机处理方式。

操作系统的用户界面在近年来发生了巨大的变化。在图形界面(GDI)技术、面向对象技术的推动下,现在个人计算机操作系统提供图形化的用户界面和 API(用户程序编程接口),这是传统的操作界面和系统功能服务界面在现代操作系统中的体现,这样的界面,用

户使用更为直观、方便、有效。

操作系统的用户界面如图 3.2.2 所示。

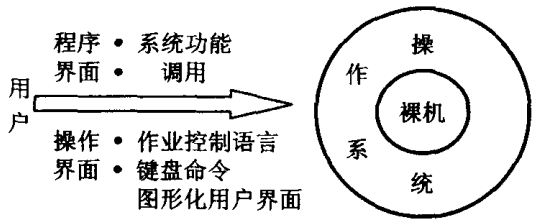


图 3.2.2 操作系统、人、机之间的关系

3.2.3 操作界面

操作界面的形式较大程度上取决于相应操作系统的类型和用户上机方式。具有联机操作方式的分时系统和个人计算机系统提供的是键盘命令或图形化用户界面。在这样的系统中,用户可以与系统"对话",推动程序的处理过程。而具有脱机操作方式的系统(如批处理系统)则提供作业控制语言。在批处理系统中,用户一旦提交了他的作业,就无法对作业运行做更多的控制。因此,用户必须事先给出一系列明确的指令,指出处理的过程,还可能需要对事先无法预测的情况进行周密的思考,指出当某种情况一旦发生时应作出什么样的处理。

现在,操作界面具有键盘命令和图形化的操作界面。在视窗操作系统中,用户可以方便地借助鼠标等标记性设备,选择所需要的图标,采用点取或拖拽等方式完成自己的操作意图。

有的系统既具有交互作用能力,又具有批处理能力,它就应该提供多种形式的操作命令。下面,首先简单地讨论传统的操作界面:键盘命令和作业控制语言,然后讨论现代操作系统中常提供的图形化用户界面。

1. 键盘命令

分时系统或个人计算机系统提供键盘命令。虽然不同的系统所提供的键盘命令有多有少,但其功能基本上是相同的。一般终端与主机通信的过程可以分为注册、通信和注销几个步骤。

(1) 注册

使用分时系统的第一件事是注册。注册有两个目的:一是让系统验证用户有无使用该系统的权限,二是让系统为用户设置必要的环境。

分时系统的功能之一是管理计算机资源,以便若干人共享一台计算机。为此,系统为每个用户维持一个独立的环境。它要记住每一个用户的名字、注册时间,还要记住每个用户已

经用了多少计算机时间,占用了多少文件,正在使用什么型号的终端等。在个人计算机系统中,不存在注册过程,因为实际地访问这个硬件就证实了用户使用这个系统的权力。

在第一次注册之前,系统管理员必须为用户建立一个账号。从用户角度来看,设置一个账号的主要目的是注册名字。注册名是用户与系统交互时需要使用的名字。UNIX 和 Linux 系统正是采用了这种方式。在 UNIX 和 Linux 接通终端之后,用户按 Enter 键,系统会显示"login:"字样,此时系统要用户输入注册名。当用户输入注册名并按 Enter 键后,系统即核对该系统是否记录了这个用户,并在核对正确后显示"password:",即系统要求用户输入口令(口令是为了证实用户的身份而打入的一个保密字),这时可以输入口令并按 Enter 键。一旦输入口令,系统就检验它,如果口令错,系统会要求再输入注册名和口令;否则,系统显示一个提示符,表明系统已经准备好,接收用户的命令了。

(2) 通信

当终端用户注册后,就可以通过丰富的键盘命令控制程序的运行、申请系统资源、从终端输入程序和数据等。属于通信这一步的键盘命令有以下几类:

① 文件管理。这类命令用来控制终端用户的文件,例如,删去某个文件,将某个文件由显示器(或打印机)输出,改变文件的名字、使用权限等。

② 编辑修改。这类命令用来编辑和修改终端用户的文件。例如删去几行、插入几行、修改几行等。这类命令是重要的,因为当终端用户发现由于某种原因需要修改他的文件时,他可以直接从终端打入命令来修改,而不需要脱机修改,然后再重新输入。

③ 编译、连接装配和运行。这类命令用来调出编译或连接装配程序进行编译或装配工作,以及将生成的主存映像文件装入主存启动运行。

④ 输入数据。终端用户打入输入命令,要求系统接受从终端输入的一批数据。这一批数据一般以文件形式放到后援存储器上。

⑤ 操作方式转换。这类命令主要用来转换作业的控制方式,例如,从联机工作方式转为脱机工作方式。

⑥ 申请资源。这类命令主要用来让终端用户申请使用系统的资源。例如,申请使用某类外部设备若干台等。

(3) 注销

当用户使用完了或暂时不使用系统时,应输入注销命令。注销就是通知系统打算退出系统。比如,当要退出 UNIX 和 Linux 系统时,应在 shell 的命令提示符后输入注销命令。注销命令随系统而异,如 Logout 或 control d 等。当用户注销后,系统将再次显示"login:",即准备接受新用户。

2. 作业控制语言

在脱机方式下,系统提供作业控制语言(JCL)。使用作业控制语言可以书写操作说明书。

采用脱机方式时,用户上机前必须准备好作业申请表、操作说明书以及程序和数据。其中,作业申请表是用户向系统提出的执行作业的请求,其内容应包含:作业名、需用 CPU 时间,最迟完成时间、资源请求(包括主存容量、外部设备台数、后援存储器容量、输出量(打印行数))等,指出使用何种语言编译程序。表明用户对作业控制意图的操作说明书则是由一条条对作业处理的命令组成的,如编辑命令、编译命令、连接命令、运行命令等。还有一些干预命令,它说明了在作业运行过程中发生意外事件时的处理方式。

操作系统根据作业申请表来分配作业所需的资源并注册该作业,通过作业说明书对作业实施运行控制。一般在批处理系统中提供 JCL 语言。

3.2.4 图形化的用户界面

随着计算机应用的普及,人们逐渐感到键盘命令的交互方式也不太方便了,因为这种命令是不直观的,是比较难懂的一串串字符命令,还带有各种参数和规定的格式。另外,不同的操作系统所提供的命令语言的词法、语法、语义和表达风格也是不一样的。当一个对 MS - DOS的键盘命令使用得十分熟悉的程序员要改用 UNIX 时,还得重新熟悉 UNIX 的命令。这种命令语言还存在的一个问题是,它是用英文表达的语言,对于非英文语种国家的计算机应用的推广会形成一种障碍。但计算机应用发展的势头太快,它迅速地进入了各行各业、千家万户,它面对的用户是不同阶层、不同文化程度的人们。如何使人机交互方式进一步变革,使人机对话的界面更为方便 、友好、易学,这是一个十分重要的问题。在这种需求下出现了菜单驱动方式、图符驱动方式直至视窗操作环境。

(1) 菜单驱动方式

菜单(Menu)驱动方式是面向屏幕的交互方式,它将键盘命令以屏幕方式来体现。系统将所有有关的命令和系统能完成的操作,用类似餐馆的菜单的方式分类分窗口地在屏幕上列出。用户根据菜单提示,像点菜一样选择某个命令或某种操作,以控制系统去完成指定的工作。菜单系统的类型有多种,如下拉式菜单、上推式菜单和随机弹出式菜单。这些菜单都基于一种窗口模式。每一级菜单都是一个小小的窗口,在菜单中显示的是系统命令和控制功能。

(2) 图符驱动方式

图符驱动方式也是一种面向屏幕的图形菜单选择方式。图符(Icon)也称图标,是一个小小的图形符号,它代表操作系统中的命令、系统服务、操作功能、各种资源等。例如,用小矩形图符代表文件,用小剪刀图符代表剪切操作。所谓图形化的命令驱动方式就是当需要启动某个系统命令或操作功能,或请求某个系统资源时,可以选择代表它的图符,并借助鼠标一类的标记输入设备(也可以采用键盘),通过鼠标的点击和拖拽功能,完成命令或操作的选择。

(3) 图形化用户界面

图形化用户界面是良好的用户交互界面,它将菜单驱动、图符驱动、面向对象技术等集成在一起,形成一个图文并茂的视窗操作环境。微软公司的 Windows 系统就是这种图形化用户界面的代表。

Windows 系统为所有的用户和应用系统提供一种统一的图形用户界面。在系统中,所有程序都以统一的窗口形式出现,提供统一的菜单格式。Windows 系统管理的所有系统资源,例如,文件、目录、打印机、磁盘、网上邻居、进程、各种系统命令和操作功能都变成了生动的图形图像。窗口中使用的滚动条、按钮、编辑框、对话框等各种操作对像也都采用统一的图形显示和统一的操作方法。在这种图形化用户界面的视窗环境中,用户面对的不再是单一的命令输入方式,而是各种图形表示的一个个对象。用户可以通过鼠标(或键盘)选择需要的图符,采用点击方式操纵这些图形对象,达到控制系统、运行某一个程序、执行某一个操作的目的。用户将通过这种统一的用户界面使用各种 Windows 应用程序,从而增强控制系统的能力。

图形化的用户界面实际上是操作系统提供的操作命令界面的革新。操作系统提供的另一个接口是针对程序设计者而提供的系统功能服务。Windows 为系统设计者提供 API(应用程序编程接口)函数和系统定义的消息形式。API 函数与传统操作系统提供的系统调用的主要不同点是函数库和动态链链技术的支持。

3.2.5 系统调用

1. 什么是系统功能调用

操作系统如何在程序级提供服务功能呢? 这必须由系统设计者事先编制好能实现用户功能要求的各种例行程序,作为操作系统核心程序的一部分。用户使用操作系统的服务例程时,不能像调用一般的子程序那样随便,因为这些能实现各种功能的例行子程序是操作系统的程序部分,它运行时,机器处于管理程序状态,而用户程序运行时,机器处于目态。所以,用户程序对这些例行子程序的调用应以一种特殊的调用方式——访管方式来实现。而要实现访管必须有硬件的支持及有关的操作系统功能模块。

(1) 自愿进管指令

用户所需要的功能,有些是比较复杂的,硬件不能直接提供,只能通过软件的程序来实现。而有些功能,硬件可以提供,如启动外设工作(机器有用于输入/输出的硬指令)。但配置了操作系统后,对系统资源的分配、控制不能由用户干预,而必须由操作系统统一管理。所以,对于这样一类功能,也需有操作系统来处理。为了实现对操作系统服务功能的调用,现代计算机系统一般提供自愿进管指令,其指令形式为:

SVC n

其中,SVC 表示机器自愿进管指令的操作码记忆符,n 为地址码。SVC 是 Supervisor Call(访问管理程序)的缩写,所以 SVC 指令又称访管指令,它是一条硬指令。

(2) 访管中断

当处理机执行自愿进管指令时就会发生中断,该中断类型称为访管中断(或自愿进管中断),它表示正在运行的程序对操作系统的某种需求。

借助中断,使机器状态由目态转为管态。为了使控制能转到用户当前所需要的那个例行子程序,指令需要提供一个地址码。这个地址码表示系统调用的功能号,它是操作系统提供的众多的例行子程序的编号。在访管指令中填入相应的号码,就能控制转到特定的例行子程序去执行,以完成用户当前所需要的服务。这样一个带有一定功能号的访管指令定义了一个系统调用。

(3) 系统调用

系统调用是用户在程序一级请求操作系统服务的一种手段,它不是一条简单的硬指令,而是带有一定功能号的"访管指令"。它的功能并非由硬件直接提供,而是由操作系统中的一段程序完成的,即由软件方法实现的。

用户可以用带有不同功能号的"访管指令"来实现各种不同的功能。可以这样说,系统调用是利用"访管指令"定义的指令。操作系统服务例程与一般子程序的区别在于,操作系统服务例程所实现的功能都是与计算机系统本身有关的,对操作系统服务例程的调用是通过一条"访管指令"来实现的。

系统调用是操作系统提供的程序界面,在不同的程序设计语言中系统调用形式可以不同,有显式调用和隐式调用之分。在汇编语言中是直接使用系统调用对操作系统提出各种要求,因为在这种情况下,系统调用具有汇编指令的形式。而在高级语言中一般是隐式的调用,如 C 语言中的 printf 语言句,就是在 C 程序语言中请求操作系统输出字符串的系统调用。在高级语言中的隐式调用,经编译后会转换成汇编级的直接调用形式。

不同的计算机系统提供的访管指令形式不同,由它们定义的汇编一级的系统调用的形式也就不同。如 IBM PC 机提供的软中断指令为"int n",其中 n 为中断类型号,由它定义了不同的软件中断。软件中断可用作"管理程序调用",也就是请求操作系统服务。其中 21H 中断类型中又包含了 DOS 丰富的系统功能调用。又如,IBM 360/370 机器中访管指令的形式为"svc n",PDP 系列机的访管指令为自陷指令"trap n"等。

2. 系统功能调用的分类

系统调用是操作系统提供给程序员的接口,程序设计人员在编写程序时,可以利用系统调用来请求操作系统服务。操作系统提供的系统调用是十分丰富的,按其功能可分为以下几类:

① 设备管理。设备管理系统调用被用来请求和释放设备,以及启动设备操作等。

② 文件管理。文件管理系统调用包括创建、删除文件,读、写文件以及移动文件指针等。

③ 进程控制。进程是程序在处理机上的一次执行过程。进程控制功能负责进程状态

的变化。用于进程控制的系统调用有:进程创建、进程撤销、进程等待、进程唤醒等。

④ 进程通信。进程通信的系统调用包含进程间传递消息或信号的系统调用。

⑤ 存储管理。存储管理系统调用包含主存块的申请、释放,获取作业占用主存块的首址、大小等。

不同的系统为用户提供的系统调用的数量或形式是不同的。一般的系统为用户提供几十到几百条系统调用。

3.3 进程及进程管理

3.3.1 为什么要引入进程的概念

为了提高计算机系统的效率和增强计算机系统各部件的并行操作能力,计算机系统中必须同时存放多个应用程序,操作系统应能处理多个正在执行的应用程序。传统的程序设计方法涉及的概念是“程序”,处理方法是程序的顺序执行。但程序的概念不能体现“并发”这个动态的含义,程序的顺序执行也决不具备并发处理的能力。为了描述操作系统的并发性,人们引入了一个新的概念——进程。为了讲清进程的概念,必须了解为什么要引入这个概念。下面就从程序的顺序执行、程序的并发执行讲起。

1. 程序的顺序执行及特点

人们在和计算机打交道时,总是使用“程序”这一概念。程序是指令的有序集合,是一个静态的概念。但一个程序必须经过处理才能得到最终的结果。一个程序的执行过程就是一次计算,这是一个动态的过程。若一个计算过程由若干操作组成,而这些操作必须按照某种先后次序来执行,则这类计算过程就是程序的顺序执行过程。

顺序程序的操作是一个接一个地以有限的速度进行的,每次执行一个操作,只有在前一个操作完成后,才能进行其后继的操作。由此产生顺序程序的如下特点:

① 顺序性。当顺序程序在处理机上执行时,处理机的操作是严格按照程序所规定的顺序执行的,即每个操作必须在下一个操作开始执行之前结束。

② 封闭性。程序一旦开始执行,其计算结果不受外界因素的影响。因为是一道程序独占系统各种资源,故这些资源的状态,当初始条件给定以后,其后的状态只能由程序本身确定,亦即只有本程序的操作才能改变它。

③ 可再现性。程序执行的结果与它的执行速度无关(即与时间无关),而只与初始条件有关。只要给定相同的输入条件,程序重复执行一定会得到相同的结果。

2. 程序的并发执行及特点

(1) 什么是程序的并发执行

为增强计算机系统的处理能力和提高机器的利用率，在现代计算机中广泛采用同时性操作技术，即并行操作技术。之所以并行操作是可能实现的，是因为计算机系统有多个物理部件；同时有许多计算可以在不同的部件上同时进行。值得一提的是，现在常将在单机系统中的多个程序的并行执行称为并发执行。

例如：在 Windows 系统中，可以同时打开发送邮件、文件编辑、听音乐等多个窗口，即有多个应用程序可以同时活动。

所谓程序的并发执行，是指若干个程序同时在系统中运行，这些程序的执行在时间上是重叠的，一个程序的执行尚未结束，另一个程序的执行已经开始，即使这种重叠是很小的一部分，也称这几个程序段是并发执行的。图 3.3.1 所示的就是并发执行的程序。

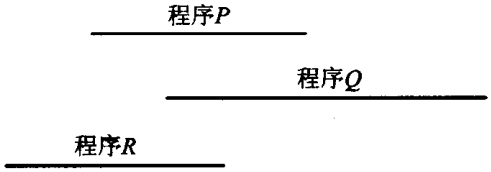


图 3.3.1 3 个并发程序段

可以用如下语句表示程序的并发执行：

COBEGIN

$S_1;S_2;\cdots;S_n$

COEND

该语句记号表示语句 $S_1,S_2,\cdots,S_n$ 可以并发执行。这是由 Dijkstra 首先提出来的。

为了确定这一语句的效果，还应该考虑该并发语句的前、后两个语句 S_0 及 S_{n+1}。即：

S_0;

COBEGIN

$S_1;S_2;\cdots;S_n$

CONED;

S_{n+1}

为了能更形象地理解并发执行，可以编写这样一个演示程序，它由一个主程序和两个子程序构成。

① 子程序 A：在屏幕第 10 行上，从左至右显示一条向前推进的直线 a。

② 子系统 B：在屏幕第 20 行上，从左至右显示一条向前推进的直线 b。

③ 主程序：按指定的执行参数交替地调用子程序 A 和子程序 B。

当每次执行步数由多变少时，交替越来越频繁，在屏幕上看到的两条直线交替地向前推进，变成越来越接近同时移动。这个例子可以很好地理解微观上的交替执行和宏观上的同

时执行。

(2) 并发程序的特点

程序并发执行虽然卓有成效地增加了系统的处理能力和机器的利用率,但它也带来了一些新问题,产生了与顺序程序不同的特征。

① 失去程序的封闭性

如果并发程序间有公共变量,一个程序的执行可能改变另一个程序变量的内容,这时,程序的执行结果就可能依赖于各程序执行的相对速度,也就是失去了程序的封闭性特点。

现以两个并发的循环程序共用一个公共变量 N 来说明这个问题。设程序 A 每执行一次都要做 N:=N+1 操作,程序 B 每隔一定时间打印出 N 值,并将它重新置为零。这两个并发程序由图 3.3.2 描述。

由于程序 A 和程序 B 的执行都以各自独立的速度向前推进,故程序 A 的 N:=N+1 操作,既可在程序 B 的 PRINT(N)和 N:=0 操作之前,也可在其后或中间(即插在 PRINT(N)和 N:=0 操作之间)。设两个程序在开始某个循环之前,N 的值为 n_0,执行完一个循环后,对应于这 3 种情况,打印机打印出来的值分别为 n_0+1,n_0,n_0;执行后 N 的最终赋值为 0,1,0。之所以会出现错误,是因为它们公用了一个公共变量 N,而又没有采取恰当的措施。这个例子说明并发程序的执行结果与执行速度有关,也就是说,并发程序已丧失了顺序程序所具有的封闭性和可再现性的特点。

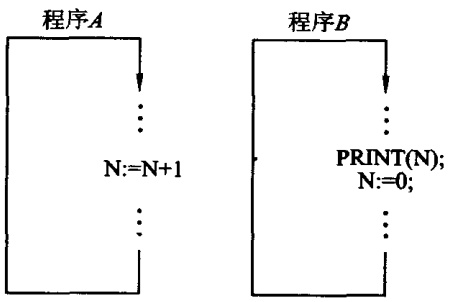


图 3.3.2 两个程序并发执行破坏封闭性的实例

② 程序与计算不再一一对应

程序和计算是两个不同的概念,前者是指令的有序集合,是静态的概念。而计算是指令序列在处理机上的执行过程,或处理机按照程序的规定执行操作的过程,是动态的概念。程序在顺序执行时,程序与计算有着一一对应的关系,但在并发执行时,这种关系就不再存在了。当多个计算任务共享某个程序时,它们都可以调用这个程序,且调用一次就是执行一次计算,因而这个程序可执行多次,即这个共享的程序对应多个“计算”。例如,在多道程序运行环境下,两个用户作业都需要调用 C 编译程序。为了减少编译程序的副本,他们共享一

个 C 编译程序(当然每个用户各带自己的数据区)。这样,一个编译程序能同时为两个作业服务,每个作业调用一次就是执行一次,即这个编译程序对应两个“计算”。

③ 程序并发执行的相互制约

当并发执行的各程序之间需要协同操作来完成一个共同的任务时,它们之间具有直接的相互制约关系,即存在着一定的逻辑联系。

3.3.2 进程的定义

对于并发执行的程序来说,它有时处于执行状态,但由于并发程序之间的相互制约关系,有时它需要等待某种共享资源,有时又可能要等待某些信息而暂时运行不下去了,只得处于暂停状态,而当使之暂停的因素消失后,程序又可以恢复执行,所以,并发程序执行时就是这样停停走走地向前推进的。换言之,由于程序并发执行时的直接或间接的相互制约关系,将导致并发程序具有“执行—暂停—执行”的活动规律,即与外界发生了密切的联系,从而失去了封闭性。在这种情况下,如果仍然使用程序这个概念,只能对它进行静止的、孤立的研究,不能深刻地反映它们活动的规律和状态变化。因此,人们引入了新的概念——进程,以便从变化的角度,动态地分析研究并发程序的活动。

进程的概念是 60 年代初期,首先由麻省理工学院的 MULTICS 系统和 IBM 公司的 TSS/360 系统引入的。其后,有许多人对进程下过各式各样的定义。下面仅列举几种比较能反映进程实质的定义。

- 进程是这样的计算部分,它是可以和其他计算并行的一个计算。
- 进程(有时称为任务)是一个程序与其数据一道通过处理机执行所发生的活动。
- 任务(或称进程)是由一个程序以及与它相关的状态信息(包括寄存器内容、存储区域和链接表)所组成的。
- 所谓进程,就是一个程序在给定活动空间和初始环境下,在一个处理机上的执行过程。

进程也可以这样定义:所谓进程,就是一个程序在给定活动空间和初始环境下,在一个处理机上的执行过程。

上述这些对进程的解释从本质上讲是相同的,但各有侧重,这说明进程这一概念至今尚未形成公认的严格的定义。但是,进程已广泛而成功地被用于许多系统,成为构造操作系统不可缺少的强有力的工具。

进程和程序是既有联系又有区别的两个概念,它们的区别和关系如下:

① 程序是指令的有序集合,其本身没有任何运行的含义,它是一个静态的概念。而进程是程序在处理机上的一次执行过程,是一个动态的概念。程序可以作为一种软件资料长期保存,而进程则是有一定生命期的,它能够动态地产生和消亡。即进程可由“创建”而产生,由调度而执行,因得不到资源而暂停,以致最后由“撤销”而消亡。

② 进程是一个能独立运行的单位，能与其他进程并行地活动。

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 同一程序同时运行于若干不同的数据集合上，它将属于若干个不同的进程。或者说，若干不同的进程可以包含相同的程序。这句话的意思是：用同一程序对不同的数据先后或同时加以处理，就对应于多个进程。例如，一个 C 编译程序可以同时为多个用户的 C 程序进行翻译工作，这就产生了多个编译进程。

进程是系统中独立活动的单位，系统中的多个进程是如何运行的呢？下面以例说明。

(1) 例 1

设有 3 个排序程序，程序 A 采用冒泡排序算法，在屏幕的左 1/3 处开设一个窗口显示其排序过程；程序 B 采用堆排序算法，在屏幕的中 1/3 处开设一个窗口显示其排序过程；程序 C 采用快速排序算法，在屏幕的右 1/3 处开设一个窗口显示其排序过程。

① 在不支持进程运行环境的操作系统下，依次运行程序 A、程序 B、程序 C，在屏幕上可以看到屏幕左 1/3 窗口显示冒泡排序过程，接着在屏幕中 1/3 处显示堆排序过程，最后在屏幕右 1/3 处显示快速排序过程。

② 在支撑进程运行环境的操作系统下建立进程 A、B、C，它们分别对应的程序是程序 A、B、C。若系统按时间片轮转的调度策略调度这 3 个进程，则这时在屏幕上开出 3 个窗口，同时显示按时间片轮转的 3 个排序过程。实际上这 3 个程序在轮流地占用 CPU 时间，由于 CPU 是高速运行，使用户看到的好像是这 3 个程序在同时执行。

(2) 例 2

设有 2 个程序，程序 C 是打印工资报表的程序，程序 D 是计算 1 000 以内所有素数并显示最后结果的程序。

① 在不支持进程运行环境的操作系统下依次执行程序 C、程序 D，可以看到，先是打印机不停地打印工资报表，打完后，接着运行程序 C，不停地计算，最后显示所计算的结果。

② 在支持进程运行环境的操作系统下创建进程 C 和进程 D。由于进程 C 是 I/O 量较大的进程，而进程 D 是计算量较大的进程，故在系统进程调度的控制下，两个进程并发执行。可以看到打印机不断打印工资报表，而处理机不停地计算，最后屏幕显示计算的结果。同时用户可以看到，当具有这样不同特征的进程同存于系统中时，I/O 设备和处理机并行操作使系统资源的效率得到充分的发挥。

3.3.3 进程的状态及变迁

1. 进程的基本状态

进程有着“执行—暂停—执行”的活动规律。一般说来，一个进程并不是自始至终一口气运行到底的，它与并发执行的其他进程的执行是相互制约的。进程有时处于运行状态，有时又由于某种原因而暂停运行，处于等待状态，当使它暂停的原因消失后，它又可准备运行

了。所以,在一个进程的活动期间至少具备3种基本状态,它们是:就绪状态、运行状态、等待状态(又称阻塞状态或挂起状态)。

(1) 就绪状态(ready)

进程已获得除中央处理机之外的所有资源,它们已经准备就绪,一旦得到CPU,就立即可以运行,这些进程所处的状态为就绪状态。

(2) 运行状态(running)

进程得到中央处理机的控制权,它的程序正在其上运行,该进程所处的状态为运行状态。

(3) 等待状态(wait)

若一进程正在等待着某一事件发生(如等待输入输出操作的完成而暂时停止执行),这时,即使给CPU时间,它也无法执行,则称该进程处于等待状态。

在一个实际的系统中,在进程活动期间至少要区分出就绪、运行、等待这3种状态来。原因是进程并发执行时存在相互制约的关系,有的进程可能因某种原因而处于等待状态,当使其等待的原因消失后,该进程又处于准备就绪状态。如果系统能为每个进程提供一台处理机的话,则系统中所有就绪进程都可以同时执行,但实际上处理机的数目总是少于进程数,因此,往往只有少数几个进程(在单处理机系统中,则只有一个进程)可真正获得处理机。有的系统较为复杂,还可设置更多的进程状态,例如等待状态可按等待原因不同进一步细分。

2. 进程的基本状态

进程的状态将随着自身的推进和外界条件的变化而发生变化。对于一个系统,可以用一张进程状态变迁图来说明系统中每个进程可能具备的状态以及这些状态之间变迁的可能原因。在进程状态变迁图中,以结点表示进程的状态,以箭头表示状态的变化。一个最简单的状态变迁图如图3.3.3所示。从图中可以看出状态之间的演变以及它们相互转换的典型理由。由运行状态转变为就绪状态的可能原因是时间片到,这种情况只有在分时系统才可能发生,在其他类型的操作系统中一般不具备这种状态变迁。

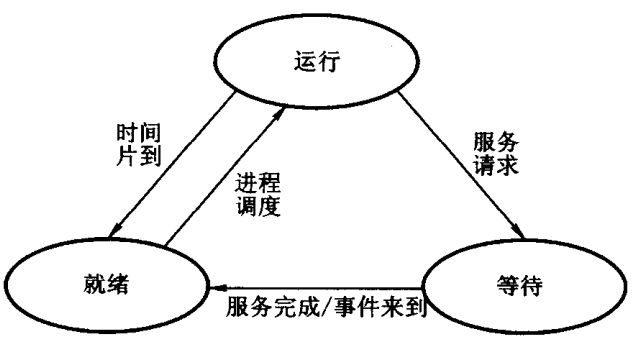


图3.3.3 进程状态变迁图

值得注意的是,处于运行状态的进程因请求某种服务而变为等待状态,但当该请求完成后,等待状态的进程并不能恢复到运行状态,它通常是先转变为就绪状态,再重新由调度程序来调度。其原因用户可自行考虑。

上面介绍了进程的3个基本状态及状态的转换。那么,进程是如何产生和消亡的呢?进程是程序的一次执行过程,它是一个活动。当用户或系统需要一个活动时,可以通过创建进程的方法来产生一个进程。进程被创建后进入就绪状态。而当一个进程的任务完成时,可以通过撤销进程的方法使进程消亡。操作系统提供进程控制功能满足用户对进程活动控制的需要。

3.3.4 进程的描述

为了适应并发程序设计的需要而引入了进程的概念。进程是程序的一次执行过程,那么,如何描述一个进程活动?进程是由哪几个部分组成的呢?下面讨论这两个问题。

1. 进程控制块

当程序并发执行时,产生了动态特征,并由于并发程序之间的相互制约关系而造成了比较复杂的外界环境。为了描述一个进程和其他进程以及系统资源的关系,为了刻画一个进程在各个不同时期所处的状态,人们采用了一个与进程相联系的数据块,称为进程控制块(Process Control Block,PCB),或称为进程描述器(Process Descriptor)。当系统创建一个进程时,必须为它设置一个PCB,然后根据PCB的信息对进程实施控制管理。进程任务完成时,系统收回它的PCB,进程也随之消亡。

为了对进程作充分的描述,PCB通常应具有的信息如表3.3.1所示。表中各项内容说明如下:

表3.3.1 PCB结构

进程标识符
进程当前状态
当前队列指针
总链队列指针
程序开始地址
进程优先级
CPU现场保护区
通信信息
进程家族
占有资源清单

① 进程标识符。每个进程都必须有唯一的标识符,可以用字符或编号表示。在创建一个进程时,由创建者给出进程的标识符。当进程创建成功后,它有一个内部标识符 PID。

② 进程当前状态。标识本进程目前所处状态(如运行、就绪、等待)。只有当进程处于就绪状态时,才有可能获得处理机,当某个进程处于等待状态时,要在 PCB 中说明等待的原因。

支持多任务运行的系统中,存在大量的进程,操作系统如何对这些进程实施有效的管理呢?操作系统采用的办法是组织队列,将具有相同状态的进程链在一起,组成各种队列。比如,将所有处于就绪状态的进程链在一起,称为就绪队列。把所有因等待某事件而处于等待状态的进程链在一起,就组成了各种等待队列。而处于运行状态的进程在单处理机系统中只有一个,所以,使用一个运行指针(running)来指示当前处于运行状态的进程。

③ 当前队列指针。该项登记了处于同一状态下的下一个进程的 PCB 地址,以此将处于同一状态的所有进程勾链起来。每个队列有一个队列头,其内容为队列第一个元素的地址。图 3.3.4 描述了一个就绪队列和一个等待打印机队列的结构。

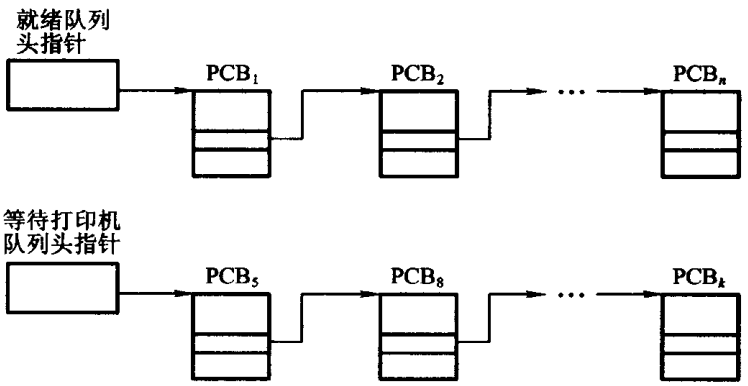


图 3.3.4 就绪队列和等待队列

④ 总链对列指针。将系统中已建立的所有进程链接在一起形成的队列,称为总链队列,进程 PCB 中的该项内容是总链中的下一个 PCB 地址。

系统中存在着大量的进程,它们依各自的状态分别处于相应的队列中,这便于对进程实施调度控制。当进行某些管理功能,如执行创建新进程的功能时,就感到系统具有所有进程的总链将是十分方便的。因为进程的标识符必须是唯一的,由创建者给出的被创建进程的名字是否会重名,必须先检查系统已有的进程名,但若分别在各个队列去查询将是十分麻烦的,所以,有的系统会提供一个进程总链结构。

⑤ 程序开始地址。该进程对应的程序将以此地址开始执行。

⑥ 进程优先级。进程优先级反映了进程要求 CPU 的紧迫程度,它将以其优先级的高低去争夺 CPU 的权利。进程优先级通常由用户预先提出或由系统指定。

⑦ CPU 现场保护区。当进程由于某种原因释放处理机时,CPU 现场信息被保存在 PCB 的该区域中,以便在该进程重新获得处理机后能继续执行。通常被保护的信息有工作寄存器、指令计数器以及程序状态字等。

⑧ 通信信息。是指每个进程在运行过程中与其他进程进行通信时所记录的有关信息。比如,可以包含正等待着本进程接收的消息个数、第一个消息的开始地址等。

⑨ 进程家族。有的系统允许一个进程创建自己的子进程,这样,会组成一个进程家族。在 PCB 中必须指明本进程与家族的联系,如它的子进程和父进程的标识符。

⑩ 占有资源清单。不同的操作系统所使用的 PCB 结构是不同的。对于简单系统,PCB 结构较小。而在一些较复杂的系统中,PCB 所含的内容就比较多,比如,还可能有 I/O、文件传输等控制信息。但是,一般 PCB 应包含的最基本内容如表 3.3.1 所示。

2. 进程的组成

从结构上讲,每个进程都由程序、数据和一个进程控制块 PCB 组成(如图 3.3.5 所示)。进程的程序部分描述了进程所要完成的功能。数据集合是程序在执行时所需要的数据和工作区。进程控制块,包含了进程的描述信息和控制信息,是进程的动态特征的集中反映。系统根据 PCB 而感知某一进程的存在,所以 PCB 是进程存在的唯一标志,或者说 PCB 是进程的唯一实体。一个程序可以存放在磁盘上,可以调入主存执行,但若一个系统不支持进程活动,它就不可能为一个程序的执行建立相应的 PCB 结构,也就不存在进程。只有在多任务系统中,才可能建立 PCB 结构,该系统才具有进程活动。

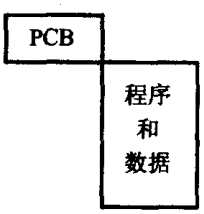


图 3.3.5 进程的组成

3.3.5 进程控制

1. 进程控制的概念

进程的状态是不断变化的,进程控制的职责就是对进程状态的变化实施有效的管理。它是处理机管理的一部分,通过建立进程控制机构,实现进程的创建、撤销和实施进程间同步、通信等功能。这一控制机构属于操作系统内核的一部分。内核是由一些具有特定功能的程序段组成的,它是通过执行各种原语操作来实现各种控制和管理功能的。原语是一种特殊的系统调用,它可以完成一个特定的功能,一般为外层软件所调用,其特点是原语执行时不可中断,在操作系统中原语作为一个基本单位出现。

用于进程控制的原语有:创建原语、撤消原语、阻塞原语、唤醒原语等。

对于应用程序而言,在多进程环境中,它包括一个主进程和可能出现的同时活动的子进程。为了完成用户程序的任务,用户必须能控制这些进程的活动,在操作系统方面应能提供

控制功能,而用户则通过服务请求方式获得这些功能。

2. 进程创建与进程撤销

(1) 进程创建

进程创建的功能是创建一个指定标识符的进程。主要任务是形成该进程的进程控制块PCB。所以,调用者必须提供形成PCB的有关参数,以便在创建时填入。这些参数是:进程标识符、进程优先级、本进程开始地址,其他资源从父进程那里继承。当创建原语执行成功后,可以得到新创建的进程的内部标识符PID。

进程创建原语的执行可以使进程的状态发生从无到有的变化。

(2) 进程撤销

一个进程由进程创建原语而创建,当它完成了任务之后就希望终止自己。这时,应使用进程撤销原语而撤销自己。进程撤销原语的功能是将当前运行的进程(因为是自我撤销)的PCB结构归还给系统,所占用的资源归还给父进程,从总链队列中摘除它,然后转入进程调度程序。因为调用者自己被撤销,所以应由进程调度程序再选一个进程去运行。

进程撤销原语的执行可以使进程的状态发生从有到消亡的变化。

3. 进程阻塞与进程唤醒

有了创建原语和撤销原语,虽然进程可以从无到有、从存在到消亡而变化,但还不能完成进程各种状态的转换。例如,由"运行"转换到"阻塞",由"阻塞"转换为"就绪",需要通过进程之间的同步或通信机构来实现,也可直接使用"阻塞原语"和"唤醒原语"来实现。

(1) 进程阻塞

当一个进程所期待的某一事件尚未出现时,该进程调用阻塞原语将自己阻塞,转换为等待状态。阻塞原语的功能是:将调用阻塞原语的进程的CPU现场送至该进程的现场保护区,置该进程的状态为"等待",并插入到相应的等待队列中,然后转入进程调度程序,另选一个进程投入运行。

进程阻塞原语的执行可以使进程的状态发生从运行到等待的变化。

(2) 进程唤醒

进程由运行转变为等待状态是由于进程必须等待某一事件的发生,所以处于等待状态的进程绝对不可能叫醒自己。比如,某进程正在等待输入/输出操作完成或等待其他进程发消息给它,只有当该进程所期待的事件出现时,才由"发现者"进程用唤醒原语唤醒它。一般说来,发现者进程和被唤醒进程是合作的并发进程。

当进程等待的事件发生时,唤醒原语开始执行其功能。唤醒原语的功能是:将等待该事件的进程从等待队列中摘下,置为"就绪"状态,最后,转进程调度。

进程唤醒原语的执行可以使进程的状态发生从等待到就绪的变化。

3.3.6 进程的同步与互斥

系统中多进程以各自独立的速度向前推进。但这些进程同存于一个系统中，由于对资源的共享和并发进程之间的合作而形成一种相互制约关系。这种制约关系又分为间接和直接两种不同形式。间接的相互制约关系是由资源共享引起的，由系统的资源管理程序协调处理。而直接相互制约关系往往在一组合作进程之间发生，它们共同完成一个任务，这时也需同步协调，而这种协调是通过进程通信来实现的。即具有合作关系的一组进程之间需要交换信息，以便达到协调步伐的目的，也就是需要同步。所谓进程同步就是对于进程操作的时间顺序所加的某种限制。例如，“操作 A 应在操作 B 之前执行”，“操作 C 必须在操作 A 和 B 都完成之后才能执行”等。这是同步概念的广义定义。在同步规则中有一个较特殊的规则是，“多个操作决不能在同一时刻执行”，如“操作 A 和操作 B 不能在同一时刻执行”。这种同步规则称为互斥。

1. 互斥的概念

本节主要讨论互斥问题。下面先通过几个例子（如两个进程共享硬件资源、共享公用变量）来说明临界资源的概念，进而建立互斥和临界区的概念。

（1）临界资源

现代操作系统中，可同时有多个进程并发执行，它们共享着各种资源。然而，系统中有些资源具有这样的特点：一次只能为一个进程使用。在操作系统中，把一次仅允许一个进程使用的资源称为临界资源。许多物理设备，如输入机、打印机、磁带机等，都具有这种性质。除了物理设备外，还有一些软件资源，如变量、数据、表格、队列等，也都具有这一特点。它们虽可被若干进程所共享，但一次只能为一个进程所利用。下面举例说明。

① 多进程共享打印机

假定进程 A，B 共享一台打印机，若让它们任意使用，那么可能发生的情况是，两个进程的输出结果将交织在一起，很难区分。解决这一问题的办法是，进程 A 要使用打印机时应先提出申请，一旦系统把资源分配给它，就一直为它所独占。这时，即使进程 B 要使用打印机，也必须等待，直到 A 进程用完并释放后，系统才把打印机分配给进程 B 使用。

② 多进程共享公共变量

并发进程对公共变量进行访问和修改时，必须作某种限制，否则进程处理所得的结果可能发生错误。例如，假定在一个机票预订系统中，某一机座的订购情况由一个确定的主存单元的内容表示，若干个进程共享这一主存单元，即共享一个变量，设该公共变量为 m。如果一次有两个（或多个）进程同时存取该变量，那么就有可能使两家（或多家）旅行社同时订到该机座。下面的动作序列会导致这种令人遗憾的情况。

进程A：旅行社 A 的订票业务	进程B：旅行社B 的订票业务
旅行社 A 查到该机有座位；	旅行社 B 查到该机有座位；

与其顾客商量；	与其顾客商量；
⋮	⋮
旅行社 A 预订该机座；	旅行社 B 预订该机座；
⋮	⋮

在上述的例子中，打印机和公共变量 m 都具有一次只能让一个进程使用的特点，它们都属于临界资源。

(2) 临界区

在每个进程中，访问临界资源(如对公共变量进行审查与修改)的程序段称为相对于该公共变量的临界区或临界段。

例如，上例中两家旅行社都执行订购飞机票的业务，它们共用一个代表某飞机航班机座的公共变量 m。旅行社 A 查询该机是否有空座位(访问公共变量)，如有，则预订该机座(修改公共变量)，这段程序是旅行社 A 访问公共变量的程序段落，称为进程 A 关于公共变量 m 的临界区。同样，在旅行社 B 的订票业务中，也有一个访问公共变量 m 的程序段落，称为进程 B 关于公共变量 m 的临界区。进程 A 和进程 B 并发执行时，最多只能有一个可以进入临界区，否则就会造成错误。

值得注意的一点是，临界区是对某一资源而言的，对于不同资源的临界区，它们之间是不相交的，所以不必互斥地执行。而相对于同一公共变量的若干个临界区，则每次只能由一个进程访问。

(3) 互斥

在操作系统中，当某一进程正在访问某一存储区域时，就不允许其他进程来读出或者修改存储区的内容，否则就会发生后果无法估计的错误。进程之间的这种相互制约关系称为互斥。

当两个进程公用一个变量时，它们必须顺序地使用，一个进程对公共变量操作完毕后，另一个进程才能去访问和修改这一变量。到底哪一方首先读写，要根据问题的性质和设计人员的意图而定。

为禁止两个进程同时进入临界区内，操作系统提供同步机构来协调进程的关系，该同步机构都应遵循下述准则：

① 当有若干进程欲进入它的临界区时，应在有限时间内使进程进入临界区。换言之，它们不应相互阻塞而致使彼此都不能进入临界区。

② 每次至多有一个进程处于临界区。

③ 进程在临界区内仅逗留有限的时间。

一般采用同步机构实现进程互斥。下面具体介绍用于实现进程互斥的同步机构。

2. 同步机构

为了实现用户进程的同步和互斥，操作系统必须提供同步机构，最常用的是信号灯及其

在信号灯上的P、V操作。

(1) 信号灯的概念

信号灯是铁路交通管理中常用的一种设备,交通管理人员利用信号灯的状态(颜色)实现交通管理。操作系统中使用的信号灯正是从交通管制中引用过来的一个术语。

信号灯是一个数据结构,由两个数据项S、Q组成,S是一个具有非负初值的整型变量,Q是一个初始状态为空的队列指针。整型变量S代表资源的实体,操作系统利用它的状态对并发进程、共享资源进行管理。信号灯的值只能通过P、V操作来改变,其可能的取值范围是负整数值、零、正整数值。信号灯是操作系统中实现进程间同步和通信的一种常用工具。

一个信号灯的建立必须经过说明,即应该准确说明S的意义和初值(注意,这个初值必须不是一个负值)。每个信号灯都有相应的一个队列,在建立信号灯时,队列为空。

(2) P操作

信号灯的数值仅能由P、V操作加以改变。

对信号灯的P操作记为P(S)。P(S)是一个不可分割的原子操作,即取信号灯值,将其减1,若相减结果为负,则调用P(S)的进程被阻,并插入到该信号灯的等待队列中,否则继续执行。

P操作的主要动作如下:

① S值减1。

② 若相减结果大于或等于0,则进程继续执行。

③ 若相减结果小于零,该进程被封锁,并将它插入到该信号灯的等待队列中,然后转入进程调度。

(3) V操作

对信号灯的V操作记为V(S)。V(S)是一个不可分割的原子操作,即取信号灯值,将其加1。若相加结果大于零,进程继续执行,否则,要唤醒在信号灯等待队列上的一个进程。

V操作的主要动作如下:

① S值加1。

② 若相加结果大于零,进程继续执行。

③ 若相加结果小于或等于零,则从该信号灯的等待队列中移出一个进程,解除它的等待状态,然后返回本进程继续执行。

3. 用信号灯实现进程互斥

利用信号灯能方便地解决互斥问题。首先应设置用于互斥的信号灯,该信号灯的初值一定为1,表示临界资源未被占用。其次,应把进入临界区的操作置于对该互斥信号灯的P、V操作之间,即可实现进程互斥。

设两个并发进程 A 和 B，具有相对于变量 N 的临界段 CS_A 和 CS_B，用信号灯实现它们的互斥情况可描述如下。

设置互斥的信号灯 mutex，赋初值为 1。

进程 A：	进程 B：
⋮	⋮
P(mutex)；	P(mutex)；
CS_A；	CS_B；
V(mutex)；	V(mutex)；
⋮	⋮

进程 A 和进程 B 在并发执行，它们谁都有可能先执行 P 操作语句。由于互斥信号灯的初始值为 1，故在第一个进程执行 P 操作后减为 0，表示临界资源初始状态为空闲，可分配给该进程，使之进入临界区。当该进程执行 P 操作通过后，表示它已占用临界资源，进入了临界区，若此时又有第二个进程欲进入临界区，也应先执行 P 操作，结果使 mutex 变为负值，这就意味着临界资源已被占用，因此，第二个进程被阻塞。并且，直到第一个进程执行 V 操作，释放临界资源而恢复 mutex 值为 0 后，方唤醒第二个进程，使之进入临界区，待第二个进程完成对临界资源的访问后，又执行 V 操作，使 mutex 恢复到初始值。

当两个进程共享临界资源，用互斥信号灯实现进程互斥时，互斥信号灯的值仅取 1、0 和 -1 3 个值。

- 若 mutex = 1，表示没有进程进入临界区。
- 若 mutex = 0，表示有一个进程进入临界区。
- 若 mutex = -1，表示一个进程进入临界区，另一个进程等待进入。

4. 同步的概念

(1) 什么是进程同步

进程互斥主要是解决并发进程对临界区的使用问题。这种基于临界区控制的交互作用是比较简单的，只要诸进程对临界区的执行时间上实现互斥，就能保证程序的正确执行。

另外，还需要解决更广泛的进程同步问题。一组相互合作的并发进程，其中每一个进程都以各自独立的、不可预知的速度向前推进，但它们又需要密切合作，以实现一个共同的任务。在这些进程执行过程中，需要互通消息，相互协调。例如，相互合作的进程之间需交换一定的信息，当某进程未获得其合作进程发出的消息之前，该进程就等待，直到所需信息收到时才变为就绪状态(即被唤醒)，以便继续执行，从而实现了诸进程的协调运行。

所谓同步，就是并发进程在一些关键点上可能需要互相等待与互通消息，这种相互制约的等待与互通消息称为进程同步。同步意味着两个或多个进程之间根据它们一致同意的协议进行相互作用，即这些进程之间存在同步关系或称同步规则，要实现同步，一定要保证同步关系不受破坏，遵循同步规则。

(2) 进程同步的例子

同步的例子不仅在操作系统中有,在日常生活中也大量存在。例如,医生为某病人看病,认为需要作某些化验,于是,就为病员开了化验单,病人取样送到化验室,等待化验完毕交回化验结果,然后继续看病。医生为病人看病是一个进程,化验室的化验工作又是另一个进程,它们是各自独立的活动单位,但它们共同完成治疗任务,所以需要交换信息。上述这两个合作进程之间有一种同步关系:化验进程只有在接收到看病进程的化验单后才开始工作;而看病进程只有获得化验结果后才能继续为该病人看病,并根据化验结果确定医疗方案。

下面举一个操作系统中进程合作的例子。假定计算进程和打印进程共同使用一个单缓冲,其中计算进程对数据进行计算,打印进程打印计算结果。当计算进程对数据的计算尚未完成,未把结果送入缓冲区之前,打印进程无法执行打印操作。一旦计算进程把计算结果送入缓冲区时,就应该给打印进程发出一信号,打印进程收到该信号后,便可从缓冲区取出计算结果打印。反之,在打印进程未把缓冲区的计算结果取出打印之前,计算进程也不能再把下一次计算结果送入缓冲区。这同样需要打印进程在取走缓冲区中的计算结果时,给计算进程发送一信号,而计算进程只有在收到该信号后,才能将下一个计算结果送入缓冲区。计算进程和打印进程之间就是用这种发信号方式实现同步的。

5. 用信号灯实现进程同步

上述的计算进程和打印进程的同步问题是共享缓冲区的同步。下面讨论这类问题的同步规则及信号灯解法。

设某计算进程 CP 和打印进程 IOP 共用一个单缓冲,如图 3.3.6 所示。其中,CP 进程负责不断地计算数据并送入缓冲区 T 中,IOP 进程负责从缓冲区 T 中取出数据去打印。

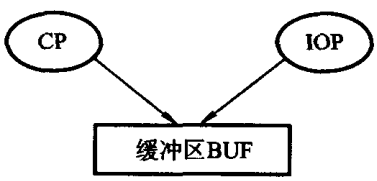


图 3.3.6 计算进程和打印进程

CP 进程和 IOP 进程必须遵循以下同步规则:

① 当 CP 进程把计算结果送入 BUF 时,IOP 进程才能从 BUF 中取出结果去打印,即当 BUF 内有信息时,IOP 进程才能动作,否则必须等待。

② 当 IOP 进程把 BUF 中的数据取出打印后,CP 进程才能把下一个计算结果送入 BUF 中,即只有当 BUF 为空时,CP 进程才能动作,否则必须等待。

为了遵循这一同步规则,这两个进程在并发执行时必须通信,即进行同步操作。为此,

设置两个信号灯 S_A 和 S_B。信号灯 S_A 表示缓冲区中是否有可供打印的计算结果，其初值为0。每当计算进程把计算结果送入缓冲区后，便对 S_A 执行 $V(S_A)$操作，表示已有可供打印的结果。打印进程在执行前须先对 S_A 执行 $P(S_A)$操作。若执行 P 操作后 $S_A=0$，则打印进程可执行打印操作；若执行 P 操作后 $S_A<0$，表示缓冲区中尚无可供打印的计算结果，打印进程被阻。信号灯 S_B 用以表示缓冲区有无空位置存放新的信息，其初值为 1。当计算进程算得一个结果，要放入缓冲区之前，必须先对 S_B 作 $P(S_B)$操作，看缓冲区是否有空位置。若执行 P 操作后 $S_B=0$，则计算进程可以继续执行，否则，CP 进程被阻，等待 IOP 进程从缓冲区取走信息后将它唤醒。打印进程把缓冲区中的数据取走后，便对 S_B 执行 $V(S_B)$操作，与 CP 进程通信，即告之缓冲区信息已取走，又可存放新的信息了。上述这两个进程之间的同步描述如图 3.3.7 所示。

$S_A=0$，表示缓冲区中有无信息

$S_B=1$，表示缓冲区中有无空位置

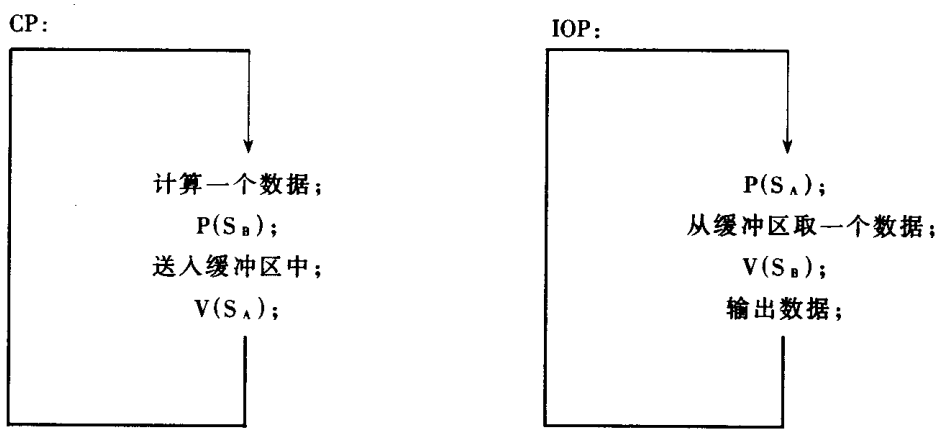


图 3.3.7　计算进程和打印进程的同步描述

用类 Pascal 语言描述生产者—消费者问题，形式如下：

```
PROGRAM
    S_A: =0;表示缓冲区中有无信息
    S_B: =1;表示缓冲区中有无空位置
    COBEGIN
    CP:
        WHILE 计算未完成 DO
            BEGIN
                    ⋮
              计算一个数据;
```

```
            P(S_B);
            送入缓冲区中;
            V(S_A);
        END;
    IOP:
        WHILE 输出未完成 DO
            BEGIN
                P(S_A);
                从缓冲区取一数据;
                V(S_B);
                输出数据;
            END
    COEND
ENDPROGRAM
```

3.3.7 线程

前面已讨论了有关进程的概念，进程是程序在处理机上的一次执行过程，它具有一个单一的控制路径和一个地址区域。在多任务系统中，进程是系统中进行处理机调度的最小单位。为了进一步提高系统的并行处理能力。在现代操作系统中，例如 Windows 系统又使用了一个叫线程(threads)的概念。为什么要提出线程概念？什么是线程？在本节中作一简单讨论。

1. 什么是线程

在有些现代操作系统中，提供了对单个进程中多条控制线索的支持。这些控制线索通常称为线程(threads)，有时也称为轻量级进程(lightweight processes)。

线程是比进程更小的活动单位，它是进程中的一个执行路径。一个进程可以有多条执行路径，即线程。这样，在一个进程内部就有多个可以独立活动的单位，可以加快进程处理的速度，进一步提高系统的并行处理能力。

线程可以这样来描述：

① 进程中的一条执行路径。

② 它有自己私用的堆栈和处理机执行环境(尤其是处理器寄存器)。

③ 它与父进程共享分配给父进程的主存。

④ 它是单个进程所创建的许多个同时存在的线程中的一个。

线程和进程既有联系又有区别，对于进程的组成，可以高度概括为以下几个方面：

① 一个可执行程序，它定义了初始代码和数据。

② 一个私用地址空间(address space)。它是进程可以使用的一组虚拟主存地址。

③ 进程执行时所需系统资源,如文件、信号灯、通信端口等。它们是程序执行时,由操作系统分配给进程的。

④ 若系统支持线程运行,那么,每个进程至少有一个执行线程。

进程是任务调度的单位,也是系统资源的分配单位;而线程是进程中的一条执行路径,当系统支持多线程处理时,线程是任务调度的单位,但不是系统资源的分配单位。线程完全继承父进程占有的资源,当它活动时,具有自己的运行现场。

2. 线程的特点与状态

(1) 线程的特点

创建一个线程和对它们进行管理,与创建进程和进程管理相比较,开销要小得多。因为线程可以共享父进程的所有程序和全局数据,这意味着创建一个新线程只涉及最小量的主存分配(线程表),也意味着一个进程创建的多个线程可以共享地址区域和数据。

在进程内创建多线程,可以提高系统的并行处理能力。例如,一个文件服务器,某时刻它正好封锁在等待磁盘操作上,如果这个文件服务器进程具有多个控制线程,那么当另一个线程在等待磁盘操作时,第二个线程就可以运行,比如它又可接收一个新的文件服务请求。这样可以提高系统的性能和提供较高的信息流通量。又如,在当前的 Windows 系统中,字处理程序要装入一个大型文档时,可以生成一个线程来安装这个大型文档,再生成另一个线程来控制 Cancel 按钮,用于管理中途中止,那么,不论用户何时想中断文件装入操作,都可以用击键来中断它。

(2) 线程的状态及变迁

如果一个系统支持线程的创建与线程的活动,那么,处理机调度的最小单位是线程而不是进程。一个进程可以创建一个线程,那么它具有单一的控制路径,一个进程也可创建多个线程,那么它就具有多个控制路径。这时,线程是争夺 CPU 的单位。线程也有一个从创建到消亡的生命过程,在这一过程中它具有运行、等待、就绪或终止几个状态。

① 创建。建立一个新线程,新生的线程将处于新建状态。此时它已经有了相应的主存空间和其他资源,并已被初始化。

② 就绪。线程处于线程就绪队列中,等待被调度。此时它已经具备了运行的条件,一旦分到 CPU 时间,就可以立即去运行。

③ 运行。一个线程正占用 CPU,执行它的程序。

④ 等待。一个正在执行的线程如果发生某些事件,如被挂起或需要执行费时的 I/O 操作时,将让出 CPU,暂时中止自己的执行,进入阻塞状态。等待另一个线程唤醒它。

⑤ 终止。一个线程已经退出,但该信息还没被其他线程所收集(在 UNIX 术语中,父线程还没有做 wait)。

线程与进程一样,是一个动态的概念,也有一个从产生到消亡的生命周期,如图 3.3.8

所示。

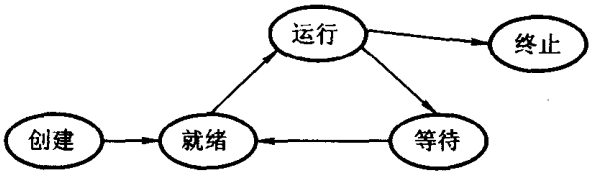


图 3.3.8 线程的生命周期

线程在各个状态之间的转化及线程生命周期的演进是由系统运行的状况、同时存在的其他线程和线程本身的算法所共同决定的。在创建和使用线程时应注意利用线程的方法宏观地控制这个过程。

3.4 操作系统资源管理

3.4.1 资源管理功能和分配策略

操作系统的重要功能之一是管理计算机系统的硬件和软件资源。资源管理的目的是为用户提供一种简单而有效地使用资源的方法,并能充分发挥各类资源的使用效率。计算机系统的资源按其特性可分为 4 类:处理机、存贮器、输入/输出设备和软件资源。操作系统的资源管理功能也分为 4 个部分:处理机管理、存贮管理、输入/输出管理和信息管理(文件系统),这 4 大管理功能与之对应。这些资源的性质各不相同,它们都具有各自的"个性",但从资源这一角度来看,它们还具有共性的部分。所以讨论操作系统对系统资源的管理功能,应首先介绍资源管理的功能与策略。

1. 资源管理功能

对资源进行管理,操作系统必须提供各种资源管理的模块,这些模块的任务是:解决资源分配、资源存取与使用方法等问题,并提供对资源的存取控制和安全保护措施。不论是哪类资源,对其进行分配和管理时,必须具备以下几个方面的功能。

(1) 定义资源数据结构

为了完成资源管理任务,首先要建立相应的数据结构,它能清晰地描述该类资源的名、使用特性,哪些是空闲的,哪些已被使用,是由谁占用等信息。它是进行资源分配的依据。

(2) 确定资源的分配原则(或称调度原则)

当有许多申请者申请某类资源时,系统必须确定一个原则,用来决定空闲资源分给谁、分多少、何时分配等问题。比如一种可能的原则可按申请者请求的先后次序来确定,另一个

可能的原则是按请求者请求该资源的紧迫程度来决定。

(3) 执行资源分配与回收

根据资源分配的原则以及用户的要求,执行资源的分配。当其不再需要时,收回资源,以便重新分配给其他请求者使用。

(4) 存取控制和安全保护

存取控制和安全保护问题越来越引起人们的重视。这一问题在各类资源管理中都是存在的,尤其是对程序资源——文件的管理最为突出。任何一个用户对任一文件的存取,都要经过文件管理系统中的存取控制验证模块的检查。只有合法的用户进行合法的操作才能通过合法性检查,否则将为系统所捕俘。由于对某一文件的操作将转换成对某个设备(磁盘或字符设备)的操作,所以对某些外部设备的存取可以认为在它的上一层已进行了合法性检查。当然,根据实际需要也可对各种外部设备作进一步的存取权限的检查。对主存单元的存取同样也要经过主存保护硬件的检查,只有检查通过者才能进行相应操作,以保证同存于主存的各个用户程序的隔离。有的系统对磁盘的某些操作采用锁、密码的方法,以实现对磁盘的存取控制。至于对中央处理机的存取权,用户可以认为处于就绪队列的进程有着存取 CPU 的权利。

2. 常用的资源分配策略

(1) 先请求先服务

先请求先服务资源分配策略是一种最简单的分配策略,一般称为简单排队策略或 FIFO (First In First Out,先进先出)策略。这种先请求先服务的策略不对请求的特征、执行时间长短等作任何考虑,其好处是实现较简单。与该策略相适应的队列是按照提出请求的先后次序排序的。每一个新产生的请求均排在队尾,而当资源可用时,资源分配程序则从队列中选取第一个请求,并且满足需要。

(2) 优先调度策略

优先调度策略是一种比较灵活的调度策略,它可以优先照顾需要尽快处理的那些请求者的要求。在这种调度策略下,每个请求者必须事先指派一个优先级,这一优先级反映了请求者要求处理的紧迫程度。与该策略相适应的队列是按请求者的优先级高低排序的。每一个新产生的请求按其优先级排在队列的适当的地方;而当资源可用时,资源分配程序总是送取队列的首元素(即优先级最高者),满足它的需要。

3.4.2 处理机管理

计算机系统中最关键的资源之一是中央处理机(CPU)。多个应用程序或多进程轮流占用 CPU 的时间来共享中央处理机。那么,处理机时间如何分片? 为适应不同需要和满足不同系统的特点,时间片的长短如何确定? 以什么策略分配处理机? 谁先占用? 谁后占用? 这些就是处理机分配的策略问题。另外,还要考虑中央处理机具体的分配问题。所以,处理

机管理的功能是决定分配策略，实施处理机控制权的转接，保证选中进程在处理机上正确运行。

1. 调度的层次

现代操作系统都提供多用户、多任务的运行环境。对处理机的调度问题应分两个步骤来考虑，首先，要决定选择哪些用户的应用程序进入系统；其次，决定哪一个进程可在处理机上执行。

在批处理系统中，有众多用户提供的大量作业存放在磁盘上，形成一个作业队列。如何从这众多的作业中选择一个或几个进入系统内，进入系统的若干进程又如何去争夺 CPU 的控制权呢？这就是处理机调度问题。处理机调度可以分成两级。

（1）宏观调度

宏观调度又称高级调度、作业调度。其主要任务是对磁盘设备上大量的后备作业，以一定的原则进行挑选，给选中的作业分配主存等必要的资源，建立其相应的进程，让它投入运行，这时该作业对应的进程具有使用 CPU 的权利。

（2）微观调度

微观调度又称低级调度。其任务是按照某种原则将处理机分配给系统中最小的活动单位——进程，即确定哪个进程在什么时候获得处理机，使用多长时间。应该注意到，当某个进程要占用处理机时，系统必须建立与其相适应的状态环境，以确保选定进程的正确执行。

在多用户批处理系统中，对处理机的分配分为作业调度和进程调度两级。在这样的系统中，每个用户提交的算题任务，作为系统的一个处理单位，称为作业。这样一道作业在处理过程中又可以分为多个并发的活动单位，称为进程。宏观调度和微观调度的任务各不相同，但又有一定的联系。前者选择作业进入主存，到底哪个进程能获得 CPU 的控制权，则由进程调度来确定。

在分时系统中，支持多用户多任务并发执行；而个人计算机操作系统一般为单用户多任务操作系统。在这两类操作系统中，系统将用户提交的任务处理为进程，一个进程又可以创建多个子进程，这些进程都是分配资源和处理机的单位。所以，在支持多进程运行的系统中，处理机调度就是进程调度。系统创建进程时，应为该进程分配必要的资源。而进程调度要完成的任务是，当处理机空闲时，以某种策略选择一个就绪进程去运行，并分配处理机时间。

在现代操作系统中，有些系统支持多线程运行。在这样的系统中，一个进程可以创建一个线程，也可以创建多个线程。这时系统中活动的最小单位是线程，其微观调度称为线程调度，其功能是当处理机空闲时，以某种策略选择一个就绪线程去运行，并分配处理机时间。

下面主要讨论进程调度的功能。

2. 进程调度的功能

在多进程系统中，众多进程争夺 CPU 的执行权。操作系统应具备进程调度的能力，按

一定的策略，动态地把处理机分配给就绪队列中的某一进程，以便使之执行。进程调度的具体功能是：

（1）记录进程的状态特征

进程调度的功能之一是记录和保持系统中所有进程的有关情况和状态特征。该功能所用到的数据结构是进程控制块 PCB 以及进程队列，并由进程控制模块（如进程创建、进程撤销、进程通信等功能模块）来实施。进程在活动期间的状态是可以改变的，如由运行变为等待，由等待变为就绪，由就绪变为运行。在发生这些状态变化时，该进程的 PCB 结构就在运行指针、各种等待队列和就绪队列之间移动。

（2）决定分配策略

处理机管理的一个重要功能是，当处理机空闲时，根据一定的原则在就绪队列中选择一个进程去运行，同时确定获得处理机的时间。常用的分配策略有两个：先请求先服务与优先调度策略。处理机分配策略的实施是依靠就绪队列的排序原则来体现的。若采用优先调度原则，进程就绪队列按优先级高低排序，若采用先来先服务原则，进程就绪队列按进程来到的先后次序排序。图 3.4.1 说明了这种情况。进程就绪队列按规定的原则排好序后，当处理机空闲时，只要选择队首元素就一定满足确定的调度原则。

（3）实施处理机的分配和回收

当调度时机来到时，根据调度原则选择一个进程去运行，把选中进程从就绪队列中移出，状态改为“运行”，并将选中进程的有关 PCB 现场信息，如保留的通用寄存器的内容、处理机状态、程序执行地址等分别送到相应的寄存器中，真正把 PCB 控制权交给被选中的进程。

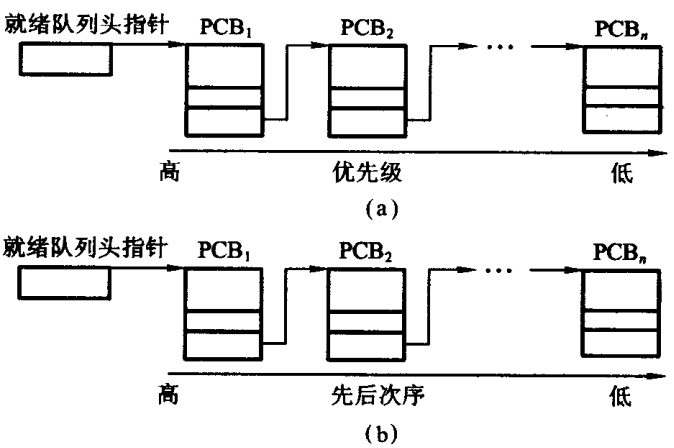


图 3.4.1　进程就绪队列排序原则

3. 进程调度算法

(1) 进程优先数调度算法

进程优先数调度算法是,当中央处理机空闲时,系统把处理机的使用权赋予就绪队列中具备最高优先权的就绪进程。一般优先权和一定的优先数相对应,而且,系统会预先确定各进程的优先数。

优先数可以按静态方式或动态算法指派给进程。以静态方式指派给进程称为静态优先数,它在进程创建时确定,且一经确定后在整个运行期间不再改变。而动态优先数是在进程运行期间根据进程的运行特征或占用系统资源的变化情况动态地改变其优先数,从而获得更好的调度性能。一般在实时系统中采用这种进程调度算法,以保证实时进程能得到及时响应。

(2) 循环轮转调度

在循环轮转调度算法下,进程就绪队列按先来后到的次序排序。每当 CPU 空闲时,取就绪队列首元素去运行,并分配一个时间片。当该进程的时间片用完时,转为就绪态并进入就绪队列末端。一般在分时系统中采用这种进程调度算法,以保证每个用户进程能得到公平的响应。

在循环轮转调度算法中,有一种简单的循环轮转调度。这种算法的特征是,就绪队列中的所有进程均以相等的速度向前推进。如果就绪队列中有 k 个就绪进程,时间片的长度为 Q 秒,则每个进程在每 kQ 的时间内可获得 Q 秒的 CPU 时间,亦即每个进程是以 $1/k$ 的实际 CPU 速度运行在处理机上。较复杂的循环轮转调度有可变时间片轮转调度和多就绪队列轮转调度。

4. 进程调度时机与调度方式

(1) 进程调度时机

进程调度程序什么时候被激活,这是一个调度时机问题。当现行程序不能再继续运行,或者有理由认为应把处理机用于另一进程时,进程调度程序将投入工作,这就是调度时机。调度时机有以下几种可能:

① 进程完成其任务。

② 在一次系统功能调用之后,该调用使现行程序暂时不能继续运行。

③ 在一次出错陷入之后,该陷入使现行进程在出错处理时被挂起。

④ 在分时系统中,当进程使用完规定的时间片,时钟中断使该进程让出处理机。

⑤ 在采用可剥夺调度方式的系统中,当具有更高优先级的进程要求处理机时。

(2) 调度方式

在优先调度策略下还要确定调度方式。这里所谓的调度方式是指,当一进程正在处理机上执行时,若有某个更为"重要而紧迫"的进程需要进行处理,亦即,若有优先数更高的进程进入就绪队列时,如何分配处理机。通常有两种进程调度方式:

① 一种是仍然让正在执行的进程继续执行，直到该进程完成或发生某事件（如提出I/O请求）而进入“完成”或“阻塞”状态时，才把处理机分配给“重要而紧迫”的进程，以使之执行。这种进程调度方式称为非剥夺方式。

② 另一种方式则是“重要而紧迫”的进程一到，便暂停正在执行的进程，立即把处理机分配给它。这种方式称为剥夺调度方式。

系统若可剥夺调度方式，则称该系统所实施的策略是可抢占的调度策略。

3.4.3 存储管理

1. 主存管理的功能

主存储器也是计算机系统的重要资源之一，它为操作系统、各种系统程序和用户程序所共享，任何程序的执行最终都要从主存中存取指令和数据，都必须和主存打交道。本节讨论主存管理中的基本概念与主存管理功能。

（1）主存管理中的基本概念

现代操作系统可区分两类主存：物理主存和逻辑主存。任何系统程序或应用程序最终都存放在物理主存中，因为程序执行时，CPU能直接存取信息的部件是物理主存。但用户所看到的，或应用程序所涉及到的是逻辑主存，这样，不仅可以方便用户编程，又能进行主存动态分配，现代操作系统都提供逻辑主存的概念。

① 物理地址。物理地址是计算机主存单元的真实地址，又称为绝对地址或实地址。处理机依据物理地址可以随机存取存放在其内的信息。

② 主存空间。主存空间又称为存储空间，它是物理地址的集合所对应的空间。

③ 逻辑地址。逻辑地址又称为虚地址或相对地址。用户的程序地址（指令地址或操作数地址）均为逻辑地址。对于每个逻辑地址，在主存中并没有一个固定的、实在的物理单元与之对应。因此，根据逻辑地址还不能直接到主存中存取信息。

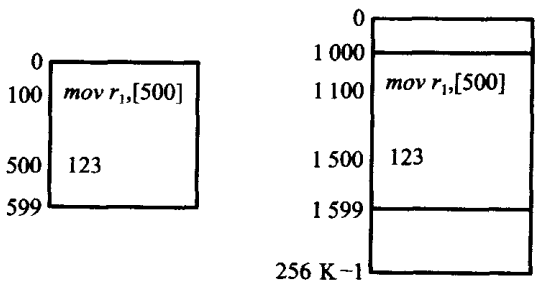
④ 虚拟地址空间。逻辑地址的集合组成了程序地址空间（通常称为作业地址空间）。为了支持多用户运行，提供方便的编程环境，系统为每个用户程序提供0～(n-1)个逻辑地址（虚地址），即提供一个虚拟地址空间。

用户面前的虚存（逻辑地址）与被共享的主存（物理地址）之间有一定的映射关系。程序执行时，必须将逻辑地址正确地转换为物理地址，此即为地址映射。

（2）地址映射

① 什么是地址映射

处理机在执行指令时，必须把逻辑地址转换为物理地址后方能访问信息。而在多用户共享主存时，由系统分配主存。一般情况下，一个应用程序分配到的存储空间和它的地址空间是不一致的。因此，该应用程序所对应的进程在处理机上运行时，所要访问的指令和数据的实际地址和地址空间中的地址是不同的。这种情况可用图3.4.2来说明。



(a)作业地址空间 (b) 存储空间

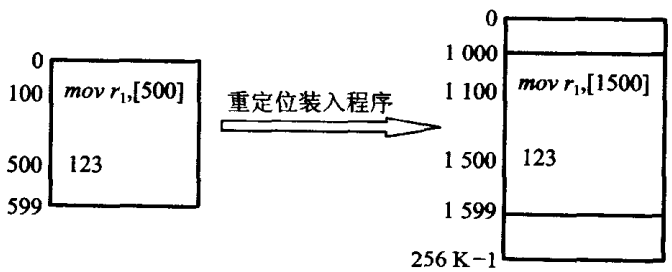
图 3.4.2 作业的地址空间装入主存

在程序的地址空间中 100 号单元处有一条指令“mov r_1[500]”。执行这条指令的结果是,将 500 号单元处的数 123 送到 r_1 寄存器中。如果现在将该程序只是简单地装入到主存 1 000 ~ 1 599 号单元,当执行到 1 100 号单元处的一条指令时,则将 500 号单元的内容送到 r_1 寄存器。这显然是错误的。由图 3.4.2 可见,正确的执行应该将 1 500 号单元的内容送到 r_1 寄存器。为此,要修改 1 100 号单元中指令的地址部分,即把其逻辑地址 500 改为 1 500。

所谓地址映射,又称地址重定位,是指将程序地址空间中使用的逻辑地址变换成主存中的物理地址的过程。存储管理的功能之一就是实现这种地址映射,地址映射的类型根据地址映射的时机不同,分为静态地址映射和动态地址映射两类。

② 静态地址映射

在程序装入主存时,实现程序的逻辑地址到主存的物理地址的变换方式称为静态地址重定位或静态地址映射。程序经过编译和连接后,形成一个可以浮动的程序模块,即作业地址空间。当将这一地址空间装入到主存中的任一位置时,由重定位装入程序对有关地址部分进行调整,即将与地址有关的部分都加上该程序在主存的起始地址。如图 3.4.3 所示,指令地址 100 调整为 1 100;操作数地址 500 调整为 1 500,经调整后的程序就可以正确地运行了。



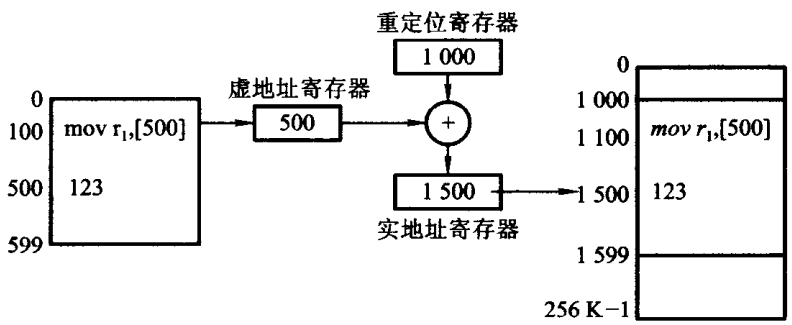
(a)作业地址空间 (b)存储空间

图 3.4.3 静态地址映射的实现

由于需要重定位寄存器对程序进行静态调整，所以，要花费大量的 CPU 时间。另外，一个已开始执行的程序是无法在主存中移动的，所以现代操作系统一般采用动态地址映射方式。

③ 动态地址映射

动态地址映射是指在程序执行期间，随着每条指令和数据的访问自动地连续地进行映射。这种重定位的实现需要硬件提供支持，一般是靠硬件地址变换机构实现的。最简单的硬件机构是一个重定位寄存器。图 3.4.4 给出了利用重定位寄存器实现动态重定位的过程。当某个进程取得 CPU 控制权时，操作系统负责将该进程对应的应用程序在主存的起始地址送入重定位寄存器中，之后，在进程的整个运行过程中，重定位寄存器始终起着地址变换的作用。每次访问存储器时，重定位寄存器的内容将被自动地加到逻辑地址中去。经这样变换后，执行的结果是正确的。图 3.4.4 中，程序直接装入主存从 1 000 号单元开始的一个区域中。在它开始执行之前，由操作系统将重定位寄存器置为 1 000。当程序执行到 1 100号单元处的指令时，CPU 给出的取数地址为 500，而所希望的数是存放在 1 500 号单元内，故经地址变换后得到的地址为 1 500，然后以它作为访问主存的物理地址，执行结果是完全正确的。



（a）作业地址空间　　（b）存储空间

图 3.4.4　动态重定位的实现

这种地址变换方式比静态重定位要好。因为，动态重定位则是由硬件自动完成的，而且重定位寄存器的内容可由操作系统用特权指令来设置，比较灵活。

（3）主存分配

在现代操作系统中，主存分配的功能包括制定分配策略、构造分配用的数据结构，响应主存分配请求，进行主存的分配与回收工作。

① 管理存储器的策略

管理存储器的策略有以下 3 种类型：

（a）放置策略——决定如何在若干个空闲存储区中选择一个空闲区的原则。

(b) 调入策略——决定信息装入主存的时机,是在需要信息时调入,还是预先调入。这是两种不同的调入策略,前者为请调策略,后者为预调策略。

(c) 淘汰策略——在主存中没有任何可用的空闲区时,决定哪些信息可从主存中移走,也就是确定如何建立一个空闲区的原则。

② 主存分配

不同的主存管理技术对主存区域的划分是不同的。有两种划分方式:一是将主存划分成大小不等的区域,二是将主存等分为一系列大小相等的块。按区分配或段式分配采用第一种划分方式。这种方式使一个主存区域可以存放一个作业程序的连续的地址空间(按区分配),或存放一个作业的一个逻辑分段的地址空间(段式系)。而页式系统往往采用第二种划分方式。这种方式将一个作业程序的地址空间划分成一系列的页面,然后放置到主存的块中去。

为了进行主存分配,必须建立相应的数据结构。用于主存分配的数据结构有:空闲区队列(分区存储管理)或存储分块表(页式存储管理)。

主存分配包括分配和回收两个功能。当应用程序请求分配主存时,由存储管理的分配程序负责决定用户程序的主存位置并将程序装入主存。当应用程序运行完毕,释放主存时,由存储管理的回收程序负责将应用程序占用的主存空间收回,归还给系统。

(4) 虚拟存储器

随着科学技术的不断进步和计算机应用的日益广泛,需要计算机解决的问题越来越多,越来越复杂。由于应用的驱动,致使操作系统必须解决主存空间不够用的问题。

为了实现多用户环境下动态的主存分配,为了方便用户编程,现代操作系统提供虚拟存储器。虚拟存储器是用户看到的应用程序编程空间,即应用程序地址空间。操作系统将应用程序地址空间的一部分放入主存内,而其余部分放在辅存上。当所访问的信息不在主存时,由操作系统负责调入所需要的部分。所以,操作系统将主存和辅存统一管理,实现主存和辅存之间信息的动态调度。这样的计算机系统好像为用户提供了一个其存储容量比实际主存大得多的存储器,这个存储器称为虚拟存储器。

实现虚拟存储器的可行性是:大多数程序执行时,在一段时间内仅使用它的程序编码的一部分,并不需要在全部时间内将该程序的全部指令和数据都放在主存中。所以,装入程序地址空间的部分代码,也能正确执行。比如,在按名字进行工资分类和按工作证号进行工资分类的程序中,由于这二者每次必定只选用一种,所以只装入其中一部分程序,仍能正确执行。

虚拟存储器的核心,实质上是将程序的地址空间和主存存储空间相分离。程序的访问地址称为虚地址,它可以访问的虚地址范围叫做程序的虚地址空间 V。在指定的计算机系统中,可使用的物理地址范围叫做计算机的存储地址空间 R。虚地址空间可以比主存空间大,也可以比主存小。在多道运行环境下,操作系统为每个用户建立一个虚存,从而提供了

若干个虚存。每个用户可以在自己的地址空间中编制程序，在各自的虚存上运行。

实现虚拟存储器，需要有一定的物质基础。其一是需要有相当容量的辅存，以便足以存放多用户作业的地址空间。其二是要有一定容量的主存。其三是要有地址变换机构。然而，引进虚存后系统就必须在地址变换上花费开销。所以，设计虚拟存储器时，应在可能的情况下力求地址变换能快速地进行。

可以部分装入页面的分页存储管理技术、分段存储管理及段页式存储管理方法是利用虚拟存储手段扩充主存的具体例子。

（5）存储保护

在多用户环境下，系统必须防止用户之间的互相干扰，以保证程序的正确执行。例如，用户甲与用户乙各分配到一块存储空间，若用户甲的程序有错误，就可能向用户乙的存储空间中写入一些杂乱无章的内容，这时用户乙的程序即使是正确无误的，也没有办法运行下去。

存储保护是操作系统在硬件的支持下实现的技术，该技术保证每道程序只能在给定的存储区域内活动。存储保护的目的是为了防止用户之间的互相干扰而采取的隔离性措施。存储保护技术有上、下界防护与存储键防护等。

2. 动态分区存储管理技术

分区存储管理是满足多道程序设计的一种最简单的存储管理方法。系统根据应用程序的大小把主存划分成若干分区，使分区正好适应应用程序的需要，这种方法称为动态分区存储管理。

下面以图 3.4.5 所示的例子说明动态分区的分配情况。设某系统拥有 256 K 主存，操作系统占用顶端 20 K 主存区。作业队列请求中各作业要求的主存容量如下：作业 1（32 K）、作业 2（14 K）、作业 3（64 K）、作业 4（100 K）、作业 5（50 K）。

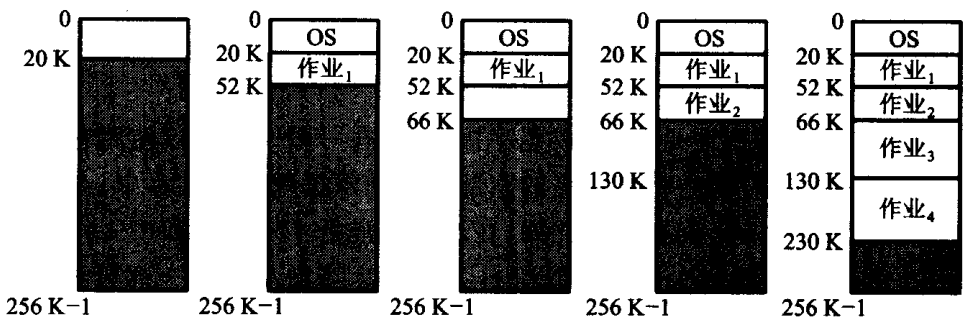


图 3.4.5　动态分区存储管理的动态分配过程

当作业完成时，它们要释放主存区域，在主存中形成一些空闲区，如图 3.4.6 中的斜线部分所示。这些空闲区可以被其他作业使用，但由于空闲区和容纳的作业的大小不一定正

好相等,因而这样剩余的空闲区域变得更小。当系统运行相当长一段时间后,主存中会出现一些更小的空闲区。

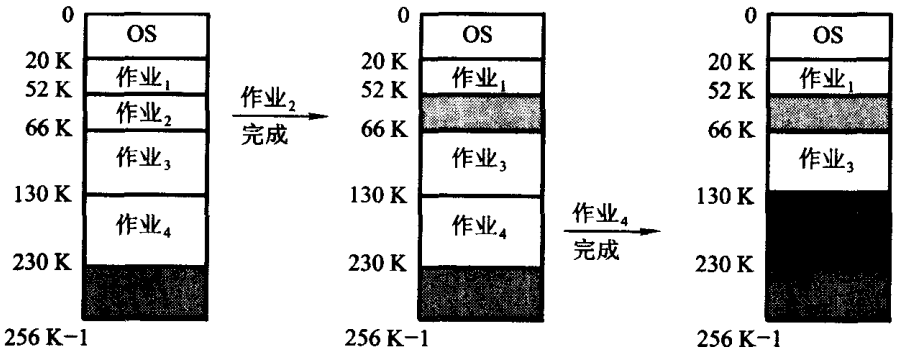
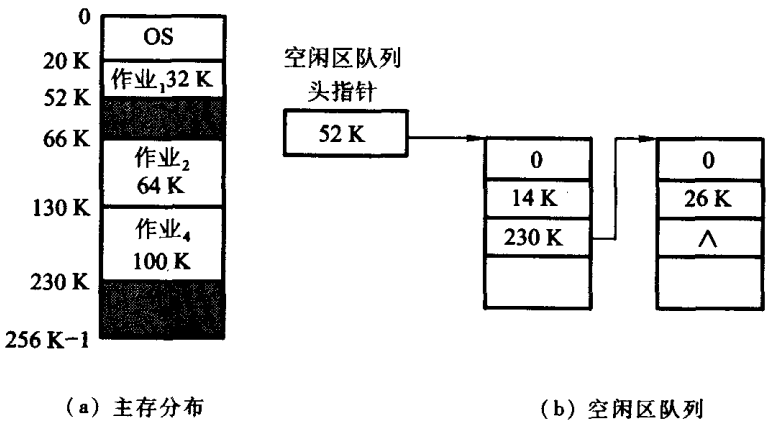


图 3.4.6 动态分区方法中存储区的释放

(1) 动态分区分配机构

在动态分区存储管理中,用于分配的数据结构是空闲区队列。每一个主存空闲区用空闲区描述器来描述。

空闲区描述器的内容包括空闲区标志、大小和地址,在队列结构中,空闲区地址就是勾链字的内容。图 3.4.7 给出了动态分区分配存储管理系统中某一时刻的主存使用情况、空闲区描述器和空闲区队列结构。



(a) 主存分布 (b) 空闲区队列

图 3.4.7 动态分区的空闲区队列

(2) 动态分区分配与回收

① 分区的分配

分区的分配是指系统根据应用程序的请求,在空闲区队列中寻找一个满足应用程序要

求的空闲区，划分与请求大小相等的分区作为已分配区，剩余部分仍留在空闲区队列中。

以图 3.4.7 所示的主存分配情况和空闲区队列结构为例说明动态分区分配的过程。当应用程序要求一个大小为 SIZE 的存储空间时，系统查询空闲区队列，找一个大于或等于 SIZE 的空闲区。这里有 3 种情况：

（a）系统中无满足要求的空闲区，分配失败。

（b）空闲区大小与 SIZE 相等，则将该空闲区从空闲区队列中摘下，修改相应的勾链字内容，向应用程序返回该空闲区首址。

（c）空闲区大于 SIZE，这时将空闲区一分为二，一为大小等于 SIZE 的分配区，系统向用户返回这个空闲区首址；另一个是余下部分，仍作为空闲区留在空闲区队列中，只是要修改空闲区首地址和大小。

② 分区的回收

当某个应用程序释放存储区时，系统需回收该存储区。首先检查释放区是否与系统中的空闲区相邻，若相邻则将释放分区合并到相邻的空闲区中，否则将释放分区作为一个空闲区插入到空闲区队列的适当位置。一个回收分区邻接空闲区的情况有 4 种，如图 3.4.8 所示。

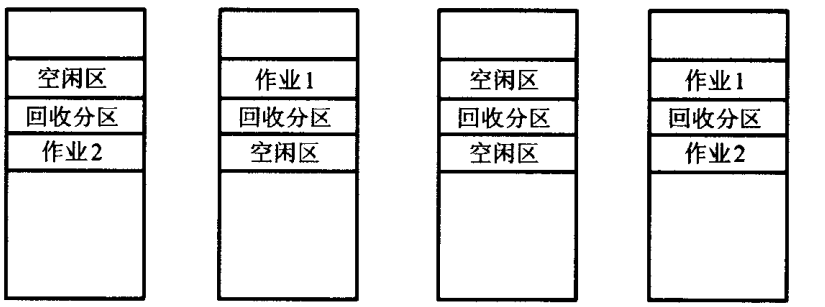


图 3.4.8　回收分区与空闲区邻接的几种情况

（a）回收分区上邻一个空闲区。合并成为一个连续的空闲区，其始址为回收分区上邻的空闲区始址，而大小为二者之和。

（b）回收分区下邻一个空闲区。合并后仍为空闲区，但该空闲区始址为回收分区的首地址，其大小为二者之和。

（c）回收分区与上、下空闲区邻接。这时应将这 3 个区域合并为一个连续的空闲区，其始址为回收分区上邻的空闲区首地址，大小为三者之和，并且应摘除与回收分区下邻的空闲区。

(d) 回收分区与空闲区不相邻接。这时应建立一个新的空闲区,并加入到空闲区队列中去。

(3) 动态分区分配的放置策略

① 空闲区的组织方法

分区分配的数据结构是空闲区队列。在进行分区分配时,依次查找空闲区队列中的每一个,只要找到第一个满足需要的空闲区就开始分割。因此,空闲区队列的排序原则就体现了选择一个空闲区的策略。队列的排序可以是无序的,即按照主存块释放的先后次序排列,也可以按某种分类方法进行排序。下面是两种常用的分类方法。

(a) 按空闲区地址增加或减少的次序排序。

(b) 按空闲区的大小增加或减少的次序排序。

空闲区队列的不同排序方法形成了不同的选择空闲区的策略,称为放置策略。最常见的放置策略有首次匹配、最佳匹配和最坏匹配。下面讨论首次匹配算法和最佳匹配算法。

② 首次匹配

首次匹配要求空闲区队列按空闲区首址递增的次序排序,该策略是将输入的作业放置到主存中第一个足够装入它的可利用的空闲区中。当要分配一个分区时,总是从低地址空闲区开始查寻,直到找到第一个足以满足该作业要求的空闲区为止。

这种算法的实质是,尽可能地利用存储器的低地址部分的空闲区,而尽量保存高地址部分的大空闲区,使其不至于被划分掉。其好处是,当需要一个较大的分区时,有较大希望找到足够大的空闲区以满足要求。

③ 最佳匹配

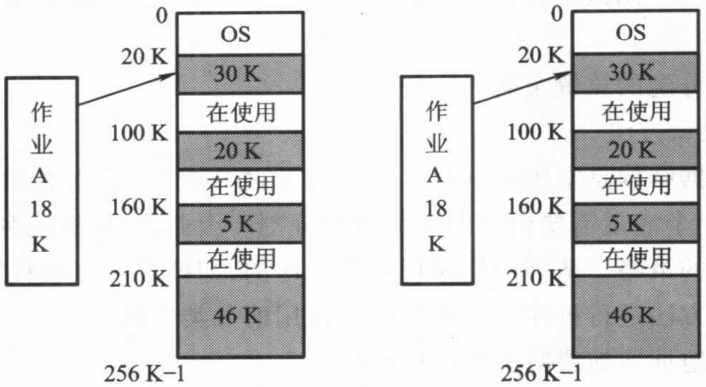
最佳匹配要求空闲区队列按空闲区大小由小到大的次序排序,该策略是将输入的作业放置到主存中与它所需大小最接近的空闲区中。最佳匹配看起来是一种最直观的、吸引人的策略,在进行分区分配时,总是从最小的一个空闲区开始查寻,因而找到的第一个能满足要求的空闲区便是最佳的一个(即从所要求的大小来看,该区和其后的所有空闲区相比它是最接近的)。

最佳匹配的主要缺点是,空闲区一般不可能正好和要求的大小相等,因而要将其分割成两部分,剩下的空闲区可能非常小,以至小到几乎无法使用,这将造成主存空间的浪费。

上述两种匹配策略的放置情况如图 3.4.9 所示。图中作业 A 大小为 18 KB,在两种不同的匹配策略下,被放置在不同的空闲区中。

这两种放置策略各有利弊,到底哪种好不能一概而论,应针对具体的作业序列来分析。对于某一作业序列来说,某种策略能将该作业序列中的所有作业安置完毕,则说该策略对这一作业序列是合适的,否则称为不合适。

设在图 3.4.9 所示的系统中(主存容量为 256 KB),有这样一个作业序列:作业 A 要求 18 KB;作业 B 要求 25 KB;作业 C 要求 30 KB。用首次匹配、最佳匹配策略来处理该作业序

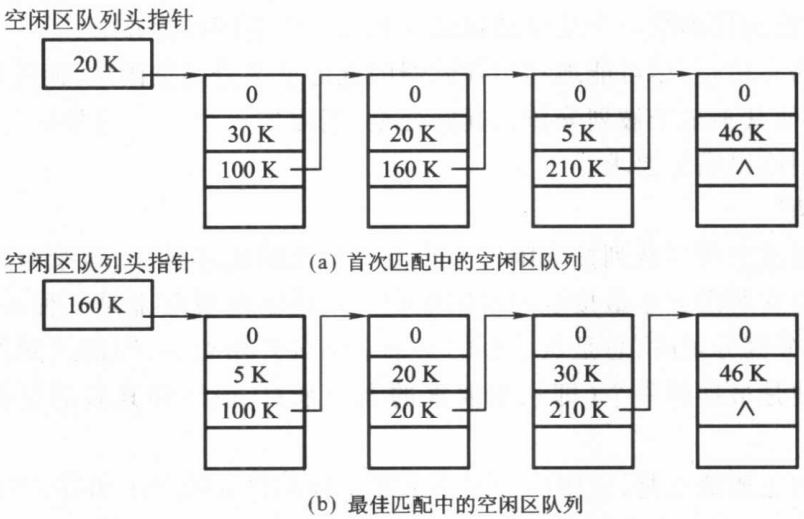


（a）首次匹配　　（b）最佳匹配

图 3.4.9　两种放置策略的说明

列，看哪种策略合适。

为了讨论这一问题，图 3.4.10 画出了在这 3 种策略下的空闲区队列。根据动态分区分配算法可以分析出：最佳匹配对该作业是合适的，首次匹配策略对该作业是不合适的。



（a）首次匹配中的空闲区队列

（b）最佳匹配中的空闲区队列

图 3.4.10　不同匹配策略的空闲区队列

（4）碎片问题及拼接技术

分区存储管理技术能满足多道程序设计的需要，但它也存在着一个非常严重的问题——碎片问题。所谓碎片，是指在已分配区之间存在着的一些没有被充分利用的空闲区。在按区分配方法中，由于空闲区的大小与申请的主存大小一般情况下不会相等，绝大多数情况是要从一个空闲区中分割一块作为已分配区，剩下的部分仍留在空闲区队列中。随着时

间的推移,空闲区的发展趋势是越来越小,直至不能满足任何用户要求。

解决这个问题的办法之一是采用拼接技术。所谓拼接技术,是指移动存储器中某些已分配区中的信息,使本来分散的空闲区连成一个大的空闲区,如图 3.4.11 所示。

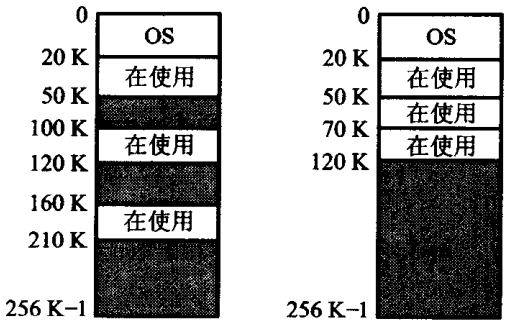


图 3.4.11 分区分配中的存储区拼接

拼接技术可以解决碎片问题,但由于拼接要消耗大量的系统资源,且有时为拼接所花费的系统开销要大于拼接技术的效益,因而这种方法的使用受到了限制。

3. 分页存储管理技术

为了寻找解决碎片问题的新途径,人们考虑是否能将应用程序的地址空间划分为若干个分片,把它们分别放置到主存中不连续的、小的空闲区中,这样能有效地解决分区存储管理中的碎片问题。这就是分页的概念。

(1) 分页系统的基本概念

① 页面和主存块

在分页存储管理中,应用程序的地址空间被等分成大小相等的片,称为页面,或称虚页。一个应用程序的所有页面从 0 开始顺序编号,称为页号。而主存也被等分成一系列的块,称为物理块,或称实页,简称为块。主存的块也从 0 开始顺序编号,称为块号。块的大小与页面的大小相等。当应用程序要进入系统时,它的各个页面会放置到主存的连续或不连续的块中。为了便于实现动态地址变换,一般主存的块和页面大小相等,并为 2 的幂次。主存分块和应用程序的地址空间分页的示意图如图 3.4.12 所示。采用这种方法,不需要移动主存原有的信息就解决了碎片问题,从而提高了主存的利用率。

② 页面映像表

程序的虚地址空间划分为若干页,并被装入主存的块中。于是,一个连续的程序地址空间在主存中可能是不连续的。为了保证该程序能正确地运行,必须在执行每条指令时将程序中的逻辑地址变换为实际的物理地址,即进行动态重定位。

在页式系统中,实现这种地址变换的机构称为页面映像表,简称页表。页表是虚页和物

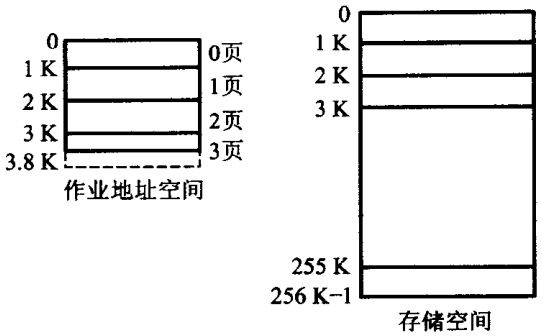


图 3.4.12　等分主存和虚地址空间

理块的对照表,它由两个数据项:页号与块号组成,如图 3.4.13 所示。

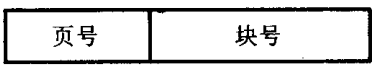


图 3.4.13　页表

页表可由高速缓冲存储器组成,这样做的结果是,地址变换速度快,但成本较高。另一个办法是,用主存的一个区域存放页表,系统为每个作业建立一张页表,系统中所有作业的页表形成页表区。图 3.4.14 给出了 3 个作业分页映像存储的情况。从图中可以看出每个作业有一张页表。页面尺寸的选择对分页存储管理而言是十分重要的。根据实际使用的经验,一般页面尺寸为 1 K、2 K 或 4 K。

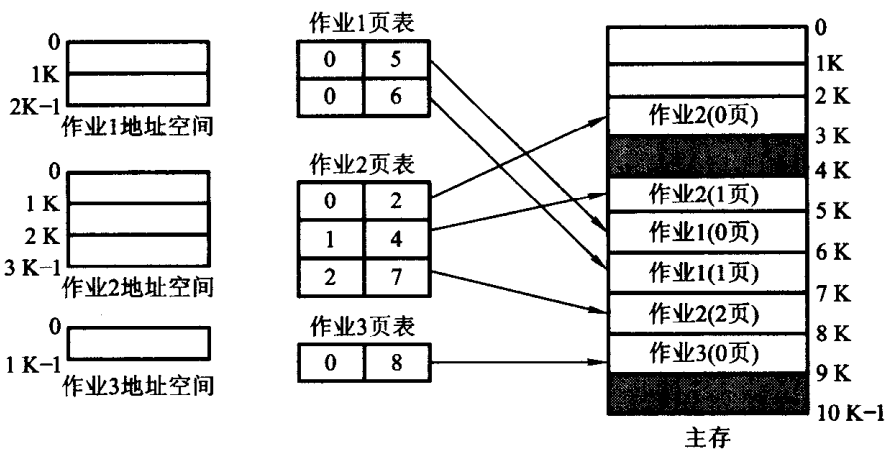


图 3.4.14　分页映像存储

(2) 页式地址变换

在页式系统中,首先要解决的问题是:应用程序地址空间中连续的页面被分配到主存连续或不连续的块中的时候,如何进行正确的地址变换,使程序能正确执行。在进行地址变换时使用的重要数据结构是页表。除此之外,还需了解计算机的虚地址结构。下面,先讨论虚地址结构,再介绍页式地址变换过程。

① 虚地址结构

在计算机系统中,程序地址字用来存放应用程序指令地址或操作数地址,它的位数及其结构称为虚地址结构。

进行页式地址变换,与计算机所采用的虚地址结构有关,而虚地址结构又与选择的页面尺寸有关。比如,当 CPU 给出的虚地址长度为 32 位,页面大小为 4 K 时,在分页系统中地址结构的格式如图 3.4.15 所示。

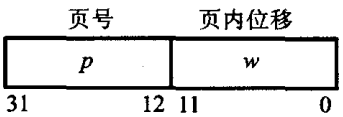


图 3.4.15 虚地址结构

每当 CPU 给出一个虚地址(指令地址或操作数地址)时,这个地址中的高 20 位(12 位~31 位)一定表示该地址所在的页号,而低 12 位(0 位~11 位)一定表示该地址在该页内的相对位移量。分页系统中具有这种特征的地址结构称为分页机构。

② 页式地址变换

下面以图 3.4.16 中所示的作业 2 程序中的一条指令的执行过程来说明页式系统的地址变换过程。程序地址空间中第 100 号单元处有一条指令为"mov r_1,[2500]"。这条指令在主存中的实际位置为 2 148 号单元(第 2 块 100 号单元),而这条指令要取的数 123 在程序地址空间中位于 2 500 号单元(第 2 页的 452 号单元)处,它在主存中存于 7 620 号单元(第 7 块 452 号单元)。

当作业 2 的相应进程在 CPU 上运行时,操作系统负责把该作业的页表在主存中的起始地址 a 送到页表起始地址寄存器中,以便快速找到该作业的页表。

当作业 2 的程序执行到指令"mov r_1,[2500]"时,CPU 给出的操作数地址为 2 500,首先由分页机构自动地把它分为两部分,得到页号 $p=2$,页内相对位移 $w=452$。然后,根据页表始址寄存器指示的页表始地址,以页号为索引,找到第 2 页所对应的块号为 7。最后,将块号 7 和页内位移量 w 拼接在一起,就形成了访问主存的物理地址 7 620。这正是所取的数 123 所在主存的实际位置。

由上述地址变换过程可知,在分页系统环境下,程序经编译、装配连接后形成一个连续

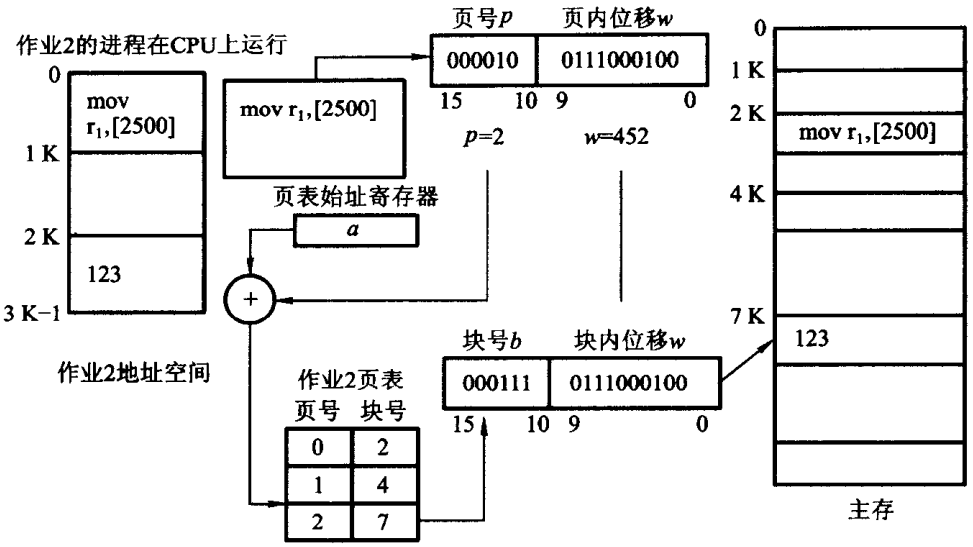


图 3.4.16　页式系统地址变换过程

的地址空间，其地址空间的分页是由系统自动完成的，而地址变换是通过页表自动、连续地进行的，系统的这些功能对用户或程序员来说是透明的。正因为在分页系统中地址变换过程主要是通过页表来实现的，因此，人们称页表为地址变换表，或地址映像表。

4. 请求分页系统的实现技术

（1）实现请调的页表功能

分页系统可分为简单页式系统和请求分页系统两种。在简单页式系统中，应用程序的全部页面都装入主存后，程序才能运行。在这样的系统中，只要实现正确的页式地址变换，程序就能正确执行。

而在请求分页系统中，允许一个应用程序只装入部分页面即可投入运行，那么，进程在运行过程中必然会遇到所需代码或数据不在主存的情况。这样，系统必须解决如下两个问题：

① 怎样发现所访问的页面在不在主存？

② 如确认所要访问的页面不在主存时如何处理？

为解决第一个问题必须扩充页表的功能。图 3.4.13 所示的页表结构只包含页号和块号两个信息，这是进行地址变换所必需的。为了能判断页面在不在主存，可在每个页表表目中，除了登记虚页所在的主存块号外，再增加两个数据项。其一是中断位 i，它用来标识该页是否在主存。若 $i=1$，表示此页不在主存；若 $i=0$，表示该页在主存。其二是该页面在辅存的位置。因此，扩充功能的页表格式应为图 3.4.17 所示。

（2）缺页处理

页号	主存块号	中断位	辅存地址

图 3.4.17 扩充功能的页表

在请求分页系统中,当进程运行时,主存中至少有一块为该进程所对应的程序所占用,并正在执行此块内的某一条指令。若这条指令不涉及访内地址则罢,如果涉及访内地址,则由分页机构得到页号,并以该页号为索引查页表。这时,有两种可能性:当此页对应的页表表目中的中断位 $i=0$ 时,表示此页面已调入主存,可查得块号 b 并形成 $b+w$ 的物理地址,从而使指令得以执行,继而执行下一条指令。当虚地址所在页号的中断位 $i=1$ 时,说明此页不在主存,而是在磁盘上等待着,则情况就比较复杂了。这时首先要把这一页调入主存,安置在某一块中,才谈得上逻辑地址的再定位。相应的步骤是,当发现 $i=1$ 时,发生缺页中断,请求调入此页。当缺页中断发生时,用户程序被中断,控制转到操作系统的调页程序,由调页程序把所需的页面从磁盘(由页表提供盘区地址)调入主存的某块中,并把页表中该页面登记项中的中断位 i 由 1 改为 0,填入实际块号,随后继续执行被中断的程序。

这一页面是根据请求而装入的,因此,这种页式系统称为请求分页系统。一般,当作业最初被调度投入运行时,通常是将相应进程的前几页装入主存,而所需的其他各页,将按请求顺序地装入。这样就可以不必装入不需要的信息,使主存的利用率进一步提高。

(3) 页面淘汰

在请求分页系统中,当所需要的页面不在主存时,需要调入所需的页,若此时,该应用程序所分得的主存块已用完,则需要淘汰已在主存中的一页。哪些页面可以被淘汰掉,这就是置换算法的问题。为了给置换页面提供依据,页表中还必须包含关于页面的使用情况的信息,并增设专门的硬件和软件来考查和更新这些信息。这说明页表的功能还必须进一步扩充。

为实现淘汰页面的功能,在页表中增加“引用位”和“改变位”。“引用位”是用来指示某页最近被访问过没有:为“0”表示没有被访问过;为“1”表示已被访问过。“改变位”是表示某页是否被修改过:为“1”表示已被修改过;为 0 表示未被修改过。这一信息是为了在淘汰一页时决定是否需要写回辅存而设置的。因此,这种情况下页表的一个表目通常在逻辑上至少应包括如图 3.4.18 所示的各数据项。

页号	主存块号	中断位	改变位	引用位	辅存地址

图 3.4.18 完整的页表结构

(4) 置换算法

置换算法是用来选择淘汰哪一页的规则。常用的置换算法有:先进先出淘汰算法(FIFO 算法)和最久未使用淘汰算法(LRU 算法)。

① 先进先出淘汰算法(FIFO 算法)

先进先出淘汰算法的实质是,总是选择在主存中居留时间最长(即最老)的一页淘汰。即先进入主存的页,先退出主存。其理由是:最早调入主存的页,其不再被使用的可能性比最近调入主存的可能性大。这种算法实现起来比较简单,只要系统保留一张次序表即可。该次序表记录了作业程序的各页面进入主存的先后次序。

② 最久未使用淘汰算法(LRU 算法)

最久未使用淘汰算法的实质是,当需要置换一页时,选择最长时间未被使用的那一页淘汰掉。LRU 算法基于这种理论:如果某一页被访问了,它很可能马上还要被访问;相反,如果它很长时间以来未曾用过的话,看起来在最近的未来是不大需要的。实现真正的 LRU 算法是比较麻烦的,它必须登记每个页面上次访问以来所经历的时间,当需要置换一页时,选择时间最长的一页淘汰。

由于最久未使用淘汰算法实现起来比较麻烦,且代价太高,实际得到推广的仅是一种简单而有效的 LRU 近似算法。在该算法中,用一个标志位——"引用位"来表示某块中的页面是否被访问,当页面被访问时,其对应的引用位为 1,而未被访问过的页面,其相应的引用位为 0。因此,可以根据引用位的状态来判断各个页面最近使用的情况。当需要置换一页时,选择引用位为 0 的页淘汰。

(5) 抖动

在请求分页系统页面置换算法中,关于淘汰一页的选择问题,很难给出一个通用的算法。这个问题既与整个存储分配有关,又与当前各并发进程的状态和特点有关。然而,置换算法又是相当重要的。如果选择的淘汰算法不好,将会使程序执行过程中请求调页的频率大大增加。甚至可能会出现这样的现象:刚被淘汰出去的页,过后不久又要访问它,因而又要把它调入,而调入后不久又再次淘汰,再访问再调入,如此反复,使得整个系统的页面置换非常频繁,以致大部分的机器时间花费在页面的调度上,只有一小部分时间用于程序的实际运行,从而降低了整个系统的效率。甚至,会导致整个计算机系统的总崩溃,通常把这种情况叫做"抖动"或颠簸(thrashing)。

简单地说,导致系统效率急剧下降的主存和辅存之间的频繁页面置换现象称为"抖动"。"抖动"现象花费了系统大量的开销,但收效甚微。因此,各种置换算法应考虑尽量减少和排除抖动现象的出现。

5. 段式系统

(1) 程序的二维地址结构

在前述的分区存储管理和页式系统中,程序的地址空间是一维线性的,因为指令地址或操作数地址只要给出一个信息量即可决定。而在实际的应用中,应用程序一般由若干个程

序段组成。对于包含许多子程序的大型应用程序来说,需要有面对程序段结构的存储管理技术。为此,提出了段式系统。

在段式系统中,作业的地址空间由若干个逻辑分段组成,每个分段有自己的名字,对于一个分段而言,它是一个连续的地址区。所谓分段,是程序中自然划分的一组逻辑意义完整的信息集合,它是用户在编程时决定的。

例如,在段式系统中,某应用程序由代码分段、数据分段、栈段组成。图 3.4.19 给出了一个具有段式二维地址结构的应用程序地址空间。

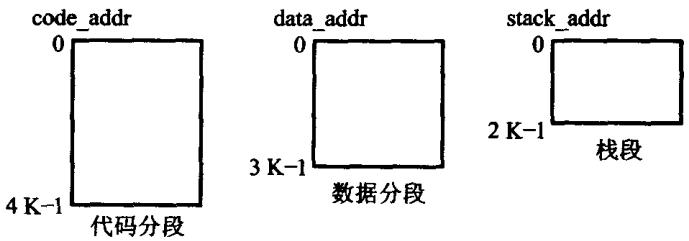


图 3.4.19 分段地址空间

程序的地址空间按段组织后,每段有个标识符,称为段名,由程序员给出。程序经过编译连接后,段名转换成段号。各段的长度一般情况下不相等,但段内编址是一维的。因此程序的地址空间由段号和段内地址组成,即二维地址空间。每个分段的长度是不相等的,系统只规定段的最大长度。例如系统规定段的最大长度为 64 KB,允许的最大段号为 16,那么程序地址字长为 20 位,其中 16 位用来表示段内地址,4 位用来表示段号,其段式地址结构由图 3.4.20 所示。

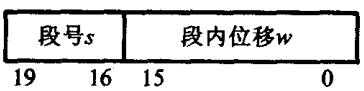


图 3.4.20 段式地址结构

(2) 段式地址变换

段式地址变换所用数据结构为段表。段表由段号、段长、状态位(标志该段是否在主存)、段首址、段保护信息等字段组成。系统为每个作业建立一张段表,每张段表由若干个表目组成。每一个表目描述一个分段的信息。段表和分段映像存储的情况由图 3.4.21 说明。

段式地址变换的步骤如下:

① 取出程序地址,它由段号 s 和段内位移 w 组成。

② 用 s 检索段表。

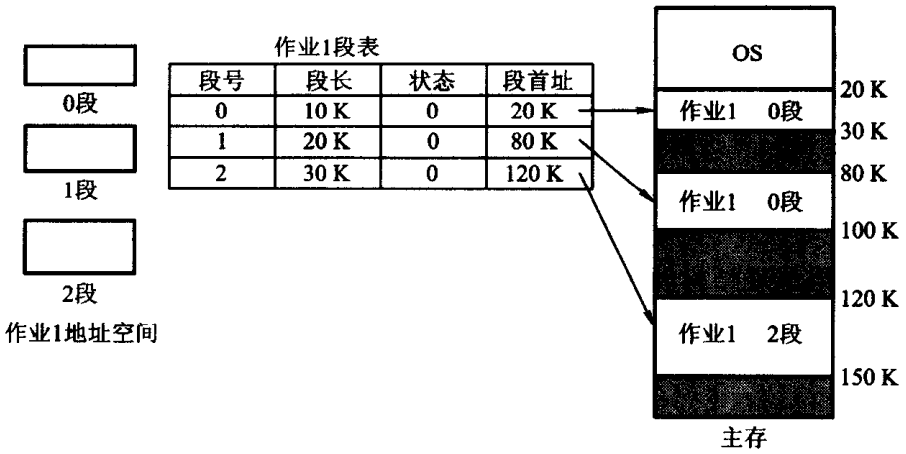


图 3.4.21　分段映像存储

③ 如 $w<0$ 或 $w\geqslant l$，则表示段越界，发生越界中断。

④ 若 $0\leqslant w<l$，则由 $b+w$ 形成所需主存地址。

段式系统的地址变换过程与页式系统类似，却有本质的区别。首先，两者的地址结构是完全不同的，页式系统是一维地址结构，段式系统是二维地址结构。其次，页式地址变换和段式地址变换有以下区别：

① 段式系统中的分段是用户地址空间的逻辑划分，而页式系统中的页面是用户地址空间的物理分割（等分）。

② 页的大小相等，且是固定的，在系统设计时确定，一经确定不会改变；而分段的长度可以不等，它由用户编程时决定（分段最大允许的长度由 w 字段的位数决定）。

③ 将程序地址分成页号 p 和页内位移 w 是分页机构自动完成的，w 字段的溢出将自动加入到页号中去；而程序地址分成段号 s 和段内位移 w 是逻辑功能，w 字段的溢出并不加到段号中去，而是表示越界，产生越界中断。

6. 段页式系统

段式存储管理技术与分页存储管理技术相结合就形成了段页式系统。应用程序由若干分段组成，而在一个分段内又划分页面。图 3.4.22 给出了一个具有段页式地址结构的用户地址空间。

段页式存储管理的用户地址空间是二维的、按段划分的。在段中再划分成若干大小相等的页。这样，地址结构就由段号、段内页号和页内位移 3 部分组成。用户使用的仍是段号和段内相对地址，由地址变换机构自动将段内相对地址的高几位解释为段内页号，将剩余的低位解释为页内相对地址。这样，用户地址空间的最小单位不是段，而是页，因此，主存按页的大小划分，按页装入，这样，一个段可以装入到若干个不连续的页内，段的大小也不再受主

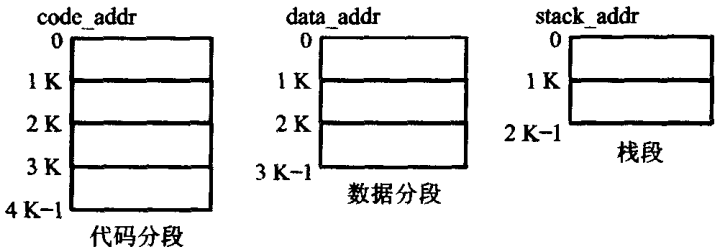


图 3.4.22 段页式地址空间

存可用区的限制了。

用于地址变换的数据结构是:每一个作业一张段表,每个段又建立一张页表,段表中的地址是页表的起始地址,而页表中的地址则为主存块号。它们的相互关系如图 3.4.23 所示。

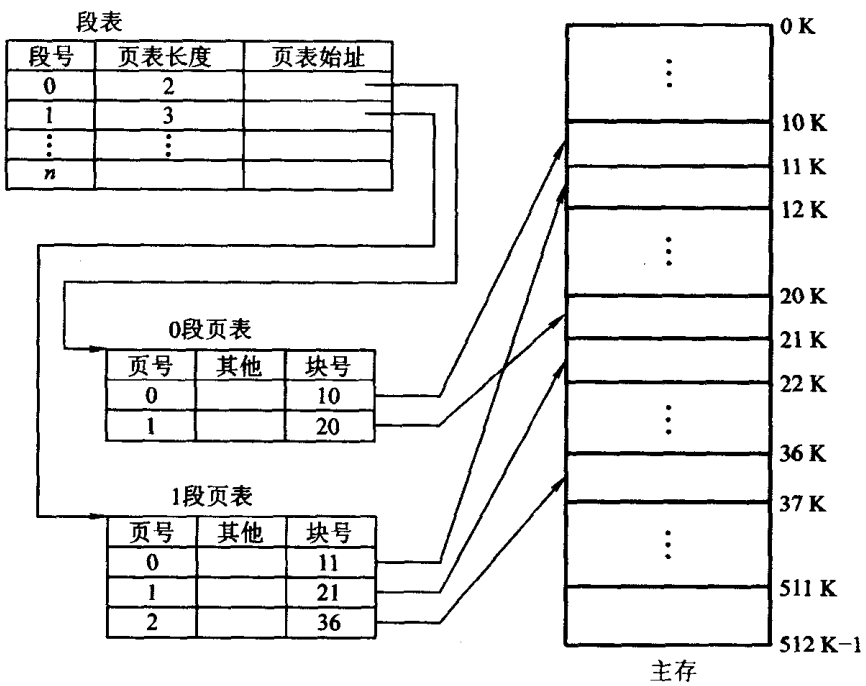


图 3.4.23 段页式管理中的段表、页表与主存的关系

3.4.4 设备管理

1. 设备管理功能

输入输出管理(简称 I/O 管理)是指对数据传输控制和对计算机系统中除中央处理机、

主存储器之外的所有其他设备的管理。

(1) 设备类型

外部设备一般分为两类:一类是存储设备,另一类是I/O设备。存储设备是计算机用来存储信息的设备,如磁盘、光盘、磁带等。I/O设备包括输入设备和输出设备两类。输入设备是将计算机外部世界传送来的信息输入给计算机。例如键盘、光字符阅读机、电传输入机、数字化仪、模数转换器等。输出设备是计算机用来"控制"外部世界的设备。它将计算机加工好的信息输出给外部世界。输出设备有宽行打印机、激光打印机、数模转换器、绘图仪等。此外,还有各种通信设备负责计算机之间的信息传输,如Modem、网卡等。有的设备既可作为输入设备,也可作为输出设备,如电传打字机。设备还可以按传输的信息特点来分类,如有些设备上的信息是以字符为单位组织的,这样的设备称为字符设备;如果设备上的信息是以块为单位组织的,则称为块设备。外部设备的经济价值在整个计算机系统中占有相当大的比重,比如一个具有磁盘、光盘、激光打印机、终端和磁带的微型计算机系统,外部设备的价值占整个系统价值的60%左右。所以,操作系统设计的第一位目标是应以最有效的方法使用这些设备。

(2) 设备管理功能

设备管理的功能包括以下几个方面:

① 监视系统中所有设备的状态

计算机系统中存在着许多设备,在系统运行期间这些设备都在处理各自所承担的工作,并处于各种不同的状态,系统要有效地管理和使用这些设备,就必须监视它们的工作状态。系统为每个设备设置设备控制块DCB结构,在DCB中登记了设备的状态信息,系统是通过对DCB中这些信息的查询和修改来监视设备的状态、控制设备的活动。

② 设备分配

I/O管理的功能之一是设备分配。系统将设备分配给进程(或作业),使用完毕时系统将其及时收回,以备重新分配。设备分配和回收可以在进程级,或者在作业级进行。在作业级,当作业进入系统时分配其所需的资源(包括设备),退出系统时收回全部资源,这称为静态分配。在进程级,当进程需要使用某设备而提出申请时进行分配,使用完毕后立即将其收回,这种分配一般称为动态分配。

③ 设备控制

每个设备都响应带有相应参数的特定的I/O指令。I/O管理的设备控制模块负责将用户的I/O请求转换为设备能识别的I/O指令,并实施设备驱动和中断处理的工作。即在设备处理程序中发出驱动某设备工作的I/O指令,并在设备发出完成或出错中断信号时进行相应的中断处理。

2. 设备独立性与设备控制块

(1) 设备独立性

设备独立性的概念是将用户(程序员)所使用的设备与机器中实际进行 I/O 操作的物理设备分离开来。用户(程序员)不用关心系统具体配置了哪些设备,也不需了解各种设备的使用方法及其特性,只要按照自己的习惯为传输工作中所需的设备起个逻辑名字,然后由操作系统根据当时设备的使用情况和用户所要求的操作,为用户的逻辑设备指派一个具体的物理设备,再由 I/O 控制程序完成所需的操作。这样既方便用户使用,又可对设备进行动态分配、提高设备利用率。

当系统提供"设备独立性"的能力时,它为用户提供设备的逻辑名。所谓设备独立性是指:用户在编程时所使用的设备与实际使用的设备无关。也就是在用户程序中仅使用逻辑设备名。

有两种类型的设备独立性:

① 独立于同类设备的具体设备号。如果一个系统中有若干台(套)相同的设备,则不论使用哪台具体的设备均可。这种意义下的设备独立性使用户(或进程)不依赖于特定的设备的完好和空闲与否。对系统而言,则能合理地分配和利用各种设备。

② 独立于设备类型。如果程序要求输入信息,不论从什么设备上输入均可,对输出也一样。例如 MS - DOS 系统中,程序的 I/O 操作不必指出在哪台设备上进行,一般情况下从键盘上输入信息,由显示器输出显示信息,但用户如果想把输出信息打印出来,只要做一次打印机的联机操作命令 Ctrl + P,输出信息就被打印出来。

在具有这种独立于设备类型的操作系统的环境中,用户不致于因缺少某种设备而不能工作,同时也方便了用户。

(2) 设备控制块

为了实现以统一的方法来处理所有的设备,应将与设备本身紧密相关的设备特性分离出来,为此,为每一个设备构造一个数据结构。记录设备的硬件特性、连接、使用情况以及登记设备各处理程序入口地址等信息的数据结构,称为设备控制块 DCB。当设备装入系统时,DCB 被创建。DCB 的基本内容如表 3.4.1 所示。

表 3.4.1 设备控制块 DCB

设备标识符
设备属性
指向命令转换表的指针
在 I/O 总线上的设备地址
设备状态
当前用户进程指针
I/O 请求队列指针

① 设备标识符。设备标识符即设备名,它是设备的系统名,或称设备的物理名。

② 设备属性。设备属性是描述设备现行状态的一组属性。特别是慢速字符设备,不同类型的设备工作特性常常不同,比如,终端设备的特性主要有传输速度、图形字符集等。

③ 指向命令转换表的指针。与该设备有关的设备处理程序的入口地址组成一张表,称为命令转换表。该表的首地址就存放在此数据项中。

④ 设备 I/O 总线地址。设备与 CPU 之间是通过 I/O 总线连接起来的,该设备在 I/O 总线上的地址,称为设备 I/O 总线地址。

⑤ 设备状态。指设备当时所处的状态。

⑥ 当前用户进程指针。该指针指向正在使用该设备的用户进程的 PCB 结构。

⑦ I/O 请求队列指针。等待使用该设备的 I/O 请求块组成等待队列,该指针指向 I/O 请求队列中的第一个 I/O 请求块。

使用设备控制块的目的之一是为 I/O 管理提供一个不变的界面,因为,每个 I/O 请求都要转换成调用一个能执行 I/O 操作的设备处理程序。为此,操作系统构造了一个命令转换表,该转换表包含设备特定的 I/O 处理程序入口地址。对于某类设备而言,不具备某个操作功能,则在其 I/O 处理程序入口地址为“-1”。通过查找 DCB 中命令转换表指针这一数据项,可以找到所需要的设备处理程序入口地址。

3. 缓冲技术

计算机系统中,各种设备(包括 CPU)的运行速度差异很大,CPU 的运行速度是以微秒甚至以毫微秒计,而设备的运行速度则是以毫秒甚至以秒计。在外部设备中,慢速字符设备(如打印机)与块设备(如磁盘)的速度又相差很大。而且系统的负荷也不均匀,有时处理机进行大量的计算工作,I/O 操作较少;有时又会进行大量的 I/O 操作,这两个极端都会造成系统中的一些设备过于繁忙,一部分设备过于空闲,严重地影响 CPU 与外设的并行操作。为了进一步解决 CPU 和 I/O 设备间速度不匹配的矛盾,引入了缓冲技术。

缓冲是两种不同速度的设备之间传输信息时平滑传输过程的常用手段。

实现缓冲的方法有两种:缓冲器和软件缓冲。缓冲器是以硬件的方法来实现缓冲的,它容量较小,是用来暂时存放数据的一种存储装置。从经济上考虑,除了在关键的地方采用少量必要的硬件缓冲器之外,大都采用软件缓冲。软件缓冲区是指在 I/O 操作期间用来临时存放 I/O 数据的一块存储区域。缓冲技术是有效地利用中央处理机和各种外部设备的重要技术。

下面讨论缓冲是如何工作的。当用户要求在某个设备上进行读操作时,首先向系统申请一个空闲缓冲区,然后从该设备读入数据到缓冲区中。当用户要求使用这些数据时,系统将从缓冲区中提取数据,并发送到用户的进程存储区中。当缓冲区空而进程又要从中取用数据时,该进程被迫等待。此时,操作系统需要重新送数据填满缓冲区,进程才能从中取数据继续运行。当用户要求写操作时,首先向系统申请一个空闲缓冲区,然后将数据从用户进

程存储区传送到缓冲区中。当缓冲区写满或某一适当的时机，由操作系统将缓冲区的内容写到物理设备上，该缓冲区变为空闲。只有在系统还来不及腾空缓冲区之前，进程又企图输出信息时，它才需要等待。

常用的缓冲技术有3种：双缓冲技术、环形缓冲技术和缓冲池。双缓冲是缓冲管理中最简单的一种方案，常用于字符设备（如显示器）的输入/输出中。而缓冲池则常用于块设备（如磁盘）的输入/输出中，UNIX系统的缓冲池管理是十分有效和成功的。

4. 设备分配技术

常用的设备分配技术有3种，它们是独享分配、共享分配和虚拟分配。

（1）独享分配

系统设备中有些外部设备，如阅读机、行式打印机、磁带机、绘图仪等具有独享分配的使用特点，往往让一个作业独占使用更为方便。因为这些设备有的在使用前需人工干预，有的可能要执行费时的I/O操作，如把磁带定位到所需的数据位置上。如果把这些设备交叉地分配给各作业使用，则操作员和外部设备都要为更换工作状态而花费较大的工作量。另外还有些设备，如行式打印机，若不加管理让几个用户共同使用，就会出现交叉输出、十分混乱的情况。

所谓独占设备，是指那些适宜于作业独占使用的设备。独占设备往往采用独享分配方式，或称静态分配方式。即在一个作业执行前，将它所要使用的这类设备分配给它；当它结束撤离时，才将分配给它的这类设备收回。静态分配方式实现简单，但采用这种分配方式时外部设备利用率不高。

（2）共享分配

外部设备中，如磁盘等直接存取设备，都能进行快速的直接存取。它们往往不是让一个作业独占，而是被多个作业、进程共同使用，或者说，这类设备是共享设备。所谓共享设备是指那些适宜于多作业、多进程的共享设备。对磁盘这类设备的共享有两个方面：一是共享磁盘的存储空间；另一方面是共享磁盘驱动器。用户对磁盘存储空间的共享使用，一般以文件方式将自己的信息存放在共享设备上。因此，通过文件系统可以按文件名来存取存储在共享设备上的信息。当进程在执行中以显式的文件读写命令提出传输要求时，则要求对磁盘驱动器进行动态分配。文件系统接收到进程的读写请求时，由文件管理作相应处理，转化为对设备的驱动要求。对进程提出的I/O请求形成I/O请求块，并按一定原则加入设备等待队列，当设备空闲时完成这一请求。

（3）虚拟分配

系统拥有的物理设备就是上述的独占设备和共享设备两类。由于独占设备只适合于静态分配方式，而这种分配方式严重地影响了整个计算机系统的效率。从另一个角度来说，独占设备一般是低速的，若采用联机操作，也会增加作业的运行时间，影响计算机系统的效率。为提高计算机系统的效率，提出了在高速共享设备上模拟低速设备功能的技术，称为虚拟设

备技术。

虚拟设备技术的实现过程是,把从独占设备输入(或输出)的信息,先复制到辅存的一个特定的存储区域中,当进程需要从独占型输入设备上读入信息时,就把这一要求转换成从辅存中读入的请求,并从辅存中读入。输出时,先把要输出的信息存入辅存,在适当的时候(如当一个作业执行完毕,或在外存中存储了一个逻辑意义完整的信息集合时),再从辅存中复制出,并通过独占型的输出设备输出。这样,就可以使输入设备和输出设备连续不断地工作。由于一台设备可以和辅存中的若干个存储区域相对应,所以在形式上就好像把一台输入设备(或输出设备)变成了许多虚拟的输入设备(或输出设备)。也就是说,把一台不能共享的独占型设备转换成了一台可共享的设备。

通常把用来代替独占型设备的那部分外存空间(包括有关的控制表格)称为虚拟设备。所谓虚拟技术,是在一类物理设备上模拟另一类物理设备的技术,是将独占设备转化为共享设备的技术。这种分配方法把独占设备改造成为共享设备,使得每一个用户感到好像拥有各类独占设备一样。在这种情况下,称操作系统给用户提供了虚拟设备。这样做改造了设备特性,提高了设备的利用率,有利于资源的动态分配。

5. I/O 控制

设备管理的功能除了监视系统设备状态、进行设备分配外,还有一个重要的功能是控制设备的物理 I/O 操作,实现用户的传输请求。

(1) 用户进程的输入输出请求

应用程序通过设备管理模块提供的系统调用来请求传输服务。这一命令形式既要使用户使用方便,又能确切地描述每次 I/O 传输所需的信息。一个典型的进程请求可通过下述通用形式的系统调用来实现。

doio (ldev,mode, amount,addr)

其中 ldev 指出执行 I/O 的逻辑设备名;mode 指出要求何种操作,例如是数据传输还是磁带反绕,必要时也可指出使用的是哪一种字符码;amount 指出传送数据的数目;addr 指出进行 I/O 的地址,对于数据输入而言,此项为传送的目的地 (准备存放数据的主存地址),对于数据输出而言,此项为传送的源(存放着准备输出的数据的主存地址)。

(2) I/O 控制功能

I/O 系统使进程能与外部设备(如磁带、终端、打印机)及网络进行通信,而控制设备I/O工作的核心模块常称为设备驱动程序。

I/O 控制的功能主要有 3 个方面:① 解释用户的 I/O 系统调用命令;② 设备驱动;③ 中断处理。

当用户发出请求 I/O 传输的系统调用命令时,设备管理模块中有一个 I/O 接口程序处理这一命令。它的功能是把逻辑设备映射为相应的物理设备,检查提供给它的参数的正确性,形成 I/O 请求块,接着由设备驱动程序启动设备进行 I/O 操作。当操作完成或出现错误

时,产生中断信号,由设备中断处理程序进行相应的处理。

3.4.5 文件系统

应用程序在计算机的整个处理过程中,都会涉及信息的加工、存储和管理问题。在早期的计算机系统中,用户想要将信息存放到辅存上,或从辅存读出信息都是一项相当复杂、极为琐碎的工作,它不仅要按照辅存设备的物理地址去安排信息的存放位置,组织相应的 I/O 指令,而且还要确切记住信息在存储介质上的分布情况。如果稍一疏忽,就会破坏已保存的信息,造成无法挽回的严重后果。尤其是在多道程序出现之后,用户想自己去协调、管理那些可为多个用户所共享的磁鼓、磁盘等大量存储介质上的信息,实际上是不可能的,也是不允许的。

人们寻求新的技术解决这些问题。对用户来说,关心的不是信息的具体存放位置,而是存取方法的方便可靠,不是信息的物理结构,而是信息的逻辑结构,因此,操作系统提供文件系统(或称信息管理)功能,它是操作系统的重要组成部分。在这一节中将讨论文件和文件系统的概念,文件系统的功能和实现技术。

1. 文件和文件系统

(1) 文件

① 文件的定义

文件系统是通过把它所管理的信息(程序和数据)组织成一个个文件的方式来实现其管理的。文件是在逻辑上具有完整意义的信息集合,它有一个名字以供标识,文件名是以字母开头的字母数字串。文件是由文件系统存储和加工的逻辑部件。

组成文件的最基本单位是一个字节或一个字符,称为信息项。如用户编制的源文件就是由字符流组成的。

一组相关的字符称作一个域。例如:"123"是数字域,"TEST"是字母域。而记录是由一组相关的域组成的。例如,一个学生记录可以包含学号、姓名、各主修课程的成绩、累计平均分数等独立的域。

构成文件的基本单位可以是信息项(单个字符或字节),也可以是记录。这样,又可以提出关于文件的两个定义:

(a) 文件是具有符号名的信息(数据)项的集合。

(b) 文件是具有符号名的记录的集合。

② 文件的分类

文件按其性质和用途大致可以分为 3 类:

(a) 系统文件——有关操作系统及其他系统程序的信息组成的文件。这类文件对用户不直接开放,只能通过操作系统调用,为用户服务。

(b) 程序库文件——由标准子程序及常用的应用程序组成的文件。这类文件允许用户

调用,但不允许用户修改。

(c) 用户文件——由用户委托给系统保存的文件。如源程序、目标程序、原始数据、计算结果等组成的文件。

为了安全可靠,每个文件可被规定保护级别。文件按保护级别可分为4类:

(a) 执行文件——用户可将文件当作程序执行,但既不能阅读,也不能修改。

(b) 只读文件——允许文件所有者或授权者读出或执行,但不准写入。

(c) 读写文件——限定文件所有者或授权者可以读写,但禁止未核准的用户读写。

(d) 不保护文件——所有用户都可以存取。

按文件流向,它又可以分以下3类:

(a) 输入文件——例如通过阅读机或从键盘输入的文件,只能读入,所以它们是输入文件。

(b) 输出文件——例如从打印机、绘图仪上输出的文件,只能写出,所以它们是输出文件。

(c) 输入输出文件——在磁盘、磁鼓、磁带上的文件,既可读又可写,称为输入输出文件。

根据文件的存取方法或文件的物理结构,还可以对它们作出多种分类。

③ 文件属性

文件可通过一组确定它的类型、操作特性和存取保护的属性来识别。文件的属性一般存放在文件的目录项中。例如在MS-DOS系统中,文件属性占目录项的一个字节,在这个字节中,01表示文件仅读,02表示隐含文件等。

(2) 文件系统

文件系统是操作系统中负责管理和存取文件信息的软件机构,它由管理文件所需的数据结构(如目录表、文件控制块、存储分配表)和相应的管理软件,以及访问文件的一组操作所组成。文件系统负责文件的创立、撤销、读写、修改、复制和存取控制等,并管理存放文件的各种资源。

从用户角度看,文件系统主要是实现了“按名存取”。就是说,当用户要求系统保存一个已命名的文件时,文件系统根据一定的格式把文件存放到文件存储器中适当的地方;当用户要使用文件时,系统根据给出的文件名,能够从文件存储器中找到所要的文件,或文件中某一个记录。因此,文件系统的用户(包括操作系统本身及一般用户),只要知道他们的文件名就可以存取文件中的信息,而无需知道这些文件究竟存放在什么地方。

引入了文件和文件系统的概念后,用户可用统一的文件的观点看待存放在各种存储介质上的信息,并以文件名来使用文件。因此文件系统也可视为用户与外存的接口。文件系统对所管理的文件是一种中性反映,它既不清楚文件信息的逻辑关系,也无力对文件中的信息进行任何解释,仅知道是一组信息的集合。指出这一点是为了使用户明确文件系统与数

据库系统的区别。

2. 文件的逻辑结构和存取方法

对于文件的组织形式,可以用两种不同的观点去研究,这就是所谓的用户观点和实现观点。用户观点主要研究用户思维中的抽象文件,为用户提供一种逻辑结构清晰、使用简便的逻辑文件。用户将按这种形式去存取、检索和加工文件。而实现观点主要研究驻留在存储介质上的实际文件结构——物理文件,研究的重点是提高设备的利用率和工作性能。文件系统的重要作用之一就是在用户的逻辑文件和相应设备的物理文件之间建立映像关系,实现二者之间的相互转换。

(1) 文件的逻辑结构

文件的逻辑结构可分为两种形式:一种是有结构的文件——记录式文件,另一种是无结构的文件——流式文件。

① 记录式文件

记录式文件是一种有结构的文件。这种文件在逻辑上总是被看成一组连续顺序的记录的集合。每个记录由彼此相关的域构成。记录可以按顺序编号为记录0、记录1、……、记录n。如果文件中所有记录的长度都相同,则这种文件为定长记录文件。定长记录文件的长度可由记录个数决定。如果记录长度不等,则称为变长记录文件,其文件长度为各记录长度之和。

② 流式文件

无结构的流式文件是相关的有序字符的集合。文件长度即为所含字符数。流式文件不分成记录,而是直接由一连串信息组成。对流式文件而言,它是按信息的个数或以特殊字符为界进行存取的。

对于慢速字符设备传输的信息,或者源程序,或者由装配程序产生的中间代码等,采用流式文件是一种最便利的文件组织形式。

(2) 文件的存取方法

文件的逻辑结构还必须用存取方法进一步描述。文件的存取方法是由文件的性质和用户使用文件的情况决定的。根据存取的次序划分,存取方法通常可以分为两大类,即顺序存取和直接存取(又称随机存取)。

① 顺序存取

顺序存取是指后一次存取总是在前一次存取的基础上进行,所以,在进行文件存取时,只需给出存取模式,而不必给出具体的存取位置。当打开文件时,文件的存取指针指向第一个信息单位,如第一个字节或第一个记录,每存取一个信息单位,存取指针加1后,指向下一个信息单位,如此类推。

② 随机存取

随机存取,也称为直接存取,以任意次序、随机存取信息的方式,称为随机存取。如存取

文件中的某个记录。这种存取方法，在请求对某个记录进行存取时，要指出其起始存取位置（例如记录号、字符序号）。

对于磁带文件，一般采用顺序存取方法；而对于磁盘、磁鼓上的文件，既可采用顺序存取，也可采用随机存取。

3. 文件的物理结构

文件的物理结构涉及文件在辅存上的安排。文件结构表示了一个文件在辅存上的安置、链接和编目的方法。它和文件的存取方法以及辅存设备的特性等都有密切的关系。因此，在确定一个文件的结构时，必须考虑到文件的大小、记录是否定长、访问的频繁程度和存取方法等。

文件的物理结构分为连续文件、串联文件和索引文件 3 种。与此相关的文件存取方法有顺序存取和随机存取两种。一般来说，对磁带上的文件采用连续文件结构，其存取方法为顺序存取；对磁盘上的文件可采用串联文件或索引文件结构，可直接存取也可顺序存取。

（1）连续文件

一个逻辑文件的信息存放在辅存的一片连续编号的物理块中，这样的结构称为连续结构，或称连续文件。连续文件结构如图 3.4.24 所示，图中表示一个连续文件 A，它由 3 个记录组成，这些记录被分配到物理块号为 100、101、102 的相邻物理块中，这里假定文件的逻辑记录和物理块的大小相等（也可以是一个物理块包括几个逻辑记录或一个逻辑记录占有几个物理块）。

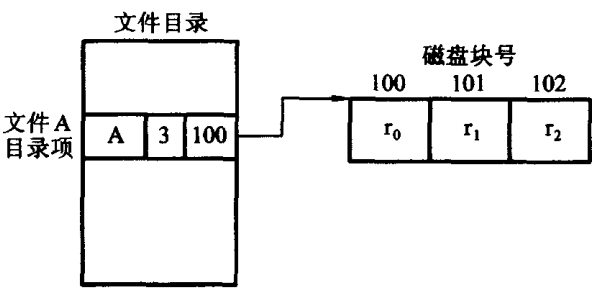


图 3.4.24　连续文件结构

连续文件结构的基本优点是：在顺序存取时速度较快。如果文件中第 n 个记录刚被存取过，而下一个要存取的是第 $n+1$ 个记录，则这个存取操作将会很快完成。对于连续文件结构来说，其文件长度一经固定便不易改变，因而不利于文件的增生和扩充。

存于磁带上的文件一般均采用连续结构，这是因为，在顺序存取设备上，顺序存取时速度较快。而对于磁盘、磁鼓这样的直接存取设备，可以采用连续结构，也可采用非连续结构（后者更为好些）。对于顺序处理的情况，顺序文件结构是一种最经济的结构方式。连续文

件结构对于变化少、可以作为一个整体处理的大量数据段较为方便，而对那些变化频繁的少量记录则不宜采用。

（2）串联文件

一个串联文件结构是按顺序由串联的块组成的，即文件的信息存放于辅存的若干块中，每个物理块有一个链接字，它指出后继块的物理地址，文件的最后一块的链接字为结束标记“∧”，它表示文件至本块结束。串联文件结构如图 3.4.25 所示，图中一个文件 A 有 3 个记录，分别分配到 100、150、57 号物理块中，它的第一个物理块号由文件目录中的该文件目录项指出。

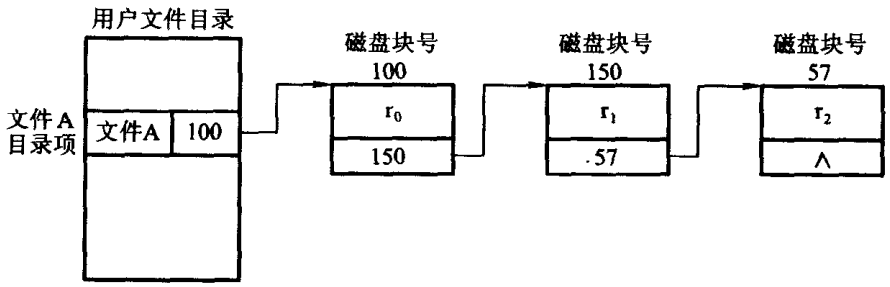


图 3.4.25 串联文件结构

串联文件采用的是一种非连续的存储结构，文件的逻辑记录可以存放到不连续的物理块中，能较好地利用辅存空间。另外，还易于对文件作扩充，即只要修改链接字，就可将记录插入到文件中间或从文件中删除若干记录。对于串联文件而言，要实现随机存取将是十分麻烦的事，因为要找到这个记录，必须从文件头开始一块一块查找，直到所需的记录被找到。

（3）索引文件

为了充分利用辅存空间、易于对文件进行扩充，同时又能随机地访问文件的任何一部分，这就构造了索引文件。索引文件由数据文件和索引表构成，索引表是逻辑块号与物理块号的对照表。若为记录式文件，文件是由记录组成的；若为无结构的流式文件，则可将流式文件顺序地划分成长度相同的逻辑块，即转化为记录式文件。然后为每个文件分别建立索引表。索引文件的索引项按文件逻辑块号顺序排列。例如，某文件 A 有 4 个逻辑块，分别存放在物理块 23、19、26、29 中，该文件的索引结构如图 3.4.26 所示。

索引文件在存储区中占两个区：索引区和数据区。索引区存放索引表，数据区存放数据文件本身。访问索引文件需要两步操作。第一步是查文件索引，由逻辑块号查得物理块号，再由此物理块号获得所要求的信息。这样做需两次访问文件存储器。如果文件索引表已经预先调入主存，则只要一次访问就行了。

索引文件的优点是可以直接读写任意记录，而且便于文件的增删。当增加或删除记录时，应对索引表及时加以修改。由于每次存取都涉及索引表的查找，因此，所采用的查找策

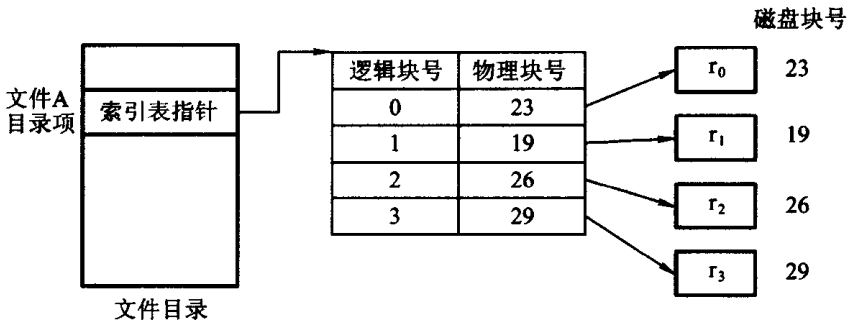


图 3.4.26 索引文件结构

略对文件系统的效率有很大的影响。通常使用的查找策略有两种:二分查找和顺序扫描。

4. 文件目录

(1) 什么是文件目录

文件系统是用户和外部设备之间的接口和界面,用户可通过文件系统去管理和使用存储在各种设备介质上的文件。文件系统所要解决的核心问题是把信息的逻辑结构映射成存储介质上的物理结构,把用户的文件操作转换成对辅存设备的 I/O 指令。转换过程所使用的主要数据结构是文件目录和辅存空间使用情况表。所以,文件目录就将每个文件的符号名和它们在辅存空间的物理地址与有关文件情况的说明信息联系起来了。

文件目录是一张记录所有文件的名字、存放地址以及有关文件说明和控制信息的数据结构。文件目录由许多表目组成,每一个文件占用一个表目,称为文件的目录项。

(2) 文件目录项的内容

文件目录项中的信息一般包括:

① 文件名。由用户赋予文件的标识符。

② 文件逻辑结构。说明该文件的记录是否定长、记录长度及记录个数等。

③ 文件在辅存中的物理位置。记录文件的物理结构信息。文件的物理结构可能是连续文件、串联文件或索引文件结构形式。若为连续文件,此项指出文件第一块的物理地址、文件所占块数;若为串联文件,此项指出该文件第一块的物理地址,但以后各块则由块中的链指针指示;若为索引文件,此项指出索引表地址。

④ 存取控制信息。此项登记文件主人具有的存取权限、核准其他用户及其相应的存取权限。

⑤ 管理信息。如文件建立日期、时间,上一次存取时间、要求文件保留的时间等。

⑥ 文件类型。指明文件的类型,例如可分为数据文件、目录文件、块存储设备文件、字符设备文件等。

(3) 文件目录结构

文件目录是文件系统中重要的数据结构。由于系统中有大量的文件，所以文件目录项的数量也是大量的。这些文件目录项如何组织，就是文件目录的结构。

在操作系统的发展过程中，文件目录的结构从一级文件目录、二级目录结构，一直发展到目前广泛使用的树型目录结构，或称为多级目录结构。

在多用户环境中，应允许用户根据自己的习惯和需要为文件命名，即具备命名的灵活性。一级文件目录不能解决命名冲突问题，即不允许重名。而在二级目录结构下，命名也不够灵活，所以现在广泛使用的是多级文件目录结构。所谓“重名”，是指不同用户对不同文件起了相同的名字，即两个或多个文件只有一个相同的符号名。例如，两个程序员为各自的测试程序命名为“test”。一个灵活的文件系统应该允许文件重名，而又能正确地区分它们。

(4) 树型目录结构

树型目录结构(或称多级目录结构)由根目录和各级目录组成，除根目录外，其他各级目录均以文件的形式组成目录文件，根目录中的每个目录项可以对应一个目录文件，也可以对应一个数据文件，同样目录文件中的每个目录项可以对应一个目录文件，也可以对应一个数据文件。如此类推，就形成多级目录结构。在有些文献中也称树形目录结构。

图 3.4.27 给出了一个树型目录结构。在树型目录结构中，根目录称为根结点，各级目录文件称中间结点，用方框表示。数据文件称为叶结点，用圆圈表示。图中，各级目录和数据文件都有一个 id 号，这是文件的内部标识符，以方便系统的管理。子目录 b、……、子目录 e 等都是目录文件，文件 a、文件 h、……、文件 d 都是数据文件。

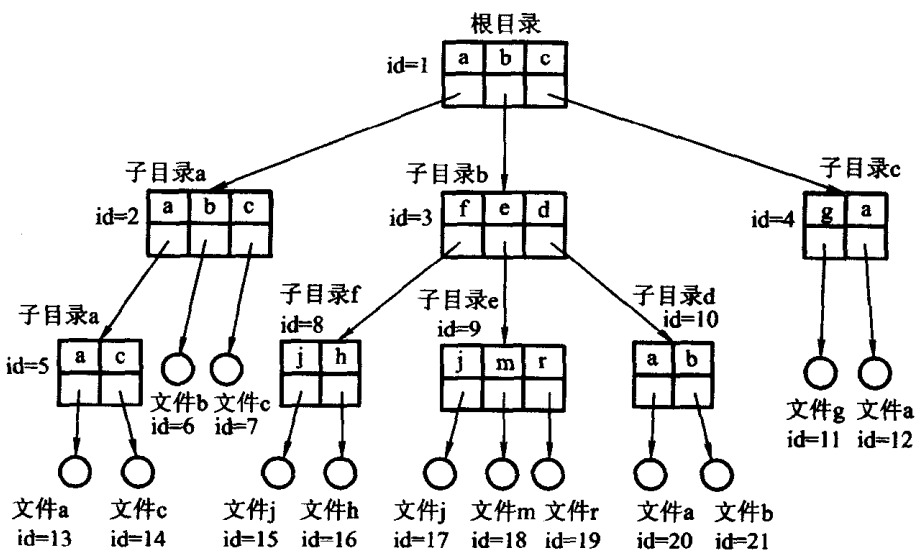


图 3.4.27 树型目录结构

(5) 文件路径名及当前目录

① 文件路径名

在树型目录中,一个文件的路径名是由根目录到该文件的通路上所有目录文件名和该文件的符号名组成的,各符号名之间用分隔符分隔。例如,对于图 3.4.27 中 id 为 15 的文件,其文件路径名为"/b/f/j"。当用户进程使用路径名来存取文件时,文件系统将根据这个路径名的顺序来查访各级目录,从而确定所要文件的位置。

采用多级目录组织后,不同用户可以给不同的文件起相同的名字。例如,id 为 13 的文件和 id 为 20 的文件,它们的文件符号名都为 a,但它们分别属于两个不同的子目录,其路径名分别为"/a/a/a"和"/b/d/a",故能解决这一命名冲突问题。而且,树型目录结构能使命名更灵活一些,它允许一个用户给其不同的文件取相同的名字,只要它们不在同一分目录中即可,如 id 为 15 的文件和 id 为 17 的文件,它们的符号名字都为 j。并且,由于它们的路径名分别为"/b/f/j"和"/b/e/j",故也不会造成混乱。

② 工作目录

在树型目录结构中,文件路径名一般较长,而用户总是局部地使用文件,为了方便用户使用,可建立工作目录。所谓工作目录,是指用户当前经常使用的文件所在的目录,又称为当前目录或值班目录。建立了工作目录后,文件路径名由工作目录到信息文件的通路上各级目录的符号名与该文件的符号名组成,各符号名之间用分隔符分隔。

一旦建立了工作目录,如无特殊说明,系统总是从当前目录开始查询。如在图 3.4.27 中,若指定当前目录为 id 为 3 的子目录 b,则 id = 15 的文件的路径名为"f/j"。若工作目录为根目录,则文件路径名用分隔符"/"开始。在进行文件查询时,若路径名以"/"开头,则从根目录开始查找,否则从当前目录开始查找。

5. 文件操作

(1) 文件控制块

进行文件操作的主要数据结构是文件控制块(FCB),它的内容是文件目录项的一部分。在文件不处于活动期间,FCB 的部分信息作为文件目录项的内容存放在相应物理卷的文件目录中。文件处于活动期间,则存在于主存之中。文件控制块的内容包括:

① 文件目录项所包含的全部信息。

② 文件所在设备的设备号。

③ 文件读写指针。

(2) 文件创建与撤销

文件的操作包括文件的创建和撤销、文件的打开和关闭、文件的顺序读写和随机读写、移动文件指针等。在树型目录结构中还有文件目录的有关操作,如工作目录的建立,子目录的建立和撤销等。这些文件操作由系统调用命令提供给用户,下面介绍几种主要的文件操作的功能。

① 文件创建

用户欲将一批信息作为文件保存时，须向系统提出创建文件的请求，并给出创建文件的文件路径名、文件控制等信息。系统创立文件要做以下工作：在用户指定的目录中找到一个空闲的目录项；在主存中申请一个存放 FCB 的存储区；把用户给出的文件名、存取控制信息填写到 FCB 中；把文件所在的设备号填到 FCB 中，并将文件读写指针置为零；向请求者返回文件号。

文件一旦创建，用户可以用系统返回的文件号使用文件。

文件号（或称文件句柄），是系统的一个数据结构——用户打开文件表中的表目序号。在用户打开的文件表中登记了用户所有已创建的文件或已打开文件的 FCB 首址，每个文件占一个表目。用户打开文件表中表目的序号称为文件号。例如一个文件的 FCB 首址登记在用户打开文件表的第 6 个表目中，那么向用户返回的文件号为 6。

② 文件撤销

当一个文件完成了任务后，就应该撤销它。撤销一个文件须向系统提出撤销文件的请求，并给出撤销文件的路径名。系统根据用户给出的撤销文件的路径名，找到相应文件的目录项，回收该文件占用的全部资源，并将该目录项置空。

（3）文件打开与关闭

在树型目录结构中，一个文件要经历多级文件目录，由于文件目录存放在辅存上，如要存取文件时都要到辅存上去查目录，那是颇为费时的。但是，如果把所有目录、在所有时间内都放在主存中，则要占用大量的存储空间，显然这也是不可取的，因为目录数可能很多，表目总数可达成千上万。实际上，在一段时间内使用的文件数总是有限的，因而也仅涉及到少量的目录表目。所以，只需将目录文件中当前正需要使用的那些文件的目录表目复制到主存中。这样既不占用太多的主存空间，又可显著地减少查寻目录的时间。为此，操作系统为用户提供了两种特殊的文件操作：打开文件和关闭文件操作。

① 文件打开

所谓打开文件，就是把该文件的有关目录表目复制到主存中约定的区域，建立用户和这个文件的联系。

若一个文件有关的目录表目已被复制到主存，则称它为已打开的（或活动的）文件。在主存中存放这些目录表目的区域可形成一张活动文件表。当用户访问一个已打开的文件时，系统不用到辅存上去查目录，而只要查找活动文件表就可得到该文件的文件说明。文件一次被打开后，可多次使用，直到关闭或撤销该文件为止。

当用户想使用一个已经存在的文件时，必须向系统提出打开文件的请求，并给出打开文件的路径名、操作类型（读或写）和存取控制信息等。其具体工作为：

（a）根据用户给出的文件名或路径名查询此文件，并作操作合法性检查。

（b）向主存申请存放 FCB 的存储区。

（c）将文件目录项的有关内容、文件所在设备的设备号等信息写到 FCB 中去。

(d) 将文件读写指针置为0。

(e) 向用户返回文件号。

文件打开成功之后,用户可用文件号对文件进行操作。文件打开操作所做的工作和文件的创建工作基本相同,所以创建文件的操作包括了打开文件操作,有的系统就只设了一个系统调用——打开文件,当查不到要打开的文件时就认为是新创建文件。换句话说,对已存在的文件是做打开的工作,对不存在的文件做创建工作。

② 文件关闭

所谓关闭文件,就是用户宣布这个文件当前不再使用,系统将其在主存中的相应目录表目删去,因而也就切断了用户同这个文件的联系。

当文件不再使用时,向系统提出关闭文件的请求,给出要关闭文件的文件号,系统把文件 FCB 的有关内容复制到相应的文件目录项中去,并释放 FCB 所占的主存空间和占用的数据结构,如用户打开文件表的相应表目。

(4) 文件读写

文件打开(或创建)之后,就可对文件进行读写操作了。在一般系统中分两种方式进行,一种是顺序读写,一种是随机读写。

顺序读操作是以 FCB 中读写指针指示的位置读取信息到读缓冲区,并修改读写指针。顺序写操作是把缓冲区的信息写到 FCB 中读写指针所指示的位置中去,同样要修改读写指针。随机读写操作与顺序读写操作的区别是,前者在进行读写操作之前,要请求系统调整 FCB 中读写指针,然后进行读写操作。

用户要读取已打开文件的信息,须向系统提出读文件的请求。读操作的系统调用的参数包括:文件号、操作模式、缓冲区的首址和读取的字节数。系统按 FCB 中读写指针指示的位置,按要求的操作模式(顺序读或随机读),读取规定的字节数到缓冲区,然后修改读写指针,并向用户返回实际读取的字节数。写操作的情况与读类似。

在上述各种文件操作中都可能产生错误,如创建立文件时无空闲目录项;打开文件时,用户给出的文件找不到,读写操作出错等。系统必须处理这些错误信息,并向用户报告。

6. 文件的共享与安全

所谓文件共享,是指某一个或某一部分文件可以让事先规定的某些用户共同使用。为实现文件共享,系统必须提供实现共享的方法,另外,系统还必须提供文件保护的能力,即提供保证文件安全性的措施。

文件的安全问题是由于文件共享而产生。在非共享环境中,唯一允许存取文件的用户是文件主本人。因此,只要对用户作一次身份检查就可确保文件的安全。对于共享文件,文件主需要指定哪些用户可以存取他的文件,哪些用户不能存取。一旦某文件确定为可被其他用户共享时,还必须确定他们存取该文件的权限。例如,可允许他的一些伙伴更新他的文件,而另一些伙伴可以读出这些文件,其他的就只能装入和执行该文件。这就涉及到文件安

全性(即保护)的问题。

文件的保护是指文件本身不得被未经文件主授权的任何用户存取,而对于授权用户也只能在允许的存取权限内使用文件。它涉及到用户对文件的使用权限和对用户权限的验证。所谓存取权限的验证,是指用户在存取文件之前,需要检查用户的存取权限是否符合规定,符合者允许使用,否则拒绝。

(1) 采用"链接技术"实现文件共享

所谓"链接",就是在相应目录表目之间进行链接,即一个目录中的表目直接指向另一个目录的表目,如图 3.4.28 中,/b/d 目录想共享/c 下的 id = 12 的文件 a,建立链接的方法是在子目录 d 中加一个表目,其文件名由共享者决定,如,起名为 k,其地址指向子目录 c 中文件 a 的表目地址。注意,这种链接不是直接指向文件,而是指向相应的目录表目。这种办法也称为连访,被共享的文件称为连访文件。现在假定工作目录为子目录 b(id = 3),则共享者可用路径名 d/k 直接存取 id = 12 的文件 a。

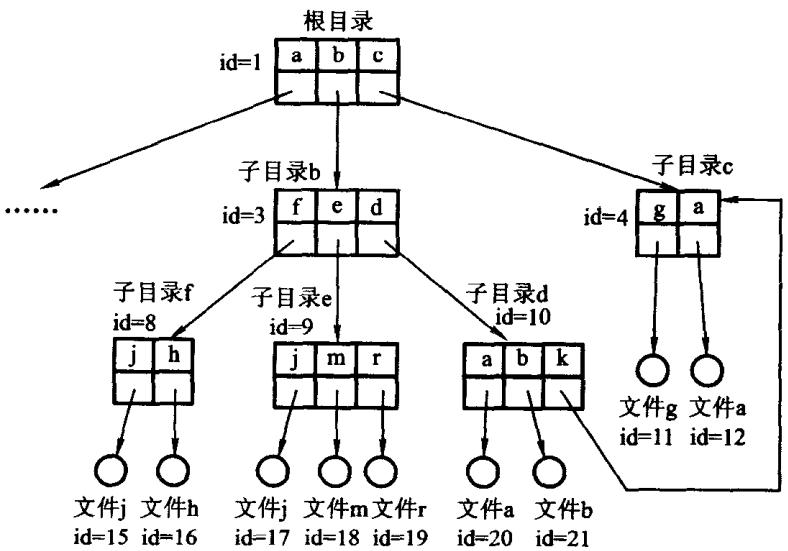


图 3.4.28 目录链接示意图

采用这种链接方法时,在文件说明中必须增加"连访属性"和"用户计数"两项。前者说明表目中的地址是指向文件还是指向共享文件的目录表目。后者说明共享文件的当前用户数目。若要删除一个共享文件时,必须判别是否有共享的用户还要使用,若有,只作减 1 操作,否则才真正删除此共享文件。

(2) 利用"存取控制表"实现文件的保护

为了解决文件的安全问题,必须对文件的存取进行控制。常用的文件保护措施有用户权限表、口令等。下面讨论利用存取控制表对文件进行保护的方法。

采用存取控制表的方法对文件进行保护时，要将所有对某一文件有存取要求的用户进行分类，同时还规定每一组用户对该文件的存取权限。所有用户组对该文件的存取权限的集合就是该文件的存取控制表。

常用的对用户分类模式是：

① 文件主。正常情况下，这是建立文件的用户。

② 指定的用户。文件主所指定的允许使用这一文件的另外用户。

③ 同组或同项目的用户。用户通常是工作在某一特定项目的小组中。在这一情况下，组内的各个成员可以全被赋予对与项目有关的所有文件的互相访问权。

④ 公用。大多数系统允许一个文件被指定为公用的，这样它就可以被该系统的用户集团中的任何成员所访问，公用访问一般只允许用户读或执行一个文件，而写则是被禁止的。

例如，对某一文件有存取要求的用户按某种关系或工程项目的类别分成若干组，而把一般的用户统统归入“其他”用户组。用户的存取权限可以有执行、读、写等。

所有用户组的存取权限的集合就是该文件的存取控制表，如表 3.4.2 所示。一般将这些信息存放在文件的文件目录项中，作为该文件记录的一部分（即存取控制信息部分）。

例如，某文件的用户按下述方式分类：文件主；伙伴；其他用户。

典型的存取权限是：只能执行（E）；只能读（R）；可以写（W）；可改变保护级别（P）；可删除（D）。每种存取权一般包含了它以上的各种特权。例如，存取级别 R 表示可以对文件进行读、执行操作。

系统还可以用一个保护关键字来描述文件的保护信息，保护关键字由三位组成：第一位表示文件主的存取权限；第二位表示伙伴的存取权限；第三位表示其他用户的存取权限。例如，建立文件时，由文件主指定的文件的保护关键字是 DWR，则表示文件主可以做任何一个操作，伙伴能够执行、读和写，而其他用户只能执行和读取。

为了实现上面这种方法，文件主必须向系统申明他的伙伴用户。

表 3.4.2 存取控制表

用户 \ 文件	alpha
文件主	ERWPD
伙伴	ERW
其他	ER

当一个用户向文件系统提出存取请求时，由文件系统中的一个存取控制验证模块查询文件目录项中的存取控制表，将本次请求和存取控制表中用户对该文件的存取权限进行比较，如果匹配，允许访问，否则就拒绝执行。

3.4.6 死锁

为了最大限度地利用计算机系统的资源，操作系统应采用动态分配系统各种资源的策略。然而，采用这种策略时，若分配不当，可能出现进程之间互相等待资源又都不能向前推进的情况，即造成进程相互封锁的危险。事实上，不同进程对资源的申请可能按某种先后次序得到部分满足，这就可能造成其中的两个或几个进程彼此间相互封锁的情况。即每个进程“抓住”一些为其他进程所等待的资源不放，其结果谁也得不到它所申请的全部资源，所有这些进程都无法继续运行。

1. 什么是死锁

首先举一例说明死锁现象。该例说明的是，当进程共享系统的独占资源时，如资源分配不当，可能出现进程互相死等的局面。

设某系统拥有一台打印机和一台光标记阅读机，并为进程 $p1$、$p2$ 所共享。在某时刻 t，进程 $p1$ 和 $p2$ 分别占用了打印机和光标记阅读机。在时刻 $t1(t1>t)$，$p1$ 又申请光标记阅读机，但由于光标记阅读机被 $p2$ 占有，因此 $p1$ 处于等光标记阅读机的状态。而到时刻 $t2(t2>t1)$，$p2$ 又申请打印机，但由于打印机被 $p1$ 占有，因此 $p2$ 处于等打印机的状态。显然，在 $t2$ 以后，$p1$ 和 $p2$ 都无法继续运行下去了。此时，系统出现了僵持局面，也称为出现了死锁现象。

死锁就是两个或多个进程无止境地等候着永远不会成立的条件的一种系统状态。这一描述适合于自然界的各种事物。由于操作系统中的死锁一般是由资源分配不当而引起的，所以它的定义常常这样描述：在两个或多个并发进程中，如果每个进程持有某种资源而又都等待着别的进程释放它或它们所占有的资源，否则就不能向前推进。此时，每个进程都占用了一定的资源但又都不能向前推进，称这一组进程产生了死锁。在这种情况下，计算机虽然处于开机状态，但未做任何有益的工作。

2. 产生死锁的原因和必要条件

（1）产生死锁的原因

并发进程共享系统资源，在竞争资源时可能会产生称为死锁的后果。产生死锁的根本原因是系统能够提供的资源个数比要求该资源的进程数要少。所以，当系统中两个或多个进程没有能力进一步执行时，系统就发生死锁。

资源竞争现象是具有活力的、是需要的，虽然它存在着发生死锁的危险性，但是，竞争并不等于死锁。在并发进程的活动中，存在着诸进程联合推进路线，如果这种联合推进路线，可使系统中所有进程都运行完毕，则称这样的推进顺序是合理的。否则，称为不合理的联合推进路线。

例如，在两个进程共享设备的例子中，当进程 $p1$ 和 $p2$ 分别占用了打印机和光标记阅读机后，又分别申请对方已占用的资源，这就是一种不合理的联合推进路线，在这一推进路线

下,这两个进程都不能运行下去,导致死锁的发生。

由此可知,产生死锁的原因是:① 系统资源不足;② 进程推进顺序非法。

(2) 产生死锁的必要条件

产生死锁的4个必要条件是:

① 互斥条件。涉及的资源是非共享的,即进程对它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,不能被其他进程强行夺走,即只能由获得该资源的进程自己来释放。

③ 部分分配。进程每次申请它所需要的一部分资源。在等待一新资源的同时,进程继续占用已分配到的资源。

④ 环路条件。存在一种进程的循环链,链中的每一个进程已获得的资源同时被链中下一个进程所请求。

3. 死锁的预防

并发进程共享系统资源时,如处理不当可能发生死锁。死锁不仅会在两个进程之间发生,也可能在多个进程之间、甚至在系统全部进程之间发生。此外,死锁不仅在动态使用外设时发生,也可能在动态使用存储区和数据库时发生,或在进程通信过程中以及在利用信号灯作同步工具时,由于 p 操作顺序不当而产生。在早期的操作系统中,系统规模较小,结构简单,而且资源的分配常常采用静态方法,使得操作系统尚未暴露死锁问题的严重性。但是,随着系统规模的增大,软件系统变得异常庞大而复杂,系统资源的种类亦日益增多,因而,死锁的可能性将大大增加。由于死锁的发生会给系统带来严重的后果,因此,处理系统死锁问题引起了人们的普遍注意,并对它进行了深入的研究。

解决死锁问题的策略归纳起来有以下3种。

① 采用静态分配方法来预防死锁。

② 采用有控分配方法来避免死锁。

③ 当死锁发生时检测出死锁,并设法修复。

下面,简单介绍常用的预防死锁和避免死锁的方法。预防死锁的方法可以分为静态预防和动态避免两种措施。静态预防方法中采用资源的静态分配,而动态避免方法中采用资源的动态分配。

(1) 死锁的预防

静态预防方法的基本思想是:每个用户向系统提交作业时,需一次说明他所需要的资源,并且作业调度程序只能在满足该作业所需的全部资源的前提下才能将它投入运行。当资源一旦分配给该作业后,在其整个运行期间这些资源为它独占。

这种方法破坏了产生死锁的必要条件中部分分配这一条,因此采用这种方法进行资源分配绝对不会产生死锁,且实现比较简单。其主要缺点是系统资源利用率很低。

(2) 死锁的避免

为了提高资源利用率,应采用动态分配资源的方法。但是,为了避免可能产生的死锁,在进行资源分配时,应采用某种算法来预测是否有可能会发生死锁,若存在可能性,就拒绝企图获得资源的请求。预防死锁和避免死锁的不同在于,前者所采用的分配策略本身就否定了必要条件之一,这样来保证死锁不可能发生;而后者是在动态分配资源的策略下采用某种算法来预防可能发生的死锁,从而拒绝可能引起死锁的某个资源请求。

动态预防方法中较简单的一种称为有序资源分配法。

有序资源分配法的基本思想是:系统中所有资源都给定一个唯一的号码(例如,光标记阅读机为1,图像输入设备为2,打印机为3,磁带机为4,磁盘为5等),所有分配请求必须以上升的次序给出,释放资源时按序号递减的次序进行。系统要求每个进程:① 对它所必须使用的而且属于某一类的所有资源,必须一次申请完;② 在申请不同类的资源时,必须按各类的编号依次申请。例如,光标记阅读机1,打印机3,磁带机4,是一个合法的申请序列;而光标记阅读机1,磁带机4,打印机3,为不合法的申请序列。当遵守上升次序的规则时,若资源可用,则分配资源的请求就能批准;若资源还不能利用,则请求者就等待。所以,在实施分配时,分配程序必须调用一个算法,该算法就是考查本次申请的序号是否符合资源序号递增的规定,若符合,则在资源可用的情况下予以分配,否则拒绝分配。

3.5 Windows 系统及使用

现代操作系统为用户使用计算机提供了一个方便、高效、功能强大的计算环境,而且是一个多用户、多任务的环境。多个用户提交的多个应用程序可以在一台计算机上同时运行;一个应用程序又可以多个任务(或称进程)的面貌在计算机系统中并发活动。

操作系统为支持多用户、多任务的并发执行,必须管理这些任务在系统中的动态活动、协调它们相互之间的约束关系;解决这些用户、任务对系统资源的申请、使用和释放,即进行有效的系统资源管理,以满足申请者的需要。这是操作系统内核必须解决的两大问题。为了实现方便用户使用的目标,操作系统还应提供方便的用户界面,即提供用户使用操作系统的途径。

操作系统配置在不同类型、不同档次的计算机上。任何应用、任何软件的运行都离不开操作系统。那么,实际的操作系统是如何工作、如何使用呢? Windows 系统以方便、易用、"平民"的面貌走进千家万户,走进人们的生活。UNIX 系统则以"贵族"的身份应用于金融等领域。当用户迫切希望能使用 UNIX 系统、了解它的内核代码与机制,但又难以实现时,Linux 这一具有与 UNIX 相同界面、功能相当的自由软件以其强有力的生命力向人们走来,并且迅速地成长。现在,Linux 系统已占领了广大的阵地,并有超过 Windows 系统的趋势。

本节和3.6节将分别讨论广泛使用的 Windows 和 Linux 系统的基本情况和使用方法。

3.5.1 Windows 系统的发展

目前,计算机应用在各行各业广泛而深入地开展,各种计算机应用不胜枚举。计算机应用之所以能如此迅速地推广,各种媒体的信息能够方便、快捷地处理、加工、流通,都得益于计算机技术、计算机网络与多媒体等技术的发展和进步。其中具有图形化界面的操作环境起到了很大的推动作用。因为计算机进入千家万户、各行各业,面对的是具有不同文化程度的人们。怎样才能使计算机不只被那些计算机专家和软件程序员垄断,而让人机对话的界面方便、友好和易学呢?这是对计算机操作系统的一个挑战!因为人们是通过操作系统来使用计算机的。用直观图形表示的操作界面可以跨越不同语言和不同文化的障碍。这种友好、形象的用户界面将给计算机用户展示巨大的魅力。

而 Windows 系统正是一个图形化的多任务操作系统。它提供一个基于图形的多任务、多窗口环境,还具备许多新的功能和特点。它具有统一的图形用户界面,更直观、方便的工作方式,更快速的执行效果,以及更强大的网络功能。

从 1983 年微软公司宣布 Windows 系统诞生到现在的 Windows XP,Windows 已走过了 20 年的历程。

Windows 系统是从图形用户接口起步的。80 年代初出现了商用 GUI 系统(由 Apple Computer 公司推出的 Apple Macintosh)后,微软公司看到了图形用户接口的重要性及其广阔的市场前景,公司内部就制定了发展"界面管理者"的计划,到 1983 年 5 月,微软公司将这一计划命名为 Microsoft Windows。

1983 年 11 月,Windows 系统宣布诞生,1985 年 11 月发布 Windows 1.0 版。1990 年 5 月推出 Windows 3.0,该版本的 Windows 系统对内存管理、图形界面做了较大的改进,使图形界面更加美观,并支持虚拟内存。接着,在 Windows 3.0 版基础上,引入了新的文件管理程序,改进了系统的可靠性,更重要的是,增加了对象链接和嵌入技术(OLE)以及对多媒体技术的支持。

随着计算机硬件技术的发展,微软于 1995 年 8 月推出了 Windows 95。它是一个独立的操作系统,无需 DOS 的支持,采用 32 位处理技术,兼容在此之前开发的 16 位应用程序,在 Windows 发展历史上起到了承前启后的作用。但该系统存在着稳定性较差、对网络支持功能欠缺等问题,微软公司在该系统基础上,扩展了高级的因特网浏览功能、提供 FAT 32 新的文件系统更新版本等,于 1998 年 8 月发布了 Windows 98 系统。

以上讨论的是 Windows 的早期版本和 Windows 9.x 系统的发展过程。随着 Windows 系统的成功,微软公司确定其发展策略是创造一个基于 Windows 的操作系统家族。这一家族能适用于从最小的笔记本到最大的多处理器工作站的多种计算机。与上述 Windows 系统研制平行的,还有一个新的、便于移植、具有 Microsoft 的新技术(NT)操作系统在研制。Microsoft NT 配置为服务器,它作为一个多用户操作系统运转,为网络上众多用户的需求服务。

每一台工作站都支持一个交互式用户及多个远程用户，每个用户或应用程序在存取该系统前需预先登录。

微软于2000年2月推出了Windows 2000。该平台建立于NT技术之上，具有高可靠性，它通过简化系统管理降低了操作耗费，是一种适合于从最小的移动设备到电子商务服务器所需的操作系统。

2001年微软有推出了Windows XP，将消费型操作系统和商用型操作系统融合为统一系统代码的Windows。Windows XP结束了Windows系统两条腿走路的历史，它是第一个既适合于家庭用户，又适合于商业用户使用的新型Windows系统。

3.5.2 Windows 系统的特点

Windows系统是一个具有图形用户界面的多任务（多用户）操作系统，具备许多优点。

（1）图形化、通用的用户界面

Windows系统提供直观、易用、高效的面向对象的图形用户界面。Windows用户界面和开发环境都是面向对象的，用户采用“选择对象——操作对象”的方式进行工作。图形用户界面（GDI）技术为每个用户分配一个窗口，在屏幕上可同时开设多个窗口，构造了一个直观、方便的工作环境。比如，要打开一个文档，首先用鼠标或键盘选择该文档图标，然后打开该文档。这种操作方式模拟了现实世界的行为，易于理解、学习和使用。

Windows系统为所有应用程序提供统一的表现方式。Windows应用程序大多拥有相同或相似的基本外观，都有统一的窗口形式、统一的菜单格式。系统中的所有资源和命令都用图符表示，并以窗口、文件夹、快捷键、滚动条、按钮、对话框等作为用户界面的元素，用鼠标的点击或拖拽功能完成用户希望的操作和控制系统的工作。

对用户而言，只要掌握了Windows应用软件之一，就很容易学会、掌握其他软件，从而降低了用户学习、掌握Windows系统的门槛。

（2）可抢占的多任务（多进程）和多线程技术

Windows系统是一个32位多任务（多进程）系统，它允许在同一时间内同时运行多个应用程序，也允许同时运行一个应用程序的多个副本（程序实例）。系统内多个进程同时执行，实际上是多个进程在轮流地使用CPU时间（即每个进程占用一个时间片）。由于CPU速度高，它能在多个进程间快速切换，使得这多个进程看起来就像在同时执行一样。

Windows系统采用的是可抢占式调度策略（也可表述为抢先式多任务）。该策略是指：当多个任务同时运行时，Windows会根据需要把控制权转给其他应用程序或收回，而不管当前应用程序是否自己释放CPU。这样可以保证最紧迫的任务优先运行，而不致于出现一个应用程序独占系统资源不愿释放CPU时，其他应用程序等待时间过长的现象。

Windows系统还支持多线程技术，它允许一个进程内创建多个执行路径，以增加并行处理能力，加快进程处理的速度。

(3) 容易理解、使用方便的文件系统

Windows 系统采用树型目录结构,以文件夹代表目录。在文件夹中既可以存放文件、文件夹,还可以有应用程序驱动器图标、打印机图标,以及其他对象,用户可以像使用自己的公文包一样来使用文件系统。

Windows 系统还使用长文件名,使文件的命令更为方便,易于理解和记忆,使查找文件也变得更容易。

(4) 即插即用功能

Windows 系统提供"即插即用"功能。所谓即插即用,即 Microsoft 提供的系统自动管理功能。对于具有"即插即用"特性的设备,无需人工进行配置,只要将它插到计算机相应的槽中,打开计算机,Windows 的即插即用子系统便会自动识别声卡、Modem、网卡、打印机以及其他设备。对于新安装的硬件设备,用户不必再设置任何跳线,IRQ 主存地址和 DMA,Windows 系统会自动对该硬件进行设置,并安装必要的驱动程序。Windows 系统对系统底层硬件资源进行自动管理,包括 IRQ、I/O 设备、DMA、通道和主存。一旦系统通电运行,即插即用子系统就可自动管理所有硬件设置并实现其更改。

即插即用功能实现了 Windows 系统的设置方便、配置容易的目标,使用户增加新设置和修改现有设备的配置工作变得非常容易。

当然,并不是所有的外部设备都支持"即插即用",只有专门为 Windows 系统设计的硬件才具有"即插即用"功能。Microsoft 提出的即插即用功能已表现出强大的生命力,它使得增加或修改各种板卡和外部设备较过去快捷、容易得多,它的进一步完善将使系统配置和设备安装变得异常简单和方便。

(5) 网络与通信

Windows 系统具有很强的联网能力,它具有对等式网络支持、局域网互连以及远程连接等能力。它还为客户机连接做了大量的工作,所以它也支持以客户/服务器方式访问其他用户。

Windows 系统提供高级的因特网浏览功能,提供快速浏览网络的方法。支持主要的因特网标准,包括 HTML、Java、ActiveX、JavaScript、Visua Basic Scripting 以及主要的安全标准。Windows 系统支持多种网络协议,如 IPX/SPX、TCP/IP 等。Windows 系统提供个性化的因特网信息分布能力,为在线通信提供了丰富的工具,包括 Out look Express、Microsoft Netmeeting 等。

Windows 系统还支持远程通信,它向用户提供访问各种邮件服务的统一界面,可用来接收和发送电子邮件、传真,并能付带任何类型的文件信息。许多 Windows 应用程序支持邮件系统的功能,用户可以直接在创建文件的应用程序中发送电子邮件。

另外,还有 FAT32 文件系统的改进版本、电源管理以及多种加强功能。

当前,广泛流行的 Windows XP 是同时适合商用和家庭用户的新型 Windows 系统。Windows XP Home Edition(家庭版)是一个易于使用的智能化家用操作系统,具有更丰富的通信

功能、更高的可移动性(笔记本电脑用户可以随时随地访问他们所需要的信息),还提供简捷方便的数码影像功能、对高品质的音频和视频信息有良好的支持。而 Windows XP Professional(专业版)则具有更多良好的特点。由于 Windows XP Professinal 建立在 Windows NT 和 Windows 2000 代码的基础上,采用了 32 位计算机体系结构和完全受保护的内存模型,所以,它成为了一个可靠的操作系统。

3.5.3 Windows 系统的结构

在初步了解了 Windows 系统的基本概念和特点后,下面接着讨论这样一个问题:Windows 系统作为一个完整的操作系统,应具备哪些功能模块,应具备怎样的整个系统的结构。

图 3.5.1 给出了一个抽象的 Windows 系统的体系结构。Windows XP 作为现代操作系统,为了保证系统的安全,通过硬件机制实现了核心态(kernel mode)或称管态和用户态(user mode)或称目态两种特权级别。当系统处于管态时,CPU 可以执行任何指令,可以使用系统所有的资源,包括整个存储区。而当系统处于用户态时,CPU 不能执行特权指令,而且只能访问执行者被允许访问的区域。操作系统的核心代码都运行在核心态下,而用户程序一般都运行在用户态。当用户执行了一条特权指令时,操作系统能借助硬件一个的保护机制剥夺用户程序的控制权,并做出相应的处理。Windows 2000/XP 的体系结构分为核心态和用户态两部分。图中粗线上部的方框代表用户进程,它们运行在各自的私有地址空间中。用户进程有 4 种基本类型:系统支持进程、服务进程、应用程序对应的进程、环境子系统包含的进程。从图中可以看到,这些进程不能直接调用操作系统服务,而是通过子系统动态链接库和系统交互。粗线以下是核心态,包括系统级服务调度进程、核心态可调用接口、系统核心和设备驱动、图形引擎和硬件抽象层。

(1) 用户进程

系统支持进程(System Support Process)包括会话管理(SMSS)、登录管理(WINLOGIN)、本地安全身份验证服务器(LSASS)服务控制器(SERVICES)及其相关的服务进程。

服务进程(Service Process)提供 Windows 2000/XP 的服务。例如,事件日志服务、假脱机(并行联机外部操作)SPOOLER 服务。

环境子系统(Environment Subsystem)向应用程序提供运行环境,即操作系统功能调用服务。它包括较为重要的、且应用广泛的 Win32 环境子系统。用户程序在 Win32 环境子系统的支持下才能请求 Windows 2000/XP 的系统服务。即用户应用程序需要调用系统服务时,必须由该环境子系统提供的子系统动态链接库为中介才能完成。在 Win32 环境子系统中提供 KERNEL32. DLL、USER32. DLL 和 GDI32. DLL 这 3 个子系统动态链接库。用户应用程序使用上述 3 个子系统动态链接库提供的 Win32API 函数(应用程序接口)实现对系统服务的请求。

(2) 执行体

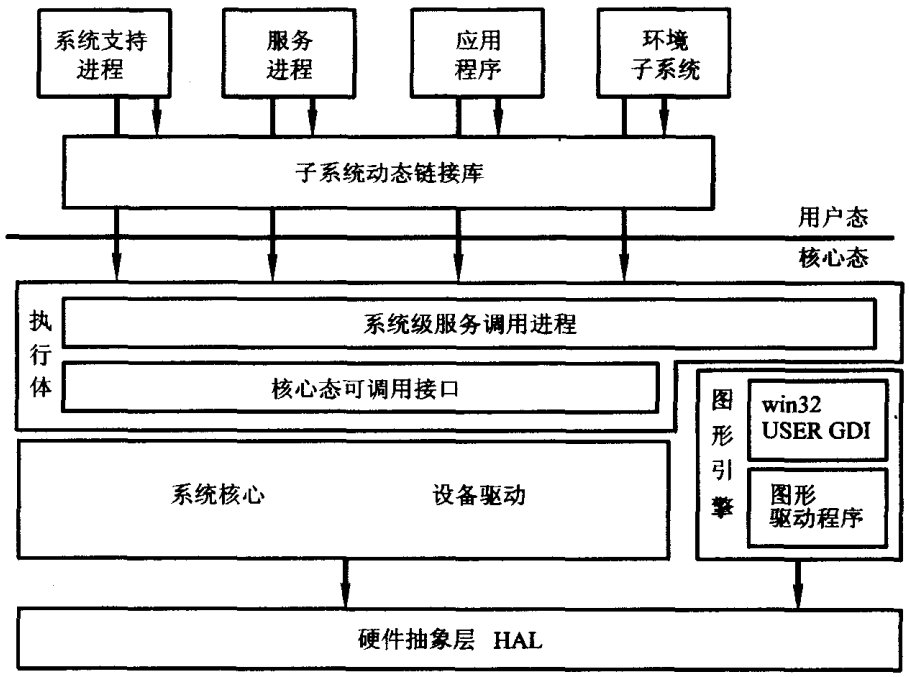


图 3.5.1 Windows 系统的逻辑结构

执行体包含了基本的操作系统服务功能,例如内存管理器、进程和线程管理、安全控制、进程间通信、文件与 I/O 服务。执行体位于核心的上层,提供核心可以调用的函数,通过 Win32API 或其他的环境子系统可以对这些调用函数进行访问。执行体由系统服务调用进程和核心态调用接口组成。

(3) 核心

Windows 2000/XP 的核心包含了最低级的操作系统服务。主要提供以下功能:① 线程调度;② 陷阱处理和异常调度;③ 中断处理和调度;④ 多处理器同步;⑤ 提供执行体所需要的基本内核对象和例程。

Windows 2000/XP 中有下列类型的设备驱动程序:① 硬件(字符)设备驱动程序,它将应用程序的输出写入物理设备或从物理设备、网络中获得输入;② 文件系统驱动程序接受面向文件的请求,系统将它转化为块设备(磁盘等)的 I/O 请求。

图形引擎包含 Win32 USER GDI 和图形驱动程序两部分,包含实现图形用户界面的基本函数。

(4) 硬件抽象层

Windows 2000/XP 设计的一个重要目标是实现多种硬件平台的可移植性,硬件抽象层是使这种可移植性成为可能的关键部分。硬件抽象层将内核、设备驱动程序同硬件分隔开,

包含各种与硬件有关的细节。通过它的工作使 Windows 2000/XP 具有了可移植性的特点。

3.5.4 Windows 系统的图形用户界面

Windows 系统与任何操作系统一样提供两个类型的用户界面(用户接口),一是操作界面,一是程序界面。Windows 提供的操作界面是使用极为方便、直观的图形用户界面;而程序界面则是 API 函数。

1. Windows 系统图形用户界面的特点

Windows 系统提供了多窗口图形用户界面。为每个应用程序提供一个窗口,在屏幕上可以同时开设多个窗口,有的在编辑正文,有的在绘图,有的在进行费时的计算,多个窗口并发执行。利用图形系统提供的各种友好界面,使用户可以方便、有效地操作和控制诸窗口的运行。

Windows 的界面色彩丰富。它提供的调色板功能,使 Windows 环境下的每个应用程序都可以使用尽可能多的颜色,而不影响其他窗口显示的颜色。这使得 Windows 下的应用程序在一般的 VGA 彩色显示器上,可以模拟在真彩显示器上的显示效果。

Windows 系统提供统一的界面风格。无论是 Windows 系统本身,还是各种 Windows 应用程序,都具有一种相似的简明的外观和特征,提供完成各类操作的相应手段。如果用户已经学会了其中一种产品的用法,那么学习同类另一种产品的用法在时间上会大大缩短。例如,如果要在一个 Windows 应用程序中打开一个数据文件,其操作过程总是单击 File 菜单,然后在此菜单中选择 Open 命令。

Windows 系统的用户界面还有一个最本质、最重要的特点,它是以文档为中心的界面。以文档为中心的方法使用户只需要关心他们的文档,而用不着关心处理该文档的应用程序是什么。由系统负责处理特定格式的数据和处理这些数据的应用程序之间的关系。对于一般用户来说,他们关心的只是正在加工的文档。例如,某个文档可以是正在编辑的一本书稿,也可以是本科生操作系统考试成绩分析图表。至于这个书稿是否由 Word 编排,这个图表是否由电子表格应用程序编排,用户是不关心的。当需要修改相应部分时,系统会自动地激活处理该部分的相应应用程序。

Windows 系统采用了 OLE 技术(Object Linking and Embedding):对象的链接和嵌入,并引入了复合文档的概念。有了 OLE 技术,在一个文档中可以包容和编辑多种不同类型的数据。要对文档的某个部分进行编辑,只需要在编辑对象上双击鼠标就可以了。适合于处理这种类型的应用程序会运行并自动装入数据,用户无需进行其他任何操作。用户看到的只是单个文档并已对它进行加工,但在这个过程中可能调用了多个不同的应用程序,而且,这些工作是由系统自动完成的。

2. Windows 系统的窗口

大家在使用 Windows 系统时,总会看到大大小小的许多窗口。下面,讨论什么是窗口、

窗口的组成等有关问题。

(1) 什么是窗口

窗口是任何一个 Windows 应用程序基本的输入和输出形式,是应用程序单独访问的一块显示区。窗口占据系统显示器上的一块矩形区,大小位置均可任意调整。从用户和程序员角度看,窗口无非是显示器上互相重叠(或不重叠)的矩形区域,他们把窗口看作"虚拟"显示器。窗口有自己的坐标系和分辨率(可变),有自己的色彩表、光标和字体,并暂时占有鼠标和键盘。作为支持图形用户界面的 Windows 组件(User)和有关的管理程序,它们负责管理屏幕上的窗口、对话框、按钮和 Windows 界面上其他元素的创建和使用。它们为每个应用程序提供了这种"虚拟"显示器,好像每个应用程序唯一控制自己的窗口,而围绕窗口的其他资源亦独立于相应的窗口。实际上系统的显示屏只有一个,其他资源也是有限的,管理窗口的软件必须为提供这些"虚拟"显示器解决许多问题,如共享冲突问题。

(2) 窗口的组成

组成窗口的基本要素包括:标题栏、控制菜单栏、菜单栏、工作区、工具栏、状态栏、最小化按钮、最大化按钮、滚动条和窗口边框等。

在 Windows 系统中,每个应用程序都使用一个窗口与用户交互,并与其他应用程序共享屏幕。用户可以对目前正在运行的所有应用程序进行控制。但在实际使用中,如果开的窗口太多,相对而言每个窗口就较小,窗口间反复切换也会降低工作效率。所以 Windows 的窗口管理功能允许将任何一个窗口缩小,或缩小到最小,而成为一个图标(它将处于任务栏上)。图标代表着一个备用的窗口,用户想要激活这个窗口时,只要将鼠标移动到图标上,单击鼠标就可以恢复这个窗口。

下面介绍窗口的主要组成部分。

① 标题栏:用于显示当前文档和应用程序的名称。在窗口以图标形式存在的情况下,标题和图标一起标识一个活动的应用程序。

② 控制菜单:在每个窗口标题栏的最左侧有一个小方框,它代表该窗口的总控菜单,用于进行恢复、移动、最小化、最大化和关闭窗口等操作。

③ 菜单栏:在窗口标题栏的下面是该窗口的菜单栏,它列出该应用程序所提供的功能,这些功能经分类后列在各个菜单中。

④ 工作区:占据了窗口的大部分,可以显示窗口的内容,例如,正在运行字处理应用程序,将在该区域显示当前文档的内容。

⑤ 工具栏:工具栏中包含了一系列按钮,用于快速执行菜单中某些常用的命令。根据文件夹和应用程序窗口的不同,工具栏的按钮也有所不同。

⑥ 滚动条:当窗口内容较多时,在窗口的右侧及底部,将出现滚动条,用鼠标单击两端的箭头,可以看窗口中未显示的内容。

⑦ 状态栏:某些窗口,下方显示一个状态栏,用于提示当前计算操作的信息。例如当某

个应用程序载入文件时，在状态栏中将显示文件信息的载入比例。

⑧ 窗口边框：在窗口没有占用整个屏幕，也不是图标的情况下，都有一个边框。用鼠标按住边框的 4 个角，可以对窗口进行 4 个方向的缩放，用鼠标拖动 4 个边框，还可以对窗口进行横向或纵向的缩放。

（3）菜单

Windows 系统提供的菜单功能，是 Windows 应用程序基本的用户输入方式。用户要求某一应用程序提供某一功能时，可以从窗口的菜单栏中选中某一菜单名，这时 Windows 会在该菜单位置下拉（或上推）一个菜单，该菜单列出在这一菜单名下的当前可执行的命令，供用户选择。Windows 负责管理这些菜单信息，并根据用户的选择发送信息给窗口函数，相应的程序就会被激活去执行这条命令。

（4）对话框

当用户选择了菜单中某一个要执行的命令时，这些命令有时还需要再输入一些参数。例如，用户在 Flie（文件）菜单中选择了 Open（打开文件）命令来打开一个文件，这个文件的路径名无法在菜单中选择，这时，Windows 提供一个称为对话框的临时窗口，用户通过它输入命令的所有详细信息。

对话框中通常包含一个或几个控制框，它是一种具有简单输入输出功能的小窗口。可以认为，对话框是应用程序与用户进行交互的具有子窗口的弹出式窗口。

3. Windows 桌面系统

（1）桌面

在启动计算机后，屏幕上显示的是 Windows 桌面。在桌面上有若干个图标，最下一行是任务栏，任务栏的最左侧是启动按钮。常见的图标有计算机形状的“我的电脑”，还有“网上邻居”，“我的公文包”、“回收站”等。“我的电脑”图标为用户提供了访问本地磁盘的服务。用户双击该图标就意味着用户想要“打开”自己的计算机，查看其中的内容。“网上邻居”也是类似的访问部件，用于访问和本地计算机连接的其他计算机系统。

桌面是 Windows 组织管理系统对象的一个平台，文档、文件夹、应用程序等都可以放到桌面上。Windows 桌面就像用户办公室里的办公桌，桌上可以存放各种文件夹，文件框里分门别类地存放各种文件，在办公桌上还可以放一个公文包，放一张照片等。Windows 桌面系统的最大特点是简洁、合理，它使用户能够非常轻松地按照原有的习惯在计算机上清理自己的思路，组织自己的工作，完成各种操作。

桌面是 Windows 中最大的对象，它位于文件系统的最顶层，即它是一个根文件夹（根目录），其他各个文件及应用程序都在它的下面。用户可以在桌面上执行各种操作，如打开一个文件夹，启动某个应用程序或者设置壁纸、设置屏幕保护等。

Windows 的设计者为了让用户方便地启动工作并能随时方便地理清自己的工作，在桌面系统上设计了两个重要的组成部件：任务栏和“开始”按钮。

(2) 任务栏

任务栏位于屏幕的底部,在默认情况下,系统任务栏总是可见而且是可操作的。在某个应用程序以其最大化的窗口运行时,系统任务栏仍然保持不变。这样不仅使用户能随时访问,且可防止没有经验的用户在系统中迷失方向。

任务栏提供了快速方便启动或切换应用程序和文档的手段。"开始"按钮也属于任务栏。"开始"按钮用于迅速访问某些常用的系统功能,如帮助或关闭系统功能。而任务栏其余部分是一块用于放置当前活动窗口的场地。系统将会在任务栏中放置按钮,代表每一个活动的窗口。如果要激活某个对象,只要在任务栏中单击该对象按钮,就会使该对象的窗口移到前台,从而实现各窗口间的切换。使用任务栏切换各个窗口,可以很形象地与电视机电视频道的转换相类比,由此可见它的易用性。当关闭一个应用程序窗口时,其最小化按钮也将从任务栏中消失。

4. 文件夹和快捷操作

(1) 文件夹

Windows 系统将所有的资源,例如文件、应用程序、打印机、与本机相连的其他计算机等,都看作对象。Windows 系统采用了一个形象的逻辑概念——文件夹来进行管理。

文件夹包含应用程序、文档、子文件等对象。其中每个对象用图标表示,图标下方设有该对象的名称。实际上,文件夹就是目录,它是目录的图形化表示。它可以包含文件和子目录,形成树型目录结构。当启动计算机后,显示的 Windows 桌面就是根目录。在桌面上有"我的电脑"、"网上邻居"、"回收站"等图标,它们就是文件夹。

如果将一个文档拖放到某一个文件夹中,实际上是将该文档拷贝到了这一文件夹所代表的目录中。如果在一个目录下创建了一个子目录,实际上是在该目录所代表的文件夹中创建了一个子文件夹。

所以文件夹就是目录,用它来组织文件、应用程序等资源显得非常直观、方便。由于 Windows 系统使用这样一套文件夹机制可容纳各种对象,这些能力是实现以文档为中心的系统所必须具备的。

(2) 快捷方式

Windows 系统提供的快捷操作方式,使用户通过建立对一个对象的链接而不是复制这个对象,来达到引用的目的。当为某个对象建立了快捷方式后,可以很方便地通过它访问该对象。

例如,将一个照片文件 07080050.jpg 放在数据盘 F:\pa 目录下,如在该目录下建立文件 07080050.jpg 的快捷方式,并将图标拖到桌面上,则桌面上就有一个左下角带一个小箭头的指向该文件的图标。这时观察桌面(即根目录)的列表,可看到一个 1 KB 大小的快捷方式文件。07080050.jpg 文件还是在原目录下,在桌面上的只是一个链接文件信息。双击该文件的快捷方式图标时,Windows 系统会通过这个链接文件,找到并打开与之相关联的

07080050.jpg 文件,这时就可以看到照片。又如,用户想打印一个文档文件 my.doc,它可以创建一个文件夹,里面放置 my.doc,再加上一个对打印机的快捷操作,以便用来打印这个文档。要打印时,用户只需简单地打开文件夹,单击这个文件图标,然后把该图标拖放到打印机图标上。这样就完成了对打印机的访问过程。Windows 系统能够自动装入相应的应用程序,并通知这个应用程序对所选择的文件执行打印操作。

快捷方式既可在桌面上创建,也可在文件栏中创建。通过创建快捷方式,用户可以在不同的地方访问一个对象,而不必将该对象拷贝到要用它的每一个位置上,同时也简化了打开对象时所做的多步操作。

3.5.5 Windows 系统的程序界面

Windows 应用程序是在 Windows 环境下运行的程序,它使用 Windows API 函数来编写程序,以完成应用程序的功能。

Windows 操作系统依靠一组用户态环境子系统,作为应用程序与操作系统之间的接口。环境子系统的主要工作是为应用程序提供编程接口和执行环境。Windows 2000/XP 操作系统有 3 个环境子系统,其中最重要的是 Win32 环境子系统。Win32 环境子系统是 Windows 2000/XP 操作系统固有的子系统,它提供应用程序运行所需要的窗口管理、图形设备接口、媒体控制、内存管理等项服务功能。这些功能以函数库的形式组织在一起,这就是 Win32 应用程序编程接口(API),简称 Win32 API。Win32 子系统负责将 API 调用转换成 Windows 的系统功能调用。

对于应用程序开发人员而言,他所看到的 Windows 系统实际上就是 Win32 API,操作系统的其他部分对用户而言是完全透明的。

Win32 API 由 Win32 子系统的 3 个动态链接库实现。

① USER32.DLL:负责处理用户接口,包括键盘和鼠标输入、窗口和菜单管理等。

② GDI32.DLL:负责在图形设备上(包括显示器和打印机)上执行绘画操作。

③ KERNEL32.DLL:操作系统核心功能服务,包括进程与线程控制、内存管理、文件访问等。

除上述模块外,Windows 2000/XP 还提供其他一些 DLL 文件,以支持另外一些功能,如通用控件(COMCTL32.DLL)、公共对话框(COMMDL32.DLL)、用户界面外壳(SHELL32.DLL)、图形引擎(DIBENG.DLL)以及网络(NETAPI32.DLL)等。

3.6 Linux 系统及使用

3.6.1 Linux 系统的发展

Linux 是一个自由的、类 UNIX 的多用户、多任务操作系统，可运行在 Intel 80386 及更高档次的 PC 机、ARMs、DECAlpha、SUN Sparc、M68000、MIPS 和 Power PC 等多种计算机平台上，已成为应用广泛、可靠性高、功能强大的操作系统。

1. Linux 与 UNIX 的关系

UNIX 操作系统是一个交互式多用户分时操作系统。UNIX 系统是由美国电报电话公司(AT&T)下属的 Bell 实验室的两名程序员 K. Thompson 和 D. M. Ritchie 于 1969—1970 年研制出来的。UNIX 问世以来，在全世界范围内迅速推广，不仅成为高档微机、工作站、小型机的主流操作系统，而且早已进入了中、大型计算机领域，成为事实上的标准操作系统。UNIX 系统基本上是商业化的软件产品，其价格昂贵，这在一定程度上影响了 UNIX 的普及。

为了让更多的人能使用 UNIX 系统，体会其强大的功能，许多人开始研制与 UNIX 界面类似的、能自由使用的免费 UNIX。其中，有一位学者 Andrew Tanenbaum 为了方便学生在学习操作系统原理与概念时，能了解实际操作系统的实现技术，开发了一个包含 UNIX 的概念和功能的操作系统，并且编写了全部源代码。这样就可以避免 AT&T 严格的许可证限制而教学生学习 UNIX，这一系统称为 Minix。Minix 不完全具备现代 UNIX 的许多特性，但由于它是一个小巧的教学用的操作系统，且可用于 PC 机，因此受到人们广泛的关注。

为了提供学习所需的功能和特征，1991 年芬兰赫尔辛基大学的学生 Linus B. Torralds 决定在他的 PC 机上扩充和改造 Minix，形成了一个新的，但十分简单的操作系统。Torralds 将他的新的操作系统源代码以 Linux（是 Linus 和 UNIX 的缩写）的名字在 Internet 上供自由使用，1991 年 8 月 Linux 的 0.01 版可以从 Internet 上下载。由于 Linux 开放源代码的本质，促使大量的、来自世界各地的自愿者通过 Internet 实现共同开发。通过互联网和其他途径，任何人都能得到其源代码，都有机会帮助开发和调试 Linux 的内核、链接新的软件、编写文档和帮助新用户。

1994 年 3 月，Linux 1.0 版本发布。该版本包括了操作系统的全部功能，包括 Linux 内核、X 窗口系统和广泛的数百个应用程序包。Linux 1.0 提供了 TCP/IP、SCIP 和打印机支持，并有足够的驱动程序来支持各种各样的设备。从此，Linux 进入大发展阶段，世界上数百万的爱好者开始使用该系统。

Linux 是在 Internet 开放环境中，由遍布世界各地的爱好者共同开发的，但它是一个类 UNIX 的操作系统，从它一开始创建就遵循商业 UNIX 版本的标准。UNIX 在其发展过程中

也出现过活跃的、众多版本的现象，为了便于共享成果和交流，必须规范无序的开发，为此，由电气工程协会(IEEE)制定了一个独立的 UNIX 标准。这一 ANSI UNIX 标准被称为计算机环境的可移植性操作系统界面(POSIX)。这个标准限定了 UNIX 系统应如何运行和操作，对系统调用也作了专门的论述。POSIX 限制所有 UNIX 版本必须依赖的大众标准。现在大部分 UNIX 和流行版都是遵循 POSIX 标准的。Linux 也是遵循 POSIX 标准，即具有与 UNIX 一样的用户界面和操作方法。

2. 关于 GPL 和 GNU

为了摆脱商业软件公司对软件、特别是对操作系统软件的控制，自 1984 年起，Richard Stallman 发起了自由软件运动，建立了自由软件基金会(FSF, Free Software Foundation)。Richard Stallman 在许多人的协作下，制定了通用公共许可证(GPL, General Public License)，并产生了 GNU 项目。GPL 保证任何人有共享和修改自由软件的自由。任何人有权取得、修改和重新发布自由软件的源代码，并且规定在不增加附加费用的条件下，可以得到自由软件的源代码(基本的发布费除外)。这一规定保证了自由软件总的费用是低的。在使用 Internet 的情况下，则是完全免费的。GPL 对推动自由软件的发展起到了重要的作用。

自由软件基金会发起的主要项目是 GNU，其目标是建立可以自由发布和可移植的 UNIX 类操作系统。由 GNU 项目产生的主要软件有：Emace 编辑软件，gcc 编译软件、BASH 命令解释程序和编程语言，此外，还有其他许多操作系统必不可少的工具。同时，GNU 操作系统的许多关键组成部分都置于 GPL 的约束之下。

3. Linux 的发展

在 Linux 的发展过程中，有一个十分重要的决定，就是加入 GNU 并遵循通用公共许可证 GPL。这一决定大大加强了 GNU 和 Linux 的应用，几乎所有应用 GNU 库的软件都移植到 Linux 上，完善并加速提高了 Linux 的实用性。

正是由于 Linux 成为 GPL 的一员，为 Linux 的进一步发展创造了极为有利的条件。多家技术力量雄厚的商业公司也可加入到原来只由业余爱好者参加的这场自由软件运动中，开发出了多种 Linux 发行版本，如 RedHat Linux、Slackware Linux 和 Caldera Linux 等。极大地推动了 Linux 的发展进程。最近两年，Linux 内核以惊人的速度快速发展，从 2.0 版本、2.2版本，很快发展到 2.4 版本，并不断升级，无论是对服务器，还是工作站，Linux 已发展成为最先进的操作系统之一。

3.6.2 Linux 系统的特点

Linux 是一个性能稳定、功能强大的、类 UNIX 操作系统。它与 UNIX 系统兼容，从一开始就遵循 POSIX 标准，具有与 UNIX 相同的用户界面，提供与 UNIX 相似的功能，但它又是一个完全从基础代码开始重写的操作系统。

(1) 符合 POSIX 标准

Linux 符合 POSIX 1003.1 标准。该标准定义了一个最小的 UNIX 操作系统接口。Linux 具有和 UNIX 一样的用户接口,可以运行 UNIX 的程序。为了使 UNIX 具有影响的两个版本 BSD UNIX 和 System V UNIX 的程序能直接在 Linux 上运行(源代码兼容,部分程序在二进制级兼容),Linux 还增加了部分 System V UNIX 和 BSD UNIX 的用户接口,成为一个较完善的 UNIX 开发平台。

(2) 支持多用户多任务

Linux 是一个真正的多用户、多任务操作系统。它支持多个用户同时使用同一台计算机,同时访问系统中的应用程序,同时从相同或不相同的终端上运行一个或多个应用程序。

Linux 支持多任务并发活动,一个用户或应用程序可以建立多个进程。大量进程可以同时在系统中活动。系统提供进程管理、控制、调度的功能,使进程在进程调度程序的控制下分时地占用 CPU 时间,使各自的活动不断地向前推进,这就是 Linux 的多任务性。

(3) 多平台

Linux 是支持硬件平台最多的一种操作系统,它主要在基于 X86、ISA、EISA、PCI 及 VIB 总线的 PC 机上运行,也可以在 Intel 以外的 CPU 上运行。目前,Linux 可以支持的硬件平台有:Alpha、Sparc、Arm、M68K、Mips、Ppc 和 S390。2.4 内核还可以支持 Super-H 和 Mips64 等硬件平台。

(4) 使用灵活的命令程序设计语言 shell

shell 首先是一种命令语言,Linux 提供的所有命令都有对应的实用程序。shell 也是一种程序设计语言,它具有许多高级语言所拥有的控制流能力,如 if、for、while、until、case 语句,以及对字符串变量的赋值、替换、传递参数、命令替换等能力。用户可以利用这些能力,使用 shell 语言写出"shell"程序存入文件。以后用户只要输入相应的文件名就能执行它。这种方法易于系统的扩充。

(5) 通信和网络

Linux 是一个多用户、多任务操作系统,而且 Linux 的联网能力与其内核紧密地结合在一起。Linux 支持 TCP/IP 网络协议,提供 TELNET、FTP、NFS 等服务功能。用户可以通过 Linux 命令完成内部网络信息或文件传输,也可以向网络环境中的其他节点传输文件和程序,系统管理员或其他用户还可以访问其他操作系统。

(6) 支持多种文件系统

Linux 支持多种文件系统。目前支持的文件系统有 EXT2、EXT、HPFS、MS - DOS、NFS、MINIX、UFS、VFAT 等十几种。Linux 缺省的文件系统是 EXT2。EXT2 具有许多特有的功能,比常规的 UNIX 文件系统更加安全。

3.6.3 Linux 系统的组成与内核结构

1. Linux 系统的组成

Linux 操作系统从逻辑上可以分为 4 个层次，从上层到下层依次为：用户进程、系统调用接口、Linux 内核和硬件。这一层次结构如图 3.6.1 所示。

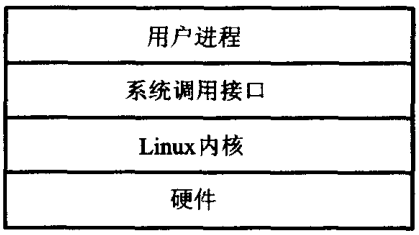


图 3.6.1 Linux 系统的层次结构

（1）用户进程

任何需要 Linux 解决的计算、数据处理或任何其他的应用问题，都需要编制相应的应用程序，并提交给系统。系统将为应用程序建立一个或多个进程。这些进程就是运行在系统最外层的用户进程。这些用户进程是 Linux 操作系统需要服务的对象。

（2）系统调用接口

操作系统能实现有效的系统资源管理，实施对进程的控制和管理。它像服务员一样为用户进程提供软、硬件资源申请、进程创建、进程通信等服务。操作系统提供服务的途径有两个，一为操作接口，另一个为程序接口。Linux 为应用程序提供一组系统调用接口。应用程序可以通过系统调用来请求操作系统内核中特定的功能服务。另外，Linux 还提供 shell（命令外壳程序）作为它的操作接口。

（3）Linux 内核

Linux 操作系统的核心是内核部分。它包括计算机系统资源管理和多任务（进程）并发活动的管理。系统资源管理包括虚拟文件系统、设备驱动、内存管理、网络通信。多任务并发活动的管理包括进程控制、进程通信和进程调度。

（4）硬件

硬件层是 Linux 运行的物质基础，包括主板、CPU、内存、通信设施等应用部件。

2. Linux 系统的内核结构

Linux 核心主要由存储管理、进程管理、文件系统、设备驱动、进程间通信、网络管理等部分组成。其逻辑结构图如图 3.6.2 所示。下面主要讨论进程管理、内存管理和虚拟文件系统等功能。

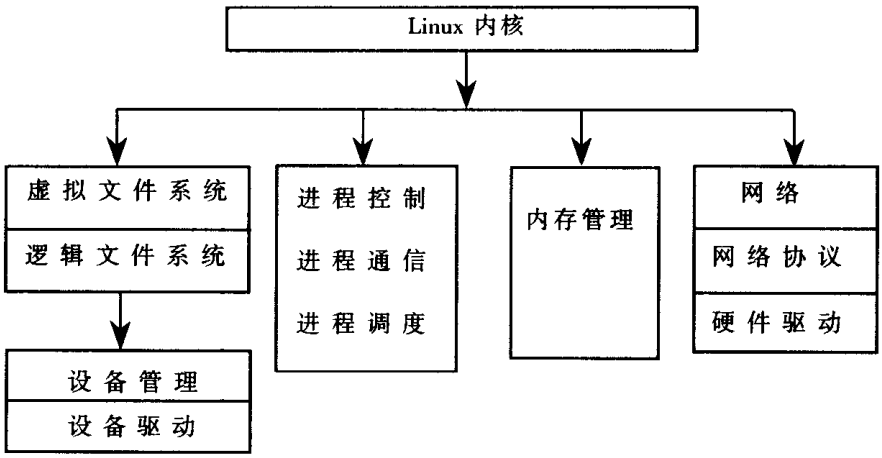


图 3.6.2 Linux 内核逻辑结构

(1) 进程管理

进程控制、进程间通信、进程调度都涉及对并发进程活动实施控制和管理的功能,所以都归为进程管理范畴。

进程控制是对进程活动过程中进程状态变化实施的控制,提供进程创建(fork())、进程撤销(exit())、加载一个特定程序的地址空间(exec())、父子进程同步(wait())等功能。进程通信(IPC)是 Linux 核心提供的进程间协调的功能,包括管道和信号通信机制,还提供 system V 的 IPC 接口和共享内存机制。Linux 系统采用可剥夺的优先调度策略。在系统中,每个进程有一个优先级,进程调度负责决定哪一个就绪进程具有最高优先权,并实施处理机的分派,让选中的进程真正获得 CPU 的控制权。

(2) 内存管理

Linux 采用请求分页虚拟存储管理技术。在 Linux 系统中,每个进程创建时都有自己的虚地址空间,每个虚地址空间都被分为用户段和核心段。每个用户段和核心段又划分为代码段和数据段,在段内划分为页面。在 I386 体系结构中,内存分为 4 KB 大小的物理块。一个页面大小(用户进程的页面大小)也为 4 KB,内存以块为单位接纳用户进程的页面。操作系统为每个用户进程分配一定数量的内存块,装入部分页面,该进程就可以运行,然后,由操作系统自动实现缺页处理、内存和辅存之间的页面交换。为简单起见,称为请求分页虚拟存储管理(将分段的概念简化)。

(3) 虚拟文件系统

虚拟文件系统(VFS)使 Linux 可支持多个不同的文件系统,如前述的 EXT2、MINIX、MS-DOS 等 10 多种文件系统。VFS 的核心工作就是转换。对应用程序提供满足 POSIX 标准的一组固定的文件操作命令:open()、close()、read()、write()等。而对于各种不同的文件

系统则提供接口。这些接口由支持 MS - DOS、MINIX、EXT2 等不同文件系统的翻译程序所使用。通过不同翻译程序的工作,对用户而言,相同的文件操作命令就可以得到不同文件系统的支持。Linux 虚拟文件系统允许用户能透明地安装多个不同的文件系统。

3.6.4 Linux 系统的用户界面

为了方便用户使用计算机,任何操作系统都提供操作界面和程序界面。用户利用操作界面与系统交互、请求系统资源、提交并控制用户程序。而程序界面则为用户程序提供操作系统服务的手段。

Linux 也具备这两个界面,它的操作界面为 shell 外壳。

(1) shell 外壳

shell 是一种命令语言,同时也是一种程序设计语言。作为命令语言,它是 Linux 提供的操作界面。用户利用 shell 命令可以与操作系统通信。当用户登录进入 Linux 之后,Linux 命令解释程序就管理和控制用户的操作,从规定的标准输入设备(键盘)中读取命令,解释执行一条一条的命令,并在标准的输出设备(显示器)上显示命令执行的结果。用户也可以利用 shell 命令重新修改输入和输出设备,例如可以利用重定向命令,将标准输入设备(键盘)改为一个文件,即从规定的文件中输入信息。

shell 又是一种程序设计语言,用户可以利用这种语言编制自己的 shell 过程。这一 shell 过程是一个文件,也是一个命令程序,给这个文件起一个名字后,可以成为一个新的 shell 命令。当系统运行这个 shell 过程时,实际上是由 shell 解释程序一条一条解释文件中的命令,并执行之。shell 作为程序设计语言,具有控制语句、循环语句、参数传递、变量等内容,其功能十分强大。所以,用 shell 语言来编写 shell 过程是十分方便的。

(2) Linux 的系统调用

Linux 与 UNIX 一样提供丰富的系统调用,这是应用程序与 Linux 内核提供的各种服务之间的接口。Linux 系统调用大致可以分为 3 类,包括与进程管理有关的系统调用,与文件和外设管理有关的系统调用以及与系统状态有关的系统调用。下面列举部分系统调用。

① 有关进程管理的系统调用

fork	建立一个进程
exec	执行一个文件
wait	等待子进程
exit	进程中止
sleep	等待一段时间
kill	发送软中断

② 与文件和外设管理有关的系统调用

create	创建文件并打开文件

open	打开文件
close	关闭文件
read	读文件
write	写文件
chmod	改变文件属性
chown	改变文件主人和用户组

③ 与系统状态有关的系统调用

getuid	取用户号
setuid	设置用户号
getgid	取用户组号
setgid	设置用户组号
time	取日历时间
stime	设置日历时间
times	取进程执行时间
gtty	读当前终端 tty 部分信息

3.6.5 Linux 系统的使用基础

1. 进入和退出系统

使用 Linux 就像进入一个图书馆,作为读者必须获得阅览证才能进入图书馆,然后根据借阅规则与方法,借阅各种图书,最后离开图书馆时归还阅览证。对 Linux 系统也是一样。作为用户,首先应从系统管理员那里取得登录名和口令,使用 Linux 时,必须经过登录进入系统,然后使用 Linux 提供的各种命令开展工作,工作结束后通过注销退出系统。

(1) 登录

当用户要进入系统时,Linux 将发出下列登录提示符,欢迎用户登录到系统中来:

login:

在这里输入用户登录名,按 Enter 键。若设置了口令,系统将发出下列提示符请用户输入口令:

password:

在输入口令后,再按 Enter 键。在屏幕上不显示作为口令输入的字符。如果正确地输入了登录名和口令,系统将送回提示符,处于命令接收状态。如果在登录过程中出现了错误,系统将显示登录错误信息:

login incorrect

并发出新的登录提示符请用户重试。直到输入了正确的用户登录名和口令后,才能进入系统使用。

(2) 退出

当用户完成了自己的工作后,就应退出注册,从系统中退出的命令为 logout,或者使用 exit 命令,如:

logout

按 Enter 键后即可从系统退出。当屏幕上重新出现注册提示符"login:"时,表明用户已从系统退出了。

退出注册是说明拥有登录时所用登录名的这个用户现在不使用系统了。由于 Linux 是一个多用户操作系统,对于一个终端用户而言,如果要关闭终端,应先从系统中退出。如果在系统中直接关闭终端电源,那么下次打开终端时它仍在系统中(假定整个系统没有关闭)。这样,别人就可以方便地进入系统,继续使用上一个登录用户的账号。所以,用户应养成先退出系统再关闭终端的习惯。

(3) 关机

如果没有用户在使用 Linux 系统,可关闭系统,但不能直接关闭电源。必须按正常顺序关机。可以使用 halt 或 shutdown 命令关闭系统。

值得一提的是,若 root 超级用户注册进入系统时,系统提示符为"#";若一般用户注册进入系统时,系统提示符为"$"。

2. 文件系统的层次结构

在登录时,有一个特殊的目录和用户的登录名联系在一起,这就是用户起始目录 home。用户起始目录就是用户进入系统时的目录,是用户最初的工作目录,也是该用户整个目录树的起点。该用户在系统中工作时所建立的所有文件和目录都在这一起始目录下。

当用户登录后,使用 pwd 命令,就可以确定用户起始目录。例如,显示名为 pa 的用户登录进入系统,并确定用户起始目录的情况:

```
Rad Hat Linux release 9(shrike)
Kernel 2.4.20-8 on an i686
Localhost login:pa
Password:
[pa@ locahost pa] $ pwd
/home/pa
[pa@ locahost pa] $
```

为了考察 Linux 文件系统的层次结构,可以用 cd 命令改变所在的目录位置,用 ls 命令查看当前目录下的文件和目录名称的清单。现用 cd 命令从用户起始命令进入到 Linux 的根目录。

```
[pa@ locahost pa] $ cd /
[pa@ locahost /] $
```

对于一个典型的 Linux 系统而言，在根目录下使用 ls 命令可以得到如下结果：

```
[pa@ locahost /] $ ls
bin     dev     home     lib          misc     opt     root     tftpboot     usr
boot    etc     initvd   lost + fount mut      proc    sbin     tmp          var
```

Linux 目录层次结构中最重要的部分如图 3.6.3 所示。

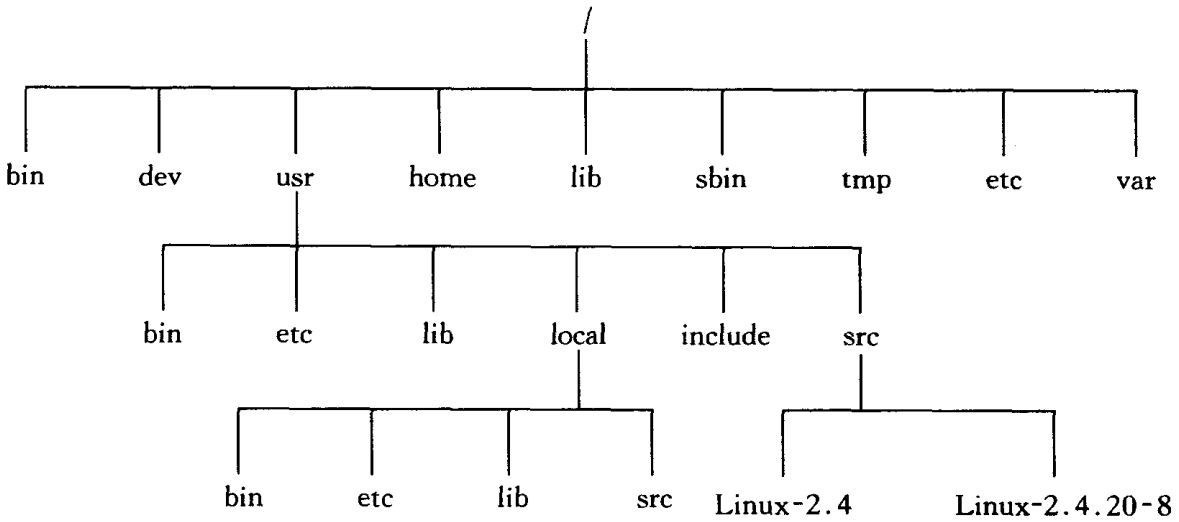


图 3.6.3　Linux 目录层次结构

图 3.6.3 中所示的 Linux 目录树中，各主要分支包含的文件的一般功能如表 3.6.1 所示。

表 3.6.1　主要文件的功能

目录	功能	目录	功能
/bin	二进制可执行命令	/var	某些大文件的溢出区
/dev	设备特殊文件	/usr/bin	其他的可执行命令
/etc	系统管理和配置文件	/usr/lib	库和软件包的配置文件
/home	用户起始目录的基点	/usr/local/bin	本地增加的命令
/lib	标准程序设计库	/usr/local/lib	本地增加的库
/sbin	系统管理命令	/usr/local/src	本地命令的源文件
/tmp	公用的临时文件存储点	/usr/src/linux	Linux 内核源程序文件

3. 常用的文件操作命令

(1) 目录管理

① 显示工作目录——pwd

工作目录就是当前用户所在的目录。用户当前的工作、所发的命令都是以当前目录为基础进行的。在系统提示符下,输入"pwd"命令,就可显示用户当前所在的工作目录。

② 列出目录下的文件——ls

ls 命令用于列出当前目录下的文件清单。若命令不带选项,则以字母顺序列出目录内容。若命令带选项,则以用户选择的方式显示。此命令常用的选项有以下几个。

-a:列出目录中所有文件,包括以"."开头的隐含文件。

-l:输出文件的详细信息。

-t:以最后修改时间为序列目录。

-f:横向列出文件和目录名,并在目录文件后加"/"。

例:[pa@ locahost home] $ ls -l

drwx------ 4 pa pa 4096 lOA 17 22: 32 pa

[pa@ locahost home] $ cd pa

[pa@ locahost pa] $ ls -l 所列信息的意义说明如下

```
-rw-rw-r--    1    pa    pa    24    LOA 18 11: 48    test
    ↓         ↓    ↘     ↘     ↘         ↓              ↘
存取权限  链接数 所属 组名 字节数 最后修改时间 文件名
```

③ 建立目录——mkdir

在工作目录下,创建新目录的命令为 mkdir。在 pa 目录下要创建一个名为 sub 的子目录,用如下命令:

[pa@ locahost pa] $ mkdir sub

[pa@ locahost pa] $ ls

sub test

④ 删除目录——rmdir

rmdir 命令可用来删除一个空目录。如果要删除的目录中还有文件或子目录,rmdir 命令将不起作用。

⑤ 改变工作目录——cd

用户可以利用 cd 命令从一个目录移到所需要的目录位置。要转入指定目录,必须在 cd 命令后面加上要去的目录名及其路径名。如果不加目录名,则回到用户起始目录。

[pa@ locahost pa] $ cd /

[pa@ locahost /] $ cd

[pa@ locahost pa] $

也可以上下变更目录,因为“.”代表当前目录,“..”代表当前目录的父目录。如要进入当前目录的上级目录,只需输入如下命令:

```
[pa@ locahost pa] $ cd ..
[pa@ locahost home] $
```

(2) 文件管理命令

① 文件显示——more 令

more 命令可以逐屏显示文件内容,此命令适合于显示内容较多的文件。现在进入 linux-2.4 目录下的 kernel 子目录去看一下 Linux 进程调度程序 sched.c 的源代码。

```
[pa@ locahost pa] $ cd /usr/src/linux-2.4/kernel
[pa@ locahost kernel] $ ls
acct.c      exit.c     kksymoops.c   panic.c     resource.c   time.c
capability.c fork.c    kmod.c        pid.c       sched.c      timer.c
```

```
[pa@ locahost kernel] $ more sched.c
```

系统执行该命令,分屏显示该文件,显示满一屏后,在屏幕下方显示

--more-- (XX%)

这里的“XX%”是指现在已显示的内容占整个文件的百分比。按空格键就可以显示下一屏内容。

② 文件显示——cat

cat 命令用于显示文件内容,但不能逐屏显示。对于大文件而言,使用该命令将使文件内容很快向上滚动,可用 Ctrl + s 键暂停滚动、Ctrl + q 重新显示下面的内容来控制文件内容的阅读。

③ 文件换名和移动——mv

mv 命令可用于文件换名,其格式如下:

mv [参数] 源文件 目标文件

如果在 pa 目录下,将文件 test 换名为 test1,命令如下:

```
[pa@ locahost pa] $ mv test test1
[pa@ locahost pa] $ ls
sub test1
```

名为 test1 的文件已换名为 test1。这条命令的功能是在 pa 目录下建立一个 test1 文件,并将 test 文件的内容送入 test1 中,再撤销 test 文件。

如果上述换名的两个文件不在一个目录中,实际上就是实现了文件移动功能。如在 pa 目录下的 test 要移动到 pa 下的 sub 子目录中,用如下命令:

[pa@ locahost pa] $ mv test sub/

[pa@ locahost pa] $ ls

[pa@ locahost pa] $

sub

[pa@ locahost pa] $ cd sub

[pa@ locahost sub] $ ls

test1

④ 删除文件——rm

rm 命令用来删除目录中的文件。用户只有具有对文件的执行和写的权利,才能删除目录中的文件,否则,系统会显示无权存取的提示信息。

⑤ 拷贝文件——cp

文件拷贝命令有两种类型,第一个类型是将一个文件拷贝到另一个文件中,第二个类型是把文件拷贝到指定的目录中。

第一个类型的命令形式为:

cp 源文件 目标文件

如:cp file1 file

上述命令是将 file1 文件的内容拷贝到 file 中,如果 file 文件已存在,新拷贝的内容将覆盖原有内容,拷贝后,这两个文件内容一样。如果 file 文件不存在,cp 命令将先建立 file 文件,并将 file1 文件的内容拷贝到 file 中。

第二个类型的命令形式为:

cp 源文件 目标目录

如将 pa 目录下的文件 test1 拷贝 sub 子目录的命令如下:

[pa@ locahost pa] $ cp test1 sub/

(3) 查询进程状态命令——ps

用户可用 ps 命令查询进程的状态,该命令显示当前在系统中活动进程的有关状态信息。下面列出执行 ps 命令后显示的信息:

[pa@ locahost sub] $ ps -l (-l 选项将产生一个长列表,用来显示进程的状态)

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
4	S	500	5135	5115	0	75	0	-	1338	Wait4	tty1	00: 00: 00	bash
0	R	500	5175	5135	0	80	0	-	774	-	tty1	00: 00: 00	ps

命令执行结果的第一行是进程状态信息表的表头,下面解释各栏的意义。

F: FLAGS 域。

S: 进程的状态:O 不存在 S 睡眠状态 W 等待状态 R 运行状态

I 中间状态 Z 已经结束状态 T 已停止状态。

UID: 进程所有者的用户 ID。

PID: 进程的 ID。

PPID: 父进程的 ID。

C: 有关调度的 CPU 信息。

NI: Nice 数值,用来计算进程优先级。

ADDR: 如果进程在内存,表示进程的内存地址,否则为磁盘地址。

SZ: 进程内存映像的块数大小。

WCHAN: 进程正在等待或睡眠的事件,若为空,则表示进程正在执行。

TTY: 进程的控制终端。

TIME: 进程的累计执行时间。

CMD: 命令名。

4. 屏幕编辑程序 vi

屏幕编辑程序 vi 不但提供面向屏幕的编辑功能,而且还有一部分行编辑功能,可以完成任何编辑任务。vi 是加利福尼亚大学伯克利(Berkeley)分校作为 BSD UNIX 的一部分开发的,在 UNIX System V 推出之后,成为 UNIX 的一部分。在 Linux 中,运行的是与 vi 兼容的编辑程序。

vi 是屏幕编辑程序,它将屏幕的内容作为被编辑文件的窗口。在 vi 进行编辑工作时,是在文件的副本上进行的,并不是对源文件本身进行修改。如果在编辑过程中发生了错误,可以放弃所做的修改,重新回到原来的文件。只有当用户发出保存命令后,被修改的文本才覆盖原来的文件。

进入 vi 编辑程序,用如下命令:

[pa@ locahost pa] $ vi myfile

其中,myfile 为要编辑的文件的路径名,若文件不存在,将建立一个新文件。

vi 有输入和命令两种状态。用户在输入状态下可以输入文本;在命令状态下则执行存档、离开 vi 等命令。如果是建立新文件,进入 vi 后,输入插入命令(i 命令),即可开始编辑输入。若要对一个已有的文件进行编辑,则需要先移动光标的位置,将光标定位到要编辑的地方,再用 i 命令插入文本,插入完成后,按 Esc 键退出文本方式。

要进入输入方式,使用 i 命令如下。

i:从光标所在位置前面开始输入文本。

a:从光标所在位置后面开始追加文本。

要结束编辑或离开编辑程序的命令如下。

:q 结束编辑(quit)。

:q! 强制离开命令,若不想存档而放弃修改过的内容时使用。

:w 存档(write),其后可加要存档的文件名。

:wq 存档后离开。

本章小结

本章首先讨论了操作系统的基本概念、基本原理和主要功能。操作系统在整个计算机系统中的位置是,位于裸机之上的第一层核心软件,负责系统的软、硬件资源的管理和调度;处理、协调并发活动,并为用户和上层软件提供良好的用户界面。

操作系统提供两种类型的用户界面:操作界面和程序界面。这两类不同的界面分别用于对计算机的操作、控制;在程序中请求操作系统的功能服务。每一个操作系统都具有两个不同的用户界面,只是操作界面与实际的操作系统有关。具有脱机操作方式的操作系统(如批量操作系统)提供的操作界面是作业控制语言;而具有联机操作方式的操作系统(如分时操作系统、个人计算机操作系统)提供的操作界面是键盘命令和图形化操作界面。应了解不同类型的实例操作系统(如 Windows 系统、Linux 系统)提供的用户界面。

现代操作系统提供多用户、多任务的计算环境。本章介绍了为实现多用户、多任务的并发执行而引入的进程概念,讨论了进程管理的重要数据结构——进程控制块(PCB)、进程的状态及动态变迁、进程状态变迁的控制,以及为协调并发活动而提供的进程互斥、进程同步的概念及实现技术。了解进程的概念、作为进程存在的重要标志的 PCB 结构、进程的动态活动以及操作系统对进程实施的管理是十分重要的。在这种认识的基础上,才能真正体会现代实例操作系统(如 Windows 系统、Linux 系统)的功能,并对其有初步的认识。

本章还讨论了操作系统资源管理的策略和方法,针对多用户、多任务对系统资源的竞争,展开操作系统资源管理策略与方法的论述,给出了处理机管理、存储管理、设备管理、文件系统的功能与实现技术。

本章最后讨论了 Windows 系统和 Linux 系统的特点、结构、功能及用户界面,还简单介绍了这两个实例操作系统的使用方法。希望用户在了解操作系统基本原理的基础上,能对操作系统的实现技术有一个初步的了解,并能较熟练地使用这两个系统。

习 题 三

1. 画出计算机系统的组成,并标出操作系统在计算机系统中的位置。
2. 什么是多道程序设计技术?试述多道运行的特征。
3. 什么是分时技术?

4. 什么是操作系统？从资源管理的角度去分析操作系统，它的主要功能是什么？

5. 最常见的操作系统类型有哪几种？

6. 什么是操作系统的用户界面？它分为哪两个类型？

7. 什么是进程？进程与程序的主要区别是什么？

8. 进程有哪几种基本状态？试画出进程的3个基本状态的变迁图，并说明发生这些变迁可能的原因。

9. 什么是进程控制块？它有什么作用？

10. 常见的进程控制原语有哪些？进程创建原语的主要功能是什么？

11. 什么是临界资源？什么是临界区？什么是互斥？

12. 3个并发进程共用一个公共变量Q，试用信号灯的P，V操作实现这3个进程的互斥，并说明信号灯值的取值范围。

13. 什么是进程同步？

14. 常用的资源分配策略有哪两种？

15. 常用的进程调度算法策略有哪两种？

16. 什么是可剥夺的调度方式？

17. 什么是逻辑地址？什么是物理地址？什么是地址映射？

18. 什么是动态地址映射？试画出动态地址映射的过程。

19. 什么是虚拟存储器？

20. 在分区分配的放置策略中有首次匹配、最佳匹配两种常用的策略，指出它们的特点和区别。

21. 设某系统虚地址位数为32位，采用分页存储技术，其页面大小为4 K。试画出该系统的地址结构。

22. 已知主存容量为64 K，某一作业A的地址空间为4 K，它的4个页面（页面大小为1 K）0、1、2、3被分配到主存的2、4、6、7块中：

（1）画出作业A的页表。

（2）当200号单元处的指令“mov r_1，[3500]”执行时，如何进行正确的地址变换，以使3 500处的内容12 345送入r_1中。要求画出地址变换过程。

23. 简单分页系统与请求分页系统有什么区别？

24. 为实现页面请调，页表应扩充哪些数据项？每个数据项有什么物理意义？

25. 什么是系统“抖动”？

26. 什么是置换算法？在页式系统中最常用的置换算法是什么？

27. 为实现页面淘汰，页表应扩充哪些数据项？每个数据项有什么物理意义？

28. 什么是设备独立性？

29. 设备管理中最重要的数据结构是什么？

30. 什么是文件？文件的逻辑结构有哪两种？

31. 对文件的存取有哪两种基本方式？各有什么特点？

32. 文件的物理结构有哪3种？

33. 什么是文件目录？

34. 什么是树型目录结构？什么是文件路径名？什么是当前目录？

35. 常用的文件操作命令有哪些？

36. 什么是文件共享？采用“链接技术”如何实现文件共享？
37. 什么是文件的保护问题？采用存取控制表如何实现文件的保护？
38. 什么是死锁？试举例说明。
39. 产生死锁的原因是什么？产生死锁的必要条件是什么？
40. Windows 系统属于哪一类操作系统？它的主要特点是什么？
41. Windows 系统提供的用户界面是什么？
42. 什么是 Windows 系统窗口？什么是文件夹？什么是快捷方式？
43. Linux 系统提供的用户界面是什么？
44. 当注册进入 Linux 系统时，用户在目录结构中的什么位置？
45. Linux 系统的核心原代码处于 Linux 目录树结构的什么位置上？
46. 路径名和文件名有什么关系？
47. 在什么条件下 mv 命令的作用是移动？什么情况下是换名？
48. 执行 ps 命令，确定该命令的进程号。

第四章　数据库系统及应用

数据库技术产生于20世纪80年代末，是当代数据管理的最新技术，经过30多年的发展，已形成理论体系，成为计算机科学的一个重要分支。它的出现使得计算机应用渗透到社会的各行各业，在数据处理领域中发挥着越来越大的作用。本章介绍数据库系统的基本概念和基本技术。

4.1　数据库系统概述

4.1.1　信息、数据和数据处理

人类生存的世界是一个物质的世界，同时也是一个信息的世界，信息伴随物质而存在，并随物质的变化而变化。

信息是现实世界各种事物的存在特征、运动形态以及不同事物之间的相互联系等诸要素在人们头脑中的抽象反映，进而形成概念，这就是消息、新闻、情报。

数据是表示信息的物理符号，是对现实世界的一种描述和表示，简单地说，它是信息的载体。信息和数据虽然是两个不同的概念，但由于它们之间有着紧密的、直接的关系，人们往往不严格地区分它们。计算机中的数据不再单指狭义的数字、字母一类的常规数据，也可以是文字、声音、图形和图像等类型的多媒体数据。

数据处理是对反映现实世界的大量原始数据进行收集、分类、整理、计算、存储、提取、传播、维护等一系列活动的总和。经过处理的数据是精炼的，能够反映出事物或现象的本质、特征以及事物间的固有联系，从而帮助人们在实际工作中找出规律，作出决策。

数据处理活动的特点是，数据量大，数据之间逻辑联系复杂，但计算相对比较简单，因而数据处理任务的矛盾焦点不是计算，而是把数据管理好。数据管理包括收集、整理、组织、存储、查询、传送、维护等基本操作，是数据处理的共性环节。数据在人类社会发展的进程中是一种极为重要的资源，因此，人们历来十分关注如何妥善地保存、利用和科学地管理这些资源，并不懈地努力开发通用而又方便好用的数据管理工具，数据库技术正是由此而发展起来的一门计算机软件技术。

4.1.2 数据管理技术的发展

数据处理的历史可以追溯到远古时代，原始人类用手指、石块、木棍、贝壳等来统计捕获猎物的数量，可以认为是数据处理的开端。按照使用技术的不同与设备的演变发展，数据处理的方式可分为人工式（1890 年以前）、机械式（1890—1946 年）和电子计算机式（1946 年以后）。利用电子计算机进行数据处理的发展过程中又经历了手工管理、文件系统和数据库系统 3 个阶段。以下简要介绍计算机数据处理的各阶段。

1. 手工管理阶段

20 世纪 50 年代中期以前，计算机只用于数值计算，解决科学与工程技术中的数学问题。这是由于计算机本身的软硬件及数据处理的现状所限。当时，计算机的内存、外存容量都极小，外存只有卡片、纸带、磁带，没有磁盘，硬件性能差，软件中也没有操作系统及数据管理的部分，因而没有条件进行数据处理。这一阶段主要是使用手工进行数据处理，即用户直接管理数据。图 4.1.1 给出了手工管理的示意，图中说明数据高度依赖于程序。

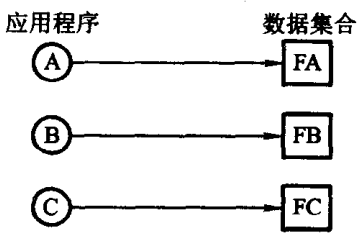


图 4.1.1 手工管理示意图

这时期的计算机数据管理，侧重处理功能，面向单个数据处理应用。数据为每个程序“自我拥有”，自己组织并使用各自的数据，应用处理单个进行，彼此不联系，数据无总体布局和计划，也没有结构，数据间缺乏逻辑组织，也不保存。当应用增加时，尽管有许多数据乃至全部数据，但新的应用仍要重新构造自己的数据。数据管理完全由用户的应用程序负责。同时，由于每个应用独立开发，对同一数据会使用不同的表示，反之，相同的数据表示亦可能代表不同的信息内容。这种不一致性极大地阻碍着信息的交流与共享。程序逻辑与数据之间高度依赖，毫无各自的独立性，一方变动，另一方随之要变动。

2. 文件系统阶段

20 世纪 50 年代中期后，磁鼓、磁盘等直接存取设备相继出现，磁盘技术的高速发展推动了数据处理技术的发展，这是计算机应用从数值计算转向数据处理的根本原因和动力，于是，管理数据的软件开始诞生，这就是文件系统。

文件系统是操作系统的一部分，负责管理、存放数据和程序。实际上，它是程序和数据间的一个接口，应用程序可以通过文件系统提供的存取方法和已经建立的一个或若干个文

件打交道，增加数据处理的灵活性。图 4.1.2 给出了文件系统管理数据的示意。

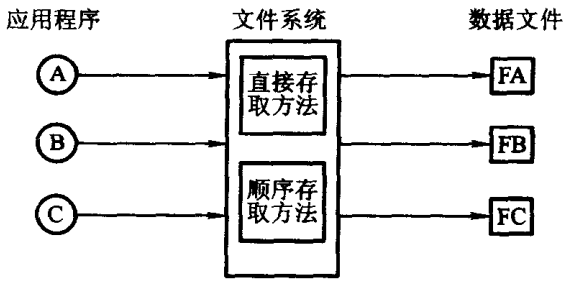


图 4.1.2　文件系统管理数据示意

这种处理方式仍然是以分散的相互独立的数据文件为基础进行的。60 年代中期，磁盘成为存储数据的主要设备，它促进了文件系统的进一步发展。文件系统提供了顺序文件、随机文件等多种文件组织形式，并提供了与之相应的多种存取策略，如顺序存取、随机存取等，它们实现了数据文件的逻辑结构与物理结构的分离和映射。利用这些技术，用户可把数据组织成顺序的和随机的文件，既可顺序存取，也可随机存取；除批量处理外，还可实时处理。此时，文件系统不仅仅只做输入/输出处理，还能完成数据组织、管理控制、数据维护等工作，既可以在整个文件级进行操作，又可以在文件记录级进行操作，如索引、排序、复制、比较等。

文件系统中，数据仍然面向特定的应用程序。根据应用要求，文件的记录内数据间的关系形成了一定的逻辑结构，但文件的整体却仍无结构。数据的共享在文件级成为可能，但在记录级或数据项级仍然不能共享。当不同应用程序使用的数据部分相同时，还必须构造各自的文件，这样仍然存在大量的数据冗余，数据独立性差，也得不到集中管理。

3. 数据库系统阶段

随着计算机技术和设备的迅速发展，计算机广泛地应用于企业和部门的管理，社会对数据处理的要求越来越高。首先，数据作为一种公共资源，需要集中控制，统一管理，由各种用户共享，因此应该大量地消去冗余数据，节省存储空间。其次，当数据变更时，能节省对多个数据副本的多次变更操作，从而可大大缩小计算机时间开销，且更为重要的是，不会因遗漏某些副本的变更而使系统给出一些不一致的数据。再次，软件价格不断上升，编制和维护应用程序的成本也随之增加，因而对数据独立性的要求更加迫切，即当数据逻辑结构改变时，那些不要求这种改变的用户的应用程序不需作相应改变，从而节省应用程序开发和维护的代价。数据库技术就是为克服文件系统的这些弊端而产生和发展起来的。

数据库系统的特点是数据不是只对某一特定应用，而是面向某领域、某部门的所有应用，具有整体的规划和结构。各种数据集成在一起，统一管理、控制，不同的用户按对整体数据库的不同逻辑视图来共享数据库，数据冗余度小，程序与数据间具有独立性。数据库系统的数据处理如图 4.1.3 所示。

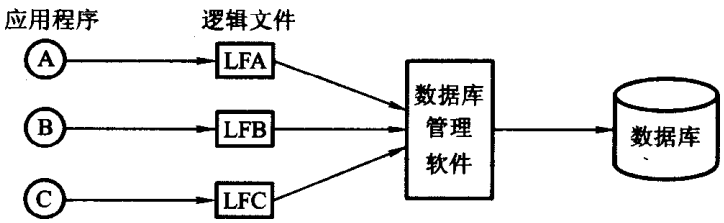


图 4.1.3 数据库系统数据管理示意

数据管理进展到数据库系统阶段是一个质的飞跃,数据库技术的产生和发展以当时国际上出现的 3 个事件为标志:

① 1968 年美国 IBM 公司研制了世界上第一个数据库管理系统(Information Management System, IMS)。

② 1969 年美国 CODASYL(Conference on Data System Language)组织的数据库工作小组发表了 DBTG(Data Base Task Group)报告。

③ 1970 年美国 IBM 公司研究员 E. F. Codd 发表了关系数据库理论的论文。

这 3 个事件是数据处理史上划时代的标志,是从文件系统进入数据库系统的里程碑。它们都曾引起计算机界较大的轰动。尤其是 E. F. Codd 的关系数据库理论的论文是一件开创性的工作,它开创了把数据理论用于计算机的历史,这一杰出成果获得了 1981 年的 ACM 图灵奖。

4. 数据库系统的特点

数据库技术之所以成为当今最先进的数据管理工具并得到广泛的应用,是由于它克服了前期数据处理方法的致命弊端,形成了自己的一系列优良特性。主要表现在以下几个方面。

(1) 数据集成

它将同一组织机构里的各种应用所涉及的信息集中起来进行统一的规划、协调和设计,构成一个面向全局的整体数据结构,并将数据按此结构组织到数据库中。

(2) 数据共享

各个用户按自己的需要对数据库中的数据进行存取和利用,达到最大限度的共享。这样,多个应用可以使用同一个数据,某一应用只是存取数据库的一个子集,不同的应用使用的数据可以任意重叠;在同一时刻多个用户可以存取同一个数据;开发新的应用可以利用原有数据;可为社会开放,成为社会的信息资源。

(3) 控制冗余

它改变了用户各自拥有自己数据文件的局面,消除了大量有害冗余,把冗余度降到了最小。许多数据文件被逻辑地集成为单一的结构,数据也只要存储一次,使结构和数据高度精

炼，能减少开销，节省空间和防止数据不一致。

应该指出，数据库中并不是完全消除了冗余，有时保留一定的冗余也会提高系统效率，尤其在分布状态下，为了减少数据在网上的传输时间，适当地安排冗余是常用的策略。

(4) 数据一致性

数据一致性是表示客观世界同一事物状态的数据，不管出现在何时何处，都是一致的、正确的，都具相同的值和特性。

由于控制冗余，可以在一定范围内避免数据的不一致性，不出现不相容和矛盾的现象。数据冗余、并发控制不当、各种故障错误以及计算机病毒是造成数据不一致的主要原因，数据库系统考虑了破坏数据一致性的因素，并采取了相应的措施来维护数据库的一致性。

(5) 数据安全性

数据安全性是防止无关人员获取其不该知道的数据。数据库系统本身采取了一系列措施和策略来保证数据的安全。如通过检查用户口令鉴别身份、利用子模式屏蔽与用户无关的数据、限定用户权限使其按规定的范围和对象进行指定的操作、对数据加密存储、审计等都是常用的安全措施。

(6) 数据完整性

数据完整性是指数据的正确性、真实性、客观性。凡是已失真了的数据就是其完整性受到了破坏或未得到保证。灾祸、故障、失误等均可能破坏数据的完整性，数据库系统采用了完整性约束与检验、并发控制、故障恢复等技术，有效地保护和维护了数据的完整性。

(7) 数据独立性

数据与使用数据的程序分离的特性称为数据独立性。其目的是使得数据或应用程序的修改不导致对方的修改。

数据独立性又分为逻辑数据独立性和物理数据独立性。

逻辑数据独立性是指数据库的全局逻辑结构发生改变时，用户应用程序不必改变，或反之。这是因为，在数据库系统管理下，局部数据(逻辑)结构与全局数据(逻辑)结构相互独立，只要局部数据结构的组成元素以某种相关形式存在于全局数据结构之中，则无论其构造方式如何改变，数据库系统都能根据用户的需要导出其局部数据结构。因而，这种全局数据结构的改变不影响局部数据结构，而应用程序是根据其局部数据结构编写的，因而应用的程序也不变。

另外，当全局数据结构增加或删除某些局部数据结构而影响到一些数据结构时，这种影响只涉及那些直接与这些改变相关的局部数据结构，而与此改变无关的其他局部数据结构无需改变。

反之，应用程序在一定范围内修改时，通常不需要修改局部数据结构，全局数据结构也无需修改。

物理数据独立性是指数据库的物理结构(存储设备、存储结构、存取方法)发生改变时，

应用程序无需改变。这是因为,数据的逻辑结构与物理结构相互独立,物理结构的改变不影响数据库的逻辑结构,应用程序是根据逻辑结构编写的,因而应用程序无需改变。

(8) 实施集中统一的管理

数据库系统对所有数据进行全面的集中统一管理,即对数据结构、数据特征(名称、类型、长度)、数据内容、数据操作、数据一致性、数据安全保密性、数据完整性等均按统一标准进行控制和管理。这一切都是由数据库管理系统(DBMS)完成的。

正是由于集中统一管理,才实现了上述共享数据、控制冗余、数据安全性、完整性、独立性、一致性等重要目标,从而获得了整体系统的优越性能。

4.1.3 数据库、数据库管理系统和数据库系统

1. 数据库

数据库是按一定结构存储在计算机存储设备上的相互关联的数据的集合。数据库中的数据是集成的,具有最小的冗余,可以为多个应用所共享,而它们又独立于应用程序,并只由一个软件来进行统一地管理。

对一个特定的数据库而言,它集中地保存着某一单位或某一领域内所有有用的信息,其中的数据是依据相互间的自然联系构造而成的,从而可以为不同用户的不同需求提供利用数据的存取路径。

因而,通常的数据库首先是一个单位或一个领域的通用数据集合。"库"的大小随单位或领域的大小、信息量的多少而不同。这里的"单位"可以是某工厂、学校、政府部门等,"领域"可以是某产品,如钢材品种、招生情况等。其次,这个数据集合是"集成化"的,它不是简单的堆积,而是经过分析、归并,尽量减少冗余而又全面兼顾各类应用要求聚集而成的精炼结构和数据,它们反映了应用领域的客观世界及其间的相互联系,可以满足所有用户的应用要求。

2. 数据库管理系统

数据库管理系统(DataBase Management System,DBMS)是数据库系统的核心软件,是为数据库的建立、使用和维护而配备的软件,它介于操作系统和用户之间,在操作系统的支持下负责对数据库进行统一的管理和控制,是一种系统软件。用户发出的或应用程序对数据库的各种操作命令,包括定义、查询、更新及各种控制,都要通过数据库管理系统来执行。数据库管理系统还要对数据库进行维护,并保证数据库的安全性和完整性、多用户对数据的并发使用以及发生故障后的系统恢复。

数据库管理系统是数据库系统的核心,是负责数据库的建立、使用、维护和进行统一的管理和控制的系统软件,在操作系统的支持下工作。用户对数据库的各种操作,都要通过数据库管理系统来执行。

(1) 数据库管理系统的功能

数据库技术经过30多年的发展，DBMS从初级到高级，其功能越来越强，技术和理论也越来越完善。最根本的发展是数据独立性越来越高，用户接口趋向于简明、方便、直观、非过程化、智能化。对DBMS的功能、性能、效率等诸方面的要求不断提高，使得DBMS的规模越来越大，复杂度越来越高。DBMS的功能主要包括如下几个方面。

① 数据定义。DBMS提供数据描述语言DDL，供用户定义其数据库逻辑结构，通常包括定义记录类、数据项类型、长度、记录间的联系、完整性、安全性约束等，并将所描述的内容从源形式转换为目标形式存入数据字典中。

② 数据操纵。实施对数据库数据的查询、插入、修改及删除等基本操作。

③ 数据库运行控制管理。这包括对整个数据库系统运行的控制、用户的并发存取控制、数据的安全性、完整性检查和执行、数据库内部维护等。

④ 数据库的组织、存储与管理。主要包括数据库的物理组织、存储结构、索引及存取方法和数据字典的管理。

⑤ 数据库的维护。主要是初始数据的装载、数据转换、数据库性能监视与分析、在数据库性能变坏或需求变化时重构造与重组织、数据库的备份及故障时的恢复等。

⑥ 数据通信。提供与其他软件系统通信的接口，例如与操作系统、其他DBMS的数据通信等，负责处理数据的传递，通常要协同完成。

(2) 数据库管理系统的组成

为了提供上述功能，DBMS通常应包括以下基本的组成部分。这就是：定义数据库结构的语言，即数据定义语言；操作数据库数据的语言，即数据操纵语言；数据库系统的各种实用程序，包括装配程序、控制程序、维护程序、故障恢复程序及其他实用程序。

① 数据定义语言及其翻译处理程序

数据库管理系统通常提供数据定义语言(Data Definition Language，DDL)和数据操纵语言(Data Manipulation Language，DML)。

DDL用于定义数据库的整体逻辑结构和局部逻辑结构中各数据对象的名、类型、长度、联系及完整性、安全性约束等内容，也就是定义数据库的模式、存储模式、外模式。它们都是源程序，经各种模式翻译程序处理生成目标，装入数据字典中。

DML提供给应用人员对数据库进行数据插入、检索、修改、删除等基本操作，又称为数据子语言。DML有宿主型DML和自主型DML两种。宿主型DML不能独立使用，必须嵌入某一种高级语言中混合编程，因而也叫嵌入式DML；自主型DML又称自含型DML，它们是交互式命令语言，可以独立使用。

② 数据库运行控制程序

数据库运行控制程序负责数据库运行时的控制与管理，包括系统总控程序、事务处理程序、空间管理程序、缓冲区管理程序、读写程序、并发控程序、事务管理程序、运行日志管理程序、存取控制程序、数据存取程序、数据更新程序、完整性检查程序、安全性检查程序等。它

们在数据库运行过程中监视着对数据库的所有操作,控制管理数据库资源,处理多用户的并发操作等。

③ 数据库实用程序

各 DBMS 提供的实用程序差异较大,主要包括数据装入程序、数据转贮程序、数据重组程序、数据库恢复程序、统计分析程序、性能监测程序、数据转换程序、通信程序等。利用它们完成数据库的建立与维护,进行数据转换和通信等。

3. 数据库系统

数据库系统是数据库、软件、硬件和人员组成的一个集合体。图 4.1.4 给出数据库系统组成的示意。

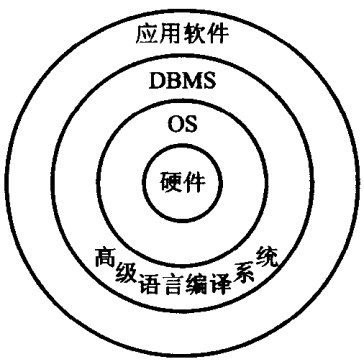


图 4.1.4 数据库系统组成

(1) 数据库

数据库包含了一个企业所有应用所需的数据,是用户操作的对象,也是企业的宝贵资源。它存储于计算机系统中,通常数据量大,一般只能存储在外部设备(如磁盘)上。库中包含多种类型的数据,以及数据间的复杂联系,按应用需求采集的数据将按特定的数据结构存储,随着数据库的运行,数据量将不断扩大,故要求外存容量较大,有的则需根据应用需要将数据分散到多个计算机系统中。

(2) 软件资源

数据库系统用户要将自己的数据处理要求变成对物理数据的存取,需要一系列软件的支持,主要包括以下几个。

① 数据库管理系统。DBMS 是数据库系统中的核心软件,用于处理所有用户对数据库存取的请求,通常它是由操作系统支持工作的。

② 操作系统。OS 是 DBMS 最重要的支持软件,DBMS 对于数据的存取最终要通过 OS 来执行,因此 OS 的性能,尤其是 OS 的 I/O 性能直接影响数据库系统的效率。另外其功能也直接影响数据库系统的功能。

③ 程序设计语言及其编译系统。DBMS 通常提供最基本的数据操作命令,提供与某些高级程序设计语言的接口。利用高级语言的这类强大的程序设计功能,协同完成较复杂的数据处理。

④ 辅助或配合 DBMS 工作的软件。这些软件因 DBMS 不同而不同,并非 DBMS 的必备功能,主要有数据库应用开发工具、数据库辅助设计工具、数据转换设施等。它们为用户高效、简单灵活地开发应用提供有力的支持,已成为一种重要的数据库软件资源。

⑤ 应用软件。应用软件是为特定用户的业务开发的专用软件系统,通常由高级语言和 DBMS 提供的数操纵语言编写而成。

(3) 硬件资源

数据库系统的硬件资源包括主机、外存及外部设备等。通常要求较大内存和外存空间,在数据库系统下,除了 DBMS 外,还有操作系统、应用程序、系统缓冲区、工作区、系统通信单元及数据库的各种表格、目录等,很多都要常驻内存,外存是存放数据库的,如果太小,装不下数据将使整个系统无法运行。同时要求有较高的通道能力,因为数据交换频率高、量大,如果能力较低,则直接影响数据库系统的效率。

(4) 数据库管理员

数据库管理员(DataBase Administor,DBA)是建立、使用、管理和维护数据库的专业技术员,其职责是:应用业务需求分析、数据库设计、决定用户使用权限、系统维护、数据库恢复等。

4.1.4 数据库系统结构

美国国家标准协会(ANSI)对数据库系统提出了一个 ANSI/X3/SPARC(Standard Planning And Requirement Committee)的分级体系结构,将数据库结构分为三级,即外模式、模式和内模式,所以又称为三级模式结构。目前流行的数据库系统通常采用三级模式结构,从不同的数据观点或用户视图出发,分别以外模式、模式和内模式的概念描述数据的不同层次。最接近物理存储的是内模式,外模式涉及每个用户观点的数据,模式介于外模式和内模式之间,反映整个数据库的抽象结构。数据库系统的三级模式结构如图 4.1.5 所示。

数据库系统的三级模式结构是对数据的 3 个抽象级别,它把数据的具体组织留给 DBMS 管理,使用户能逻辑抽象地处理数据,而不必关心数据在计算机中的表示和存储。三级模式结构之间差别往往很大,为了能够在内部实现这 3 个抽象级别的联系和转换,数据库管理系统在这三级结构间提供了两级映像:外模式/模式映像,模内/内模式映像。下面详细考察这个三级结构。

(1) 模式

模式也称概念模式,是数据库中全部数据的逻辑结构和特征的描述,是所有用户的公共数据视图。它是数据库系统三级模式结构的中间层,不涉及数据的物理存储细节和硬件环境,与具体的应用程序、所使用的应用开发工具及高级程序设计语言无关。

实际上模式是数据库数据的逻辑视图。一个数据库只有一个模式。数据库模式是以某种数据模型为基础,统一、综合地考虑所有用户的需求而构造成的逻辑整体。定义模式时不仅要定义数据的逻辑结构,例如,数据记录由哪些数据项组成,数据项的名字、类型、取值范围等,而且要定义与数据有关的安全性、完整性要求,定义这些数据之间的联系。

(2) 外模式

外模式也叫子模式或用户模式,是用户与数据库系统的接口,是用户看到和用到的局部数据的逻辑结构和特征的描述,它是数据库用户的数据视图,是与某一应用有关的数据的逻辑表示。

外模式通常是模式的逻辑子集。由于一个数据库有多个用户或多种应用,所以一个数据库可以有多个外模式。同一个外模式可以为某一用户的多个应用所使用,但一个应用程序只能使用一个外模式。每个用户只能看见和使用所对应的外模式涉及到的数据,数据库中的其余数据对它们来说是不可见的。外模式也是保证数据库安全性的一个有力措施。

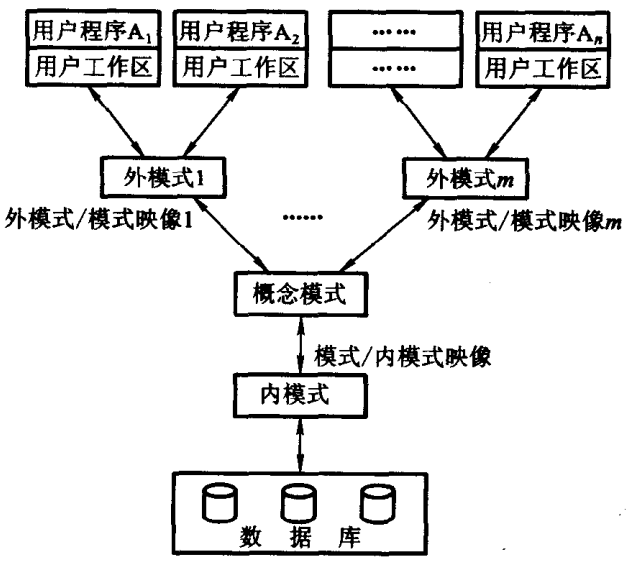


图 4.1.5 数据库系统的三级模式结构

(3) 内模式

内模式也称作存储模式,它是数据在物理结构和存储结构方面的描述,是数据在数据库内部的表示方式。例如,记录的存储方式是顺序存储、按树结构存储还是按索引(hash)方法存储;索引按照什么方式组织;数据是否加密,是否压缩存储;数据的存储记录结构有什么规定等。一个数据库只有一个内模式。

3 种不同的模式间存在如下关系:外模式是模式的逻辑子集,它可以从模式推导出来,模式是各外模式的逻辑汇总和综合,内模式是模式的具体实现。

(4) 外模式/模式映像

模式描述的是数据的全局逻辑结构,外模式描述的是数据的局部逻辑结构,一个模式可能存在多个外模式,对于每一个外模式,数据库系统都有一个外模式/模式映像,它定义了外模式和模式之间的对应关系。这些映像定义通常包含在各自的外模式描述中。如果数据库的整体逻辑结构(即模式)要作修改,那么只需对外模式/模式映像作出相应的修改,就可以使外模式仍然保持不变,应用程序不必修改,这就是数据库实现的逻辑数据独立性。

(5) 模式/内模式映像

数据库中只有一个模式,也只有一个内模式,所以模式/内模式映像是唯一的,它定义了数据的全局逻辑结构与存储结构之间的对应关系。这个映像定义通常包含在模式描述中。如果内模式要作修改,即数据库的存储设备和存储方法有所变化,那么只需对模式/内模式映像作出相应的修改,就可以使模式仍然保持不变。这就是数据库实现的物理数据独立性。

三级模式间的两种映像(或映射)由 DBMS 实现,从而建立起外模式同模式、模式同内模式间的联系。正是由于 DBMS 的映像功能,才能将基于外模式的逻辑操作转换成对数据库的物理操作。

4.1.5　数据库系统的工作过程

数据库系统的工作过程是以子模式、模式、内模式为依据展开的,当一个应用程序需要存取数据库中数据时,由应用程序、DBMS、操作系统、硬件等协同完成。图 4.1.6 显示了应用程序通过 DML 语句对数据库进行检索操作时数据库系统的工作过程。其步骤如下:

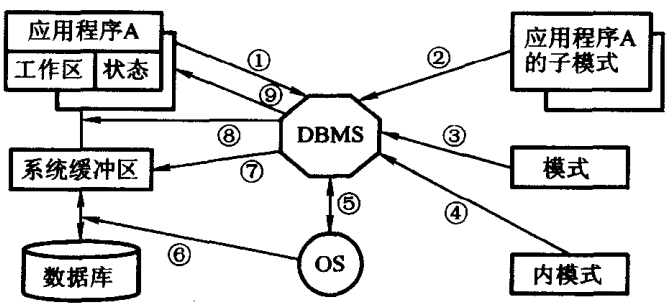


图 4.1.6　数据库系统的工作过程

① 执行应用程序 A 中检索数据库的语句并向 DBMS 发出读取数据的操作命令,该命令包含进行什么操作、操作对象及读取的数据应满足的条件信息。

② DBMS 进行存取权限检查,若合法,则调入应用程序 A 引用的子模式。

③ DBMS 调入模式,进行子模式到模式的变换,确定应读入的模式中的数据对象。

④ DBMS 进行模式到内模式的变换,确定应读取的物理记录。

⑤ DBMS 向 OS 发出读进所需物理记录的命令。

⑥ OS 执行读命令。将所需数据经过 I/O 缓冲区读入到系统缓冲区中。

⑦ DBMS 比较模式和子模式。将系统缓冲区中的数据组织成应用程序 A 所要求的记录形式。

⑧ DBMS 将应用程序记录送到相应用户工作区(UWA)。

⑨ DBMS 将检索命令执行状况信息送到应用程序 A 对应的状态返回区。

DML 中的其他操作命令执行状况过程类似于上述的读操作过程。对于删除操作,需要通过检索找到将删除的对象;对于插入操作,需要找到插入的位置;对于修改操作,需要将要修改的对象数据读入 UWA,在 UWA 中修改完成后再写入到数据库原位置中去。

需要强调的是,应用程序对数据库的任何操作都是通过 DBMS 在 UWA 中进行的。

4.2 数据模型

4.2.1 什么是数据模型

所谓模型,是指对不能直接观察的事物进行形象的描述和模拟,也可以说是对客观世界中复杂对象的抽象描述。在数据库领域中,数据模型是客观事物及其联系的数据描述,也就是把现实世界中的各种事物及其之间的联系用数据及数据间的联系来描述,以一定的结构形式表示。数据库是一个结构化的数据集合,数据模型就是数据库的框架,这个框架形式化地描述了数据库的数据组织形式,包括数据的结构、数据的性质、数据之间的联系、完整性约束条件以及某些数据变换规则等,在这个框架约束下填上具体数据才是数据库。

表示事物及事物间联系的模型称为数据模型。数据模型是数据库设计人员、程序员和用户之间进行交流的工具。

目前广泛使用的数据模型可分为两种类型。

一种是独立于计算机系统的模型,完全不涉及信息在系统中的表示,只是用来描述某个特定组织所关心的信息结构,这类模型称为"概念数据模型"。概念模型用于建立信息世界的数据模型,强调其语义表达功能,应该概念简单、清晰,易于用户理解。它是现实世界的第一层抽象,是数据库设计人员和用户之间进行交流的工具。这一类中著名的模型是"实体联系模型"。

另一种数据模型是直接面向数据库的逻辑结构,它是现实世界的第二层抽象。这类模型涉及到计算机系统和数据库管理系统,又称为"结构数据模型"或"实施模型"。这类数据模型有严格的形式化定义,便于在计算机系统中实现。例如,层次、网状、关系等模型。

一个数据库的结构数据模型一般应包括数据结构、数据操作和数据完整性约束3个部分。

数据结构是对系统静态特性的描述，是所研究的对象类型的集合，它们通常是数据库结构的基本组成部分。数据库中的研究对象有实体和实体间联系两类，数据模型的数据结构用于描述它们，应具有简单、易于理解和表达能力强的特点。

数据操作是对系统动态特性的描述，是对数据库中各种对象类型的实例(值)所允许操作的集合，其中包括各种操作的规则。对数据库的基本操作主要有查询、插入、修改、删除等。数据模型要定义这些操作的确切含义、操作符号、操作规则以及实现操作的语言。

完整性约束给出数据模型中数据及其联系所具有的制约和依赖规则，进一步给数据模型动态特性作出描述和限定，强制对数据库的插入、删除、修改操作必须在满足该组规则的条件下执行，以保证数据的正确、有效和相容。

4.2.2 数据的描述

数据处理先要进行数据描述，就是要把事物的特性和事物之间的联系表示成计算机中的数据，这个过程要经历现实世界、信息世界和机器世界3个领域。

1. 数据描述涉及的3个领域

(1) 现实世界

现实世界是存在于人们头脑之外的客观世界，它由客观存在的事物、事件、活动及所处环境等组成，通过对这些对象的管理，从而达到对现实世界既定目标的管理。

计算机中所处理的每一数据，都对应着现实世界中某一事物的某一特性和物理状态，运行的每一程序都是模拟人做某种事务处理，现实世界中要处理的主要对象就是事物，或是它们之间的联系。这些实体是千姿百态的，其间的联系也是错综复杂的。

(2) 信息世界

信息世界是现实世界在人们头脑中的反映，人们把它们用文字和符号记载下来。信息世界中，数据库技术用到下列一些术语。

① 实体。客观存在并可相互区别的事物叫做实体。可以触及的对象是实体，如一个人、一辆车、一个城市等；无形的事物和抽象的概念是实体，如一场球赛、一次约会、交通规则等；事物之间的联系也可以是实体，如学生选课、订货合同等。实体是应该可以标识的，可以相互区分开来的。

② 实体集。性质相同的同类实体的集合叫做实体集。如所有的人、所有的车、所有的订货合同等。

③ 属性。实体所具有的某一特性叫做属性。例如人的姓名、年龄、性别、住址等；订货合同的合同号、货物名称、规格、单价、交货地点等，这些都表示了实体的固有特征。

④ 实体标识符。能够区分同类实体中各个实体的属性集叫做实体标识符。如身份证

号就可以将人这个实体集中的每个人区分开来,因而身份证号这个属性就是人这个实体的实体标识符。

属性和实体都有“型”与“值”之分。例如,性别是属性型,而男、女则是该属性的值;又如,年龄是属性型,20岁、30岁是对应的属性值。型相当于变量,值相当于变量值。同样,“人”是一个实体型,而牛顿、达尔文是具体的人,是值。

(3) 机器世界

在计算机处理时,实体用记录来表示,记录是数据项的集合。同样,记录也有“型”与“值”之分,与实体“型”与“值”有着对应的关系。

2. 记录之间的联系

实体间有两种联系:一是实体内部的联系,反映在数据上是记录内部及数据项间的联系;二是实体与实体之间的联系,反映在数据上就是记录之间的联系。

数据库系统中要考虑这两种联系,尤其是记录之间的联系,这种联系比较复杂,因而增加了数据库系统的复杂性。记录之间的联系虽然复杂,但抽象化后,可把它们归结为三类。

(1) 一对一联系(1:1)

如果两个实体集A、B中的每一个实体至多和另一个实体集中的一个实体有联系,则A、B之间是一对一联系。记作1:1。例如车票和座位是两个实体集,一张车票对应一个座位,一个座位只有一张车票,车票和座位是一对一的联系。再如,学生班和班长之间也是一对一的联系。

(2) 一对多联系(1:n)

有两个实体集A、B,如果A中至少有一个实体与B中任意一个实体有关,而B中每一个实体至多和A中的一个实体有关,则A对B是一对多联系。记作1:n。如学生班级是一个实体集,学生是一个实体集,一个班级可以有多个学生,一个学生只属于一个班级,班级与学生之间是一对多联系。

(3) 多对多联系(m:n)

如果两个实体集A、B中,都至少有一个实体和另一个实体集中的任意个实体有关,则A、B之间是多对多联系。记作m:n。如学生与教师,学生与选课,货物与供应商等都是多对多联系。

4.2.3 3种经典的数据模型

实体间的联系反映到数据库中就是数据间的联系,数据模型的作用就是能清晰的表示实体、实体之间的联系,也就是数据库的逻辑结构,以便用户更有效地存取数据。

最著名的数据模型有3种,即层次模型、网状模型和关系模型。

1. 层次模型

用树型结构表示实体及实体间联系的模型叫层次模型。树是由结点和连线组成的,结

点表示实体，连线表示实体间的联系。这种联系只能是 1:n 联系。通常把表示 1 的实体放在上方，称为父结点。表示 N 的结点放在下方，称为子结点。整个树是倒向的，最上层只有一个结点，称为根。根以外的结点都有一个父结点与它相连，同时有一个或多个子结点与它相连。没有子结点的结点称为叶结点，它们处在树的末端。同一父结点的子结点称为兄弟结点。较严格地说，层次模型是指满足下列条件的基本层次联系的集合：

① 有且仅有一个结点无父结点，此结点称为根。

② 根以外的其他结点都有且仅有一个父结点。

图 4.2.1 给出了一个层次模型的示例。其中大学实体为树根，各层父子结点间均为一对多的关系，即一个大学可以有多个系、部、处，一个系可有多个教研室、研究所，一个处可以有多个科等。现实世界中许多实体间的联系本身就是一个自然的层次关系，如家族关系，行政机构等，用层次模型很容易表示。

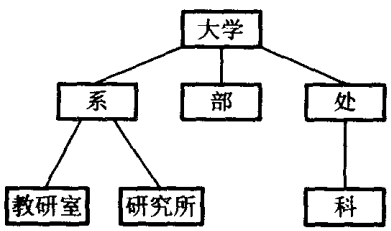


图 4.2.1 层次模型示例

但是，多对多联系不能直接用层次模型表示，必须设法分解为两个一对多联系，然后再用层次模型来表示。

对于层次模型，有必要说明以下几点：

(1) 结点间联系是单向的

即由父结点指向子结点，且父子结点间只存在单一联系。这样，对任一结点，只有唯一的一条自根结点开始的到达路径，这意味着对结点的存取必须从根开始，因而随机存取效率低。

(2) 树是有序树

即从左到右的顺序规定了任一子树的所有结点的先后次序。这意味着对结点的存取除上述的从根开始的从上到下的限制外，还必须遵循从左到右的规则。

(3) 树结点的任何记录的任何属性不可再分

层次模型是三大经典数据模型中最早出现的一个。基于层次模型的数据库管理系统的代表是 IBM 公司推出的 IMS(Information Management System)，它是世界上第一个数据库管理系统。层次模型能自然直观地描述现实世界中的自然层次关系，结构简单，查询路径唯一，但它不能直接描述实体间的复杂联系。目前基于关系模型的数据库管理系统占主导地

位,但层次模型仍有其历史的地位和固有的用户。

2. 网状模型

用有向图结构表示实体及实体间联系的模型叫网状模型。它是指满足下列条件的基本层次联系的集合。

① 可以有一个以上的结点无父结点。

② 一个结点可以有多于一个的父结点。

网状模型同层次模型一样,用结点表示实体集,用带箭头的连线表示实体间的联系。不过,在网状模型中,同一父子结点间的联系不再是唯一的,也可以是互为父子的。因而,要对每个联系命名。图 4.2.2 给出网状模型的示例。

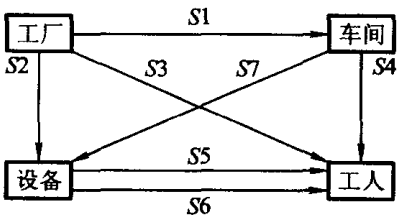


图 4.2.2 网状模型示例

图中矩形框表示的是实体,连线表示实体间的联系,*S*1、*S*2、…、*S*7 是给联系的命名。设备与工人之间有两种不同的联系,*S*5 表示设备由工人使用的关系,*S*6 表示设备由工人维护保养的关系,每个工人使用和保养的设备可以是不同的。

显然,网状模型是层次模型的一般形式,层次模型是网状模型的特例。它们的差别在于网状模型的连线更加复杂,纵横交错。

理论上网状模型可以直接表示实体间多对多的联系,多对多联系可以化解为两个一对多联系来处理。网状模型的描述能力较强,但其结构较复杂,由于到达一个结点的路径不唯一,用户须自选较优的路径,可能要涉及到系统结构的细节,加大了使用的难度。

网状模型的代表是美国的 CODASYL(Conference of Data System Language)所属的数据库任务组(Database Task Group,DBTG)于 1971 年提出的 DTBG 报告,它提出了一个网状数据库系统的方案。目前市场上流行的网状数据库系统大多是基于这个报告实现的。

3. 关系模型

用二维表结构表示实体及实体间联系的模型叫关系模型。一个二维表格称为一个关系,表中的每一行称为一个元组,每一列称为一个属性,每一属性的值集称为域,能唯一标识一个元组的属性集称为候选关键字。当有多个候选关键字时,被选定作为标识的候选关键字称为主关键字。组成主关键字的各属性称为主属性。

若一个关系有 N 个属性,则称该关系为 N 元关系。关系和属性都必须给出唯一的

命名。

表 4.2.1 给出的学生实体就是一个二维表，也称为一个关系。

表 4.2.1 学生

学 号	姓 名	性 别	年 龄	班 级
001	张明	男	18	981
002	李雪	女	19	982
003	陈志	男	18	981

其中，属性有学号、姓名、性别、年龄、班级，由它们构成的命名序列即为“学生”关系的型，也称关系模式。该关系中含有 3 个五元元组。

从前面的介绍可知，层次模型和网络模型都是用“连线”表示实体间的联系，是用指针来实现的，文件中存放的是实体本身的数据，而关系模型中则用“数据”来表示实体间的联系，因此文件中除存放实体本身的数据外，还要存放联系数据，在表示实体间的联系方法上有着本质的区别。

表 4.2.2、表 4.2.3 给出课程和选课两个实体，和表 4.2.1 一起标识现实世界中学生、课程和学生所选课程的情况。表 4.2.1、表 4.2.2 表示的是现实世界中的基本实体，表 4.2.3 则表示这两个实体间的联系，但它也是一个实体。这种联系是通过在“选课”关系中引入“学生”关系中的“学号”和“课程”关系中的“课程号”属性的数据而建立的。

选课关系中的学号和课程号分别是学生和课程两个实体集的关键字，这个关系中的数据可以说明“谁选修了什么课”和“某门课有谁选修”。“谁”和“课”在这里仅是代号，要知道代号代表的那位学生的具体情况，可以在学生关系中通过比较相同的学号值找到，同样也可以通过课号在课程关系中找到该课号所代表的课程详细情况。关系模型中具有一套非常灵活的查找所需数据的办法。

关系具有如下性质：表中任一属性均为不可再分的原子属性；行、列次序无关；表中任何两行不能完全相同；表中每一列命名不能相同。

表 4.2.2 课程

课程号	课程名	学时
001	DB	60
002	OS	68
⋮	⋮	⋮

表 4.2.3 选课

学号	课程号	成绩
001	001	90
001	002	85
002	001	88
⋮	⋮	⋮

4. 实体联系模型

实体联系(Entity - Relationship)模型简称 E - R 模型,它是一种用特定形式直接表示实体及实体间联系的模型。它在数据库设计时用来表示一个企业部门的信息结构十分方便有效,被广泛地应用于概念建模。E - R 方法的不断发展,使得它的应用早已超出数据库的范围而进入了软件工程、知识工程等领域。

E - R 模型包括实体、属性和实体间联系 3 个基本构件,用 E - R 图的方式直接表达实体及实体间的联系。其方法是:

① 实体集用矩形框表示,矩形框内标上实体名。

② 实体的属性用椭圆表示,椭圆内标上属性名,并用无向边与其实体相连。

③ 实体间的联系用菱形框表示,给联系命名,名字写在菱形框中,用无向连线将参加相应联系的实体矩形分别与菱形相连,并在连线上标明联系的类型(1∶1,1∶m,n∶m),"1"和"m"要写在对应实体矩形那边的连线上,不可写反。

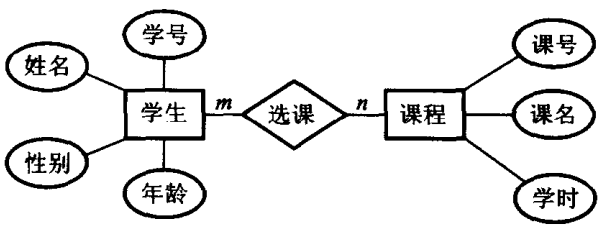


图 4.2.3 学生选课 E - R 模型

图 4.2.3 给出了一个学生选课数据的 E - R 模型。

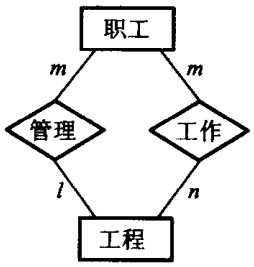
E - R 模型对复杂对象、复杂联系也有很强的表达能力。下面仅介绍几种特殊的实体间联系的表示方法。

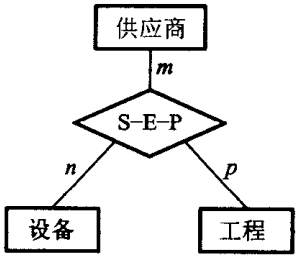
图 4.2.4 给出的是两实体间的多元联系。对于"工作"联系,它表示一个职工可参与多项工程工作,一项工程可有多个职工参加;对于"管理"关系,它表示一项工程可由多个职工管理,但一个职工只能管理某一项工程。

图 4.2.5 给出的是 3 个实体的多元联系,它表示某些供应商为某些工程供应某些设备的联系。

图 4.2.6 给出的是一个实体集内的二元联系,它表示一种零件可由多种其他零件装配而成,而一种零件又可装配于多种零件之中。

为了简洁,上述各 E - R 图中略去了各实体的属性细节。





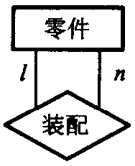


图 4.2.4 两实体多元联系　　图 4.2.5 3 个实体的多元联系　图 4.2.6 一个实体内的二元联系

E－R 模型是描述现实世界的有力工具，很有实用价值，得到广泛的应用。但目前还没有可直接支持 E－R 模型的 DBMS，至今 E－R 模型是作为中间数据模型使用的，还必须将它再转换为具体 DBMS 所能支持的模型。

4.3 关系数据库基础

在三大经典数据模型中，关系模型形成最晚，发展最快，是最具活力的一种数据模型。美国 IBM 公司研究员 E. F. Codd 对关系模型的发展作出了最杰出的贡献。关系模型具有结构简单、理论基础严密、数据独立性高、支持非过程化语言、一次操作存取多个元组、可直接表示多对多联系和使用方便灵活、简单易学等优点，它使得关系数据库广泛地应用于各个领域，在当今实用数据库产品中占主导地位。

关系数据库按关系模型组织数据库，关系模型的组成包含数据结构，数据操作和数据完整性约束 3 个部分。其数据结构十分简单，所谓的数据模型不过是一些二维表格框架，其中公共的属性名指示着表格间的联系。与层次模型、网络模型错综复杂的指针相比，采取了截然不同的处理思想。本节对关系数据库的有关内容作简要介绍。

4.3.1 基本概念

(1) 关系

关系是一个二维表，给表的每列取一个名字，称为属性(Attribute)，属性名应唯一，属性的取值范围称值域(Domain)。表中的每一行数据称为一个元组。所有元组的任意子集构成一个关系。

(2) 关键字

如果一个属性集的值能唯一标识一个关系的元组，而又不含有多余的属性值，则称该属性集为候选关键字。被选用的候选关键字称为主关键字，简称关键字。每一个关系都有且

仅有一个主关键字。

(3) 关系模式

一个关系的属性名表称为关系模式,也就是关系的型。关系模式就是二维表的表框架,相当于记录类型。设关系名为 R,其属性的一般形式为 $A_1, A_2, \cdots, A_n$,则关系模式通常可写成 $R(A_1, A_2, \cdots, A_n)$。一个关系数据库的逻辑结构是所有的关系模式、属性名、关键字的集合,把它称为关系数据库模式。

(4) 关系数据库

对应于一个关系数据库模式的所有关系的集合称为关系数据库。关系数据库模式是数据的型的表示,而关系数据库则是数据的值的表示。

(5) 视图

视图一般由关系数据库模式中的某个或几个关系中满足用户给定条件的若干属性列或元组组成,也可以是若干个不同关系进行关系运算的结果。简单地说,它是从一个或几个关系导出的关系,它本身是个逻辑上存在而物理上并不存在的虚表。因此,在数据库中只存在视图的定义,而并不存储视图的数据。它是外模式或子模式的基本内容。

应该指出,关系数据库中的视图与通常意义上的外模式有所差别。通常,特别在网状数据库系统中,用户必须预先定义外模式而构造出自己的子模式,并且只能通过子模式存取数据库。而这里的视图往往可以动态地从一个或多个基本关系上导出,进而还可以从视图上导出。用户不仅可以在自己定义的视图上使用数据库,还可以不通过视图而直接在基本关系上使用数据库。

4.3.2 关系数据库系统的数据描述

应用涉及的所有关系模式(或者称为关系框架)的集合就构成了该应用的关系数据库模式,也就是它的关系数据库的全局逻辑结构(即概念模型),通过数据描述语言来定义它。

关系概念模型的描述主要包括定义关系及其属性的名称、类型及长度、定义数据完整性约束及安全保密要求、关键字等。对于物理存储方面的问题,描述时不用涉及,只需说明数据本身的一些特性。物理存储细节由 DBMS 自行处理。

关系数据库定义包括模式定义、视图定义、索引定义等。其具体定义方法参见 SQL 中的数据定义部分。

4.3.3 关系数据库系统的数据操作

关系模型与其他模型相比,最具特色的是它的数据操纵语言,这种语言灵活方便,表达能力和功能都很强。例如它每次操作提供给用户的不是一个记录,而是一个记录集。它是非过程化的,用户只需指出做什么,无须指出怎么做;它可以独立使用,也可以与其他语言结合起来使用。这些都深受用户欢迎,也是它有生命力和发展前途的原因。

关系数据操纵语言的基础是关系运算，它以关系作为操作对象，对其施加集合运算、谓词演算和创造等几种特殊运算，把二维表进行任意的分割和组合，随机地构造出各式各样用户所需要的表格，即关系。

关系运算有关系代数和关系演算两种表达方式，由于它们的完全等价性，在此只介绍关系代数中的运算。它可以分为以下两类：

① 通常的集合运算，如并、差、交、笛卡儿积等。

② 专门的关系运算，如投影、选择、连接等。

1. 通常的集合运算

(1) 并(Union)

两个关系 $R1$、$R2$ 的并是由属于 $R1$ 或 $R2$ 的元组组成的集合 $R3$。记为 $R3 = R1 \cup R2$。$R1$、$R2$、$R3$ 的属性个数相同，且相应属性分别有相同的值域。

设 $R1$ 和 $R2$ 如表 4.3.1 和表 4.3.2 所示，则 $R1$ 和 $R2$ 的并 $R3$ 如表 4.3.3 所示。

(2) 差(Difference)

两个关系 $R1$、$R2$ 的差是由属于 $R1$ 但不属于 $R2$ 元组组成的集合 $R3$，记为 $R3 = R1 - R2$。$R1$、$R2$、$R3$ 的属性个数相同，且相应属性分别有相同的值域。表 4.3.4 给出 $R1 - R2$ 的结果。

(3) 交(Intersection)

两个关系 $R1$、$R2$ 的交是由既属于 $R1$ 同时又属于 $R2$ 的元组组成的集合 $R3$，记为 $R3 = R1 \cap R2$。$R1$、$R2$、$R3$ 的属性个数相同，且相应属性分别有相同的值域。表 4.3.5 给出 $R1 \cap R2$ 的结果。

(4) 笛卡儿积(Cartesian Product)

设 $R1$ 为 m 元关系，$R2$ 为 n 元关系。$R1$ 和 $R2$ 的笛卡儿积产生一个新关系 $R3$，记做 $R3 = R1 \times R2$。$R3$ 是一个 $(m+n)$ 列元组组成的集合，每一个元组的前 m 列是 $R1$ 的一个元组，后 n 列是 $R2$ 的一个元组，$R3$ 的元组个数等于 $R1$ 的元组个数与 $R2$ 的元组个数的积，其构成方法是用 $R1$ 的每个元组分别与 $R2$ 的每个元组组合成新元组，直到 $R1$ 和 $R2$ 的每个元组两两组合完，即得到了新关系 $R3$ 的所有元组。表 4.3.6 给出了 $R1 \times R2$ 的结果。

表 4.3.1　关系 R1

A	B	C
a	b	c
d	e	f
g	h	i

表 4.3.2　关系 R2

A	B	C
b	d	a
d	e	f

表 4.3.3 关系 $R1 \cup R2$

A	B	C
a	b	c
d	e	f
g	h	i
b	d	a

表 4.3.4 关系 $R1 - R2$

A	B	C
a	b	C
g	h	i

表 4.3.5 关系 $R1 \cap R2$

A	B	C
d	E	f

表 4.3.6 关系 $R1 \times R2$

A	B	C	A	B	C
A	b	c	b	d	a
a	b	c	d	e	f
d	e	f	d	e	f
d	e	f	d	e	f
g	h	i	b	d	a
g	h	i	d	e	f

2. 专门的关系运算

(1) 选择(Selection)

选择运算是从指定关系中选取满足给定条件的元组,组成新的关系。记做 $\sigma_F(R)$。其中 F 为条件表达式,由属性名或列号、常数、算术比较符(<, >, ≤, ≥, =, ≠)及逻辑运算符(NOT, AND, OR)组成,R 为指定的被运算的关系名。选择是在行的方向上进行挑选,为用户提供了构造符合某种条件的原关系子集的方法。表 4.3.7 给出了选择 $R1$ 中属性 A 的值为 g 的元组的结果。

(2) 投影(Projection)

投影运算是从指定的关系中选取所需要的属性,按要求排列组成一个新的关系,新关系的各个属性值来自原关系中相应的属性值,并去掉重复的元组,记为 $\pi_A(R)$。其中 R 为被运算关系,A 为投影表达式,可以是属性名序列,也可以是属性序号的系列。

投影运算在给定关系的列的方向上进行选择。它为用户提供了挑选某些属性并改变它们的排列次序来构造新关系的一种方法。表 4.3.8 给出了选择关系 $R1$ 的 A, C 两列并交换次序所得到的结果。

表 4.3.7 $\sigma_{A=g}(R_1)$

A	B	C
g	h	i

表 4.3.8 $\pi_{C,A}(R_1)$

C	A
c	a
f	d
i	g

(3) 连接(Join)

连接运算是从两个指定关系 $R1, R2$ 的笛卡儿积中选取属性间满足给定条件的所有元

组，组成一个新关系，记为 $R1 \underset{i\theta j}{\bowtie} R2$，其中 $R1$，$R2$ 代表两个不同的关系名；i，j 分别代表 $R1$ 的一个属性和 $R2$ 的一个属性，可以是属性名，也可以是属性序号；θ 表示算术比较符。连接的结果关系的属性个数为两个关系属性个数之和。表 4.3.9 给出 $R1 \underset{B=A}{\bowtie} R2$ 的结果。

表 4.3.9　$R1 \underset{B=A}{\bowtie} R2$

A	B	C	A	B	C
a	b	C	b	d	a

当连接运算中的比较符 θ 为“=”时，称为等值连接；θ 为“<”时称为小于连接；θ 为“>”时，称为大于连接。

在等值连接情况下，如果参与比较的两个关系中用于比较的属性相同，即有公共属性，则此连接称为自然连接，记为 $R1 \bowtie R2$。结果关系中属性个数不再是两关系属性个数之和，而是要去掉重复的属性。显而易见，若在两个关系无公共属性情况下作自然连接运算，实际上等同于求它们的笛卡儿积。表 4.3.12 给出了表 4.3.10 和表 4.3.11 自然连接的结果。

表 4.3.10　关系 $R1$

A	B	C
1	2	3
4	5	6
7	8	9

表 4.3.11　关系 $R2$

B	C	D
2	3	2
5	6	3
9	8	5

表 4.3.12　$R1 \bowtie R2$

A	B	C	D
1	2	3	2
4	5	6	3

4.3.4　关系数据库标准语言——SQL

SQL 是结构化查询语言（Structrued Query Language）的缩写，是关系数据库的标准语言，因具有诸多的优点而备受推崇。国际标准化组织 ISO 已几次公布了 SQL 的国际标准，SQL 语言目前已在各种关系数据库 DBMS 中得到广泛采纳。

1. SQL 语言的特点

SQL 虽然命名为查询语言，但它实际上包括了定义、查询、更新和控制 4 大功能，是一种统一的综合关系数据库语言。SQL 与其他数据库语言相比，有如下特点。

（1）一体化。SQL 将所有的对数据库的定义、查询、更新、控制、维护、恢复、安全等一系列操作要求统一为一种语言，并不像其他数据库语言分为数据库定义语言、数据库操纵语言。

（2）高度非过程化。SQL 是一种非过程化语言，用户只需指出干什么，无须指出怎么干。对数据库存取路径的选择由系统自动完成，从而大大提高了用户编程的效率。

（3）语言简洁，方便易学。SQL 类似英语句子，易学易用，且命令少，便于记忆和使用。

2. SQL 的基本功能

(1) 数据定义。包括定义基本表、视图、索引,以及相应的删除操作,主要命令有 CREATE(建立)、DROP(清除)等。

(2) 数据查询。SQL 查询功能很强,只要数据库中存在的数据,均可通过适当的命令形式查询出来,基本命令是 SELECT,它有许多可选择的形式,包括查询条件的构成,其他语句成分的选择搭配,丰富灵活,可按人们的不同意愿查询出结果。

(3) 数据更新。数据更新是指进行插入、删除、修改数据的操作,它们分别由 INSERT 语句、DELETE 语句及 UPDATE 语句完成。

(4) 数据控制。数据控制包括安全性控制、完整性控制、事务控制和并发控制。由授权(GRANT)、收权(REVOKE)、事务提交(COMMIT)、撤销事务(ROLLBACK)等命令来完成。

SQL 的功能主要是实现数据库的定义和对其数据的操作,优势在于对数据的处理,而对于复杂的计算和复杂的事务处理,还需利用高级语言灵活的表达和强有力的计算功能。因而一般 DBMS 均提供 SQL 嵌入主语言使用的方式,二者融为一体,发挥各自的优势,以满足各种应用要求。

3. SQL 的基本概念

为了后面讨论方便,先介绍 SQL 的一些基本概念。

(1) 基表(Base Table)

基表是在数据库中实际存在的一个独立的关系,每一个基表对应一个物理文件。基表是组成概念模式的实体内容的基本单位。

(2) 视图(View)

视图是从一个或几个基表导出的表,它本是逻辑上存在而物理上并不存在的虚表。因此在数据库中只存在视图的定义,不存储视图的数据。它是外模式或子模式的基本内容。

(3) 列

列就是实体的一个属性,列名即属性名。在 SQL 中不叫属性而叫列。

(4) 游标

游标是用于嵌入式的一种技术,它是为了简化用户编程而将经常查询的内容以固定形式定义,并由系统在内存生成一个相对应的表,供用户程序逐个处理表中的每个元组,而游标即指向当前被处理的元组,相当于一个指示器。

(5) 集函数

又称库函数,是对满足某种限制条件的从基表导出的一个元组集的统计操作。常用的有最大值、最小值、平均值、个数、总和等。

(6) 值表达式

是出现在限制条件或输出要求中的一种表达式,其中可能含有列名、集函数、值说明及算术运算符。值表达式大大增强了表达功能。

(7) 谓词

谓词指明一个条件,该条件求解后,结果为一个布尔值:“真”、“假”或未知。

(8) 子查询

从某一查询结果中再查询所要求的值。子查询可多次嵌套。

为了说明 SQL 的使用方法,后面各节均以图 4.3.1 所示的关系模型和相应的关系作为操作对象。

SQL 语言将数据定义、数据查询、数据更新、数据控制等功能糅合成一个整体的语言,既可以作为独立语言,以联机交互式使用,也可以嵌入到主语言中使用。下面分述各种功能的主要语句形式及使用方法。

学生

学号	姓名	性别	年龄	班级

课程

课程号	课程名	课时

选课

学号	课程号	成绩	教师

学生登记表

学号	姓名	性别	年龄	班级
001	张明	男	18	981
002	李雪	女	19	982
003	陈志	男	18	981
⋮	⋮	⋮	⋮	⋮

课程表

课程号	课程名	课时
001	DB	60
002	OS	68
⋮	⋮	⋮

学生选课表

学号	课程号	成绩	教师
001	001	90	刘
001	002	85	李
002	001	88	刘
⋮	⋮	⋮	⋮

图 4.3.1　关系模型和相应关系的示例

4. 数据定义

SQL 的数据定义功能用来定义关系数据库模式，其主要定义成分是基表、视图和索引，同时还提供了修改基表结构的语句。

(1) 基表的定义和修改

① 定义基表

基表定义涉及基表名、属性名及数据类型、长度等，其语句格式为：

```
CREATE TABLE 表名
    (列名 1 类型 [NOT NULL]
    [,列名 2 类型 [NOT NULL]
    …)
    [其他参数];
```

其中类型可分为：

CHAR(*n*)	表示长度为 *n* 的字符串。
NUMBER(*n*,*m*)	表示长度为 *n*，小数点后为 *m* 位小数的实数。
INTEGER	表示全字长整数。
SMALLINT	表示半字长整数。
FLOAT	表示双精度浮点数。

此外，还有一些其他类型。

选择项[NOT NULL]用以指明相应属性是否可取空值。SQL 支持空值的概念。空值是不知道的值，或不能用的值，任何列均可以有空值，使用了 NOT NULL 则指明该列不能为空值，通常用于关键字属性。

例如对上述学生选课的关系模型可用建表语句定义如下：

```
CREATE TABLE XS(XH CHAR(6)NOT NULL,
    XM CHAR(8),
    XB CHAR(2),
    NL INTEGER,
    BJ CHAR(3));
CREATE TABLE KC(KH CHAR(3)NOT NULL,
    KM CHAR(2),
    KS INTEGER);
CREATE TABLE XK(XH CHAR(6)NOT NULL,
    KH CHAR(3)NOT NULL,
    CJ INTEGER,
    JS CHAR(8));
```

通过上述3个建表语句的执行,则在数据库中建立了学生、课程和选课3个基表结构,表明分别为XS、KC和XK表结构的描述存放在数据库系统的数据字典中,供DBMS运行时使用。

② 删除基表

格式为:DROP TABLE 表名;

该语句用来删除数据库中已存在的一个基表,执行后,表名指出表的全部数据,在其上导出的视图、所创建的索引等将全部被从数据库中清除,并回收空间以作它用。当然表能否被删除、如何删除,不同DBMS可能有不同的约定。

③ 修改基表

语句格式为:

```
ALTER TABLE 表名
    ADD 列名 类型;
```

此语句执行后,在指定的表中增加一个新列。如要增加多列,则只能一列一列地逐步进行,不能利用一个语句完成。增加的列不允许NOT NULL。这里只能增加列,有的系统还提供了对应本语句的减少列的功能。

(2) 定义视图

视图是从一个或几个基表中导出的表,某一用户可以定义若干视图,同一视图也可以为多个用户所共享。一个用户的外模式由若干基本表和若干视图组成。

视图与基表不同,它只是一个虚表,在数据库中只存在视图的定义,不存在对应的数据。但视图一经定义就可以和基表一样被查询,在视图上还可以定义新的视图,对视图的更新有特定的限制,在此不详细介绍。定义视图语句的格式为:

```
CREATE VIEW 视图名[列名[,列名]…]
    AS 子查询
    [WITH CHECK OPTION]
```

例如,从学生表中导出一个20岁以下学生情况的视图,则对应的视图定义语句为:

```
CREATE VIEW V1 AS
    SELECT XH,XM
    WHERE NL<20;
```

当视图不再使用时,可以从数据库中删除,完成此功能的语句格式是:

```
DROP VIEW 视图名;
```

当上面所建的视图V1不再有用,需要从数据库中删除时,可用下面语句:

```
DROP VIEW V1;
```

执行此语句,视图V1的定义即被清除,而由V1视图导出的其他视图也被删除。

(3) 定义索引和删除索引

索引建立在基表上,其语句格式为:

CREATE [UNIQUE] INDEX 索引名 ON

基表名(列名 1[次序][,列名 2[次序]]…)

[其他参数];

若选用 UNIQUE,表示作为索引的列只有唯一的值。索引可以建立在一列或几列上,分别称为单索引和组合索引。当在多列上建立索引时,列名的排列先后指明了索引列的顺序,其中"次序"指明按列取值的升序还是降序排列,分别用 ASC 表示升序,DESC 表示降序,缺省时为升序。

例如,对学生登记表中学号按升序建立一个索引:

CREATE UNIQUE INDEX IND1 ON XS(XH)

该命令执行结果是在表 XS 上定义了一个以学号"XH"为索引列的主索引,UNIQUE 用来保证索引列值的唯一性,此时,任何重复的学号都不能进入数据库中,因此它又具有保护某种数据完整性的作用。

如果已建立的索引不再使用,可用以下命令删除:

DROP INDEX 索引名;

5. 数据操纵

SQL 的数据操纵功能很强,语义也极丰富。数据、操纵功能包括查询、插入、修改、删除 4 大类操作,以下分别介绍。

(1) 数据查询

SQL 最核心的检索语句的基本格式及含义如下:

SELECT [DISTINCT] 选择输出属性表

FROM 检索所涉及的关系名表

[WHERE 检索应满足的条件]

[GROUP BY 分组属性表(HAVING 检索条件)]

[ORDER BY 排序属性表]

其中 SELECT 子句指出需要查找输出的属性值或值表达式;FROM 子句指出 SELECT 子句、WHERE 子句、GROUP BY 子句、HAVING 子句及 ORDER BY 子句所涉及到的属性,所在的关系名;WHERE 子句指出查找必须满足的条件;GROUP BY 子句是把一个关系按某一条件分组,这一条件就是指定的分组属性上的值相同;HAVING 子句与 WHERE 子句作用类似,但 WHERE 是对一个元组的限制条件,而 HAVING 子句是对一个分组的元组限制条件,因此,HAVING 子句只能跟 GROUP BY 子句配合使用;ORDER BY 子句指出查询结果输出时按其后的排序属性的值进行排序,排序属性可以有多个,每个属性可以各自指定不同的升降序排序方式。

SQL 的 SELECT 语句的组成成分很多,按文本标准,可以组合成多种多样的语句形式,

这里只给出一些实例说明它的一般用法,以加深对该语句的理解,更详细的了解请参阅 SQL 文本。

① 简单查询

例 1:列出学生的姓名和班级

```
SELECT XM,BJ
FROM XS;
```

例 2:查阅所有课程的全部情况

```
SELECT *
FROM KC;
```

这里“*”为通配符,表示按 FROM 所指出的表中列的排列次序输出其所有属性的值,“*”代表全部的属性名。

例 3:查找有一门课程成绩在 85 分以上的学生的学号和课程号

```
SELECT DISTINCT XH,KH
FROM XK
WHERE CJ>85;
```

此处,WHERE 子句后的检索条件包括比较操作符 =,<,>,<>,>=,<=及逻辑运算符 AND,OR,NOT。

例 4:查找还未确定课时的课程

```
SELECT KH,KM
FROM KC
WHERE KS IS NULL;
```

例 5:按成绩从高到低顺序列出选修数据库课程的学生的学号及成绩

```
SELECT XH,CJ
FROM XK
WHERE KM='DB'
ORDER BY CJ DESC;
```

② 含谓词的查询

SQL 文本中有很多谓词,如 IN,LIKE,BETWEEN,EXIST 等,它们都可用于 SELECT 中组成不同形式的查询语句,以下用例子介绍几种。

例 1:查学习成绩在 80 分和 90 分之间的学生学号

```
SELECT DISTINCT XH
FROM XK
WHERE CJ BETWEEN 80 AND 90;
```

BETWEEN 相当于一个闭区间,因此,这里包括 80 分和 90 分的学生在内。

例 2:查找所有姓“王”的学生的情况

```
SELECT *
FROM XS
WHERE XM LIKE"王%"
```

LIKE 的一般格式是:

属性名 LIKE 字符串常量

其中属性名的类型必须是 CHAR,常量字符的含义如下:

- 短横线“-”表示在此位的一个字符。
- 百分号字符“%”代表任意字符串(含空串)。
- 其余字符代表自身。

LIKE 谓词非常有用,可以找出记不太清楚的一些信息,即可以根据记得的某些符号进行模糊查询,这样可能得到一批信息,再从这些信息中确定所要求的信息,这是相当方便的。

还有一些谓词,在此就不一一介绍。

③ 连接查询

关系数据库通过数据的自然状态发生联系,当所需的信息分散在多个表中时,则要进行表的连接查询,从而得到想要的结果。

例 1:查找成绩不及格的学生姓名及不及格课程号

```
SELECT  XM、KH
FROM    XS、XK
WHERE   (CT>60) AND (XS、XH = XK、XJ);
```

此例是一个等值连接,就是找与指定属性(这里具体是学号属性 XH)值相等的元组,这要从学生和选课这两个表中找数据,因此在 FORM 子句中指出两个表名,而在 WHERE 中指出等值的属性名。所以又称这种条件为连接条件或连接谓词。对此还有几点说明如下:

- 连接谓词中的属性必须都是数字型或字符型。
- 不要求属性相同,但实际上往往是相同的。
- 除了连接谓词外,还可包含其他条件。
- 不要求连接的属性出现在结果关系中。
- 连接可涉及到任意多个关系。

等值连接的特例是自然连接,它在下述方面比等值连接限制严格:

- 具有相同的属性名。
- 具有相同的属性类型。
- 结果关系中不含重复属性,即自动删除一个相同的属性。

实际上,SQL 中不严格区分等值连接、自然连接或其他的连接,只是严格把握连接条件,符合条件的则取之,否则弃之。

例2:查找选修了数据库课程(DB)的学生的信息和该课程的信息

```
SELECT XS.*,KC.*
FROM XK,XS,KC
WHERE XS.XH = XK.XH AND KC.KH = XK.KH AND KM ='DB';
```

④ 嵌套查询

嵌套查询指包含子查询的检索,即 WHERE 子句中又包含 SELECT 子句,这种嵌套结构,加强了 SQL 的表达功能,它使得一些给不出具体确定条件,往往要通过对数据库的查询才能确定这些条件的应用要求可以得到解决。

例1:查找选修了001号课程且成绩在80分以上的学生的学号,姓名

```
SELECT XH,XM
FROM XS
WHERE XH IN
(SELECT XH
FROM XK
WHERE(KH ='001')AND(CJ>80));
```

例2:查找选修了数据库课程的学生的姓名、班级

```
SELECT XM,BJ
FROM XS
WHERE KH IN
    (SELECT XH
    FROM XK
    WHERE KH IN
            (SELECT KH
            FROM KC
            WHERE KM ='DB'));
```

本例是一个三层嵌套查询,一般嵌套可以多层,嵌套层数由具体 DBMS 实现者决定。

⑤ 计算查询

SQL 为某些简单常用的计算、统计提供了专用的内部函数,称为集函数,在查询语句中配合使用,使得在查询的过程中同时对某属性上的值进行计算或统计,以提高系统的检索能力。通常有下列内部集函数:

COUNT	统计某属性上所有值的个数。
SUM	统计指定元组集中某属性上所有值的总和。
AVG	统计某属性上所有值的平均值。
MAX	统计某属性上所有值的最大值。

MIN 统计某属性上所有值的最小值。

函数 SUM 和 AVG 所涉及的属性必须是数值型的。如果要消除重复值,可以在函数变量前加上 DISTINCT。对 MAX,MIN 不需加,而对 COUNT 必须加。还应指出:函数集不能嵌套使用,即集函数的自变量不能是集函数。

例 1:查选修了 001 号课程的学生的总人数

```
SELECT COUNT(XH)
FROM XK
WHERE KH = '001';
```

例 2:查 002 号课程的总平均成绩

```
SELECT AVG(CJ)
FROM XK
WHERE KH ='002';
```

例 3:查出每个人所选课的平均成绩

```
SELECT AVG(CJ)
FROM XK
GROUP BY XH;
```

(2) 数据更新

数据更新包括插入、删除和修改 3 种操作,分别由 INSERT、DELETE 和 UPDATE 语句实现。

① 数据插入

数据插入语句的功能是把元组插入到关系中。其语句格式为:

```
INSERT INTO 关系名[(属性名[,属性名]…)]
    VALUES(常量[,常量]…)
```

属性名序列为要插入值的属性名集合,常量序列为要插入的对应值,第 i 个属性对应于第 i 个常量。若省略属性名,则表示属性值按顺序对应于该关系的原定义的属性名。

例:插入一个学生的信息到学生关系 XS 中

```
INSERT INTO XS(XH,XM,XB,NL,BJ)
    VALUES('005','刘畅','男',18,'982')
```

通常如果插入的是一个关系的全部属性值,则属性名可以省缺;若嵌入的是关系的部分属性值,则必须列出相应属性名,此时,该关系中未列出的属性名取值为空值。

SQL 还允许从一个关系中选一些元组插入到另一个关系中去,这样则可进行批量插入。但此时更要注意值与关系中属性的对应关系。最简单的是参与操作的两个关系具有完全相同的结构,利用此语句可很方便地完成数据从一个关系向另一个关系的全拷贝。

假设与学生关系 XS 结构相同的关系 S,希望复制 XS 关系中的数据,则可用下面的语句

完成。

```
INSERT INTO S
SELECT *
FROM XS;
```

② 数据删除

数据删除语句的功能是从指定的一个关系中删除一个或多个元组。其语句格式为:

```
DELETE FROM 关系名
[WHERE 检索条件]
```

其中 WHERE 子句为任选项,若不选用,则表示删除指定关系的全部元组,但结构仍然存在;若选用,则表示删除满足给定条件的部分元组。

例 1:删除学号为 001 的学生的信息

```
DELETE FROM XS
WHERE XH ='001';
```

上例是删除单个元组,还可以一次删除多个元组。

例 2:删除选修 001 号课程的学生选课登记

```
DELETE
FROM XK
WHERE KH ='001';
```

③ 数据修改

修改语句的功能是改变一给定关系中属性的值,其语句格式为:

```
UPDATE 关系名
SET 属性名 = 表达式[,属性名 = 表达式]…
[WHERE 检索条件]
```

WHERE 子句为任选项,若不选用,则表示修改关系中所有元组;若选用,则表示仅修改满足给定条件的那些元组。

例 1:将学号为 001 的学生选修的 002 号课程成绩改为 90 分

```
UPDATE XK
SET CJ = 90
WHERE(XH ='001')AND(KH ='002');
```

本例修改的是单个元组,还可一次修改多个元组。

例 2:将所有学生的所有选修的课程的成绩提高 5%

```
UPDATE XK
SET CJ = CJ * 1.05;
```

(3) 视图的操作

视图是从一个或几个基表导出的表,也可以从视图导出视图。一个用户可以定义一个或多个视图,同时经授权后,一个视图也可以为多个用户共享。但视图是一个虚表,只有定义没有对应的物理数据。但视图一经定义就可以和基表一样被查询、删除,还可以用来定义新的视图,但对更新操作(如增加、删除、修改)有一定的限制。

① 视图的查询

视图的查询格式和基表查询一样,由于在定义视图时已经根据需要进行了限制,故在查询视图时,不必再写这些条件。如果所加的条件不是定义视图所用的限制条件,则必须加在后面,用显式表达。

② 视图的更新

对视图的更新最终被转换成对基表的更新。一般只允许对单个基表导出的视图进行更新,并有如下限制:

- 若视图的字段由表达式或常数组成,则不允许对此视图执行 INSERT 和 UPDATE 操作,只可执行 DELETE 操作。
- 若视图的字段由集函数组成,则不允许更新。
- 若视图定义中有 GROUP BY 子句,DISTINCT 选项,则不允许更新。
- 若视图定义中有查询,且内外层的 FROM 子句中的表是相同的,则不允许更新。
- 由多个基表导出的视图和不允许更新的视图导出的视图不允许更新。

(4) 数据控制

SQL 数据控制功能是为了保护数据库中数据的安全,防止不合法的存取和破坏。这是通过控制用户存取数据的权限实现的。SQL 的安全保护可以从一个指定的数据值到一个表。一个用户对不同的表有不同的存取权力,不同的用户对同一个表有不同的存取权力。这些权力是由 DBA 授予的,DBMS 保证这些权力的实施。SQL 对权力控制提供了 GRANT(授权)和 REVOKE(收权)两个语句。

① 授权

授权语句格式为:

GRANT 权力[,权力]…[ON 对象类型 对象名]

TO 用户[,用户] …

[WITH GRANT OPTION]

该语句的功能是将对某些对象的某些操作权力授给特定的用户。

对不同类型的操作对象有不同的操作权力,例如,对基表有修改和建立索引的权力;对数据库有建表的操作权力,拥有此权的用户不仅可以建基表,还拥有对此表的一切操作权利。

WITH GRANT OPTION 子句的作用是使获得某种权力的用户可将该权力再授给别的用户。

可由 GRANT 语句指定的权限有 SELECT, INSERT, DELETE, UPDATE, INDEX, ALL PRIVILIGES, DBA 等。

下面举例说明该语句的用法。

例 1:把对学生表(XS)的查询、插入权授予用户 U1

GRANT SELECT, INSERT ON TABLE XS TO U1

例 2:把对 XS, KC, XK 3 张表的全部权力授给用户 U2、U3

GRANT ALL PRIVILIGES ON TABLE XS, KC, XK TO U2, U3;

全部权力包括查询、修改、插入、删除等。

例 3:把对学生选课表(XK)的查询权授给所有用户

GRANT SELECT ON TABLE XK TO PUBLIC

例 4:把对学生表(XS)的修改权授予用户 U1 并允许他转授予给别人

GRANT UPDATE ON TABLE U1

WITH GRANT OPTION;

② 收权

收权语句的格式为:

REVOKE 权力[,权力]…[ON 对象类型 对象名]

FROM 用户[,用户]…

该语句的功能是将授予用户的权力撤销,是 GRANT 语句的逆操作,用法也与 GRANT 基本相同,在此不再详述。

(5) SQL 的嵌入式使用

SQL 提供两种使用方式:一是作为独立语言以交互方式在终端上使用;二是作为宿主语言嵌入到某种主语言中使用。

交互式不用编程,用户输入 SQL 命令,机器执行并反应结果,通过人机交互完成某些应用。交互式接口各具特色,但都是以菜单的形式让用户使用。通常包含一个屏幕/对话管理程序及各种命令。这种方式对简单的小型应用比较方便。

嵌入式是将 SQL 语句与高级语言一起混合编程,如嵌入到 C、PASCAL、FORTRAN、COBOL 等,这种方式下高级语言通常称为主语言,这样充分利用高级语言的极强的表达计算功能和 SQL 的数据处理功能,来完成用户复杂的事务处理。

使用 SQL 嵌入主语言时要注意下述几个基本问题。

① 标识 SQL 语句

使用 SQL 嵌入主语言时,通常的处理方法是先由预编译机制将 SQL 语句转换成等价的主语言执行语句,然后由主语言编译程序处理并生成目标程序,然后运行。因此必须在 SQL 语句前加标记以便预编译识别,同时,SQL 语句结束时也必须加上标志。不同的系统采用的标记可能不同,常用的为添加“EXEC SQL”。

② 数据库工作变量与主语言程序变量间的通信

在这种混合编制的程序中,SQL 语句可能涉及两类变量,一是数据库模式中定义了的属性变量,它不需在程序中再定义,而可由 SQL 语句直接使用;另一类是程序中定义的主语言工作变量,简称主变量。主变量在使用前必须加以说明,并在其前面冠以冒号":"或其他特定符号(如"#"号),以区别于属性变量。

③ 通过游标(Cursor)协调主语言与 SQL 语句的处理能力

主语言一般一次只能处理一个记录值,而 SQL 语句通常一次处理一批元组值。因而必须注意到两种语言处理能力上的差异,并设法协调它们,SQL 提供游标机制来解决这个问题。

前面已经介绍了游标的概念,它是用于嵌入式的专门技术,基本思想是将经常查询的内容以固定形式定义,由系统将 SQL 语句执行后产生的一批元组在内存生成一个相对应的表,由游标指向当前被处理的元组值,起指示器的作用,供用户程序逐个地处理表中的每个元组。

对于游标有 4 个相关的 SQL 语句,它们是:

- 定义游标语句(DECLARE 游标名 CURSOR)。
- 打开游标语句(OPEN 游标名)。
- 拨动游标语句(FETCH 游标名)。
- 关闭游标语句(CLOSE 游标名)。

下面给出一个 SQL 嵌入 C 语言的程序,其中包含了上述 3 个问题的一些相关内容。嵌入式 SQL 的应用可以参看 SQL 标准和具体 DBMS 的使用手册等相关资料。

```
1  EXEC SQL BEGIN DECLARE SECTION;          ┐
2  Char empno(6),name[20];                  ├ 定义主变量
3  Short age, sex;                          ┘
4  EXEC SQL END DECLARE SECTION;
5  EXEC SQL DECLARE C1 CURSOR FOR           ┐
6       SELECT EMPNO,NAME,AGE,SEX           ├ 定义游标
7       FROM EMP                            ┘
8       WHERE DEPTNO = 1;
9  EXEC SQL OPEN C1;
10 EXEC SQL WHENEVER NOT FOUND GOTO aaa;      出错处理
11 For(;;){
12    EXEC SQL FETCH C1 INTO :empno,:name :age,:sex;      拨动游标
13    printf("NO :%S, Name :%S, AGE :%d, SEX %d",
14        empno, name, age, sex);
```

```
15  }
16  aaa :EXEC SQL CLOSE C1;        关闭游标
```

上面的例子中,1~16是为了说明方便加的语句编号,实际编程时是不需要的。假定数据库模式中定义了一个职员表EMP,含有4个列:职员编号、姓名、年龄、性别,分别命名为EMPNO,NAME,AGE,SEX。

1~4行定义SQL语句要使用的主变量:empno,name为字符型,age,sex为数字型,这些主变量与数据库职员表中的列名同名是允许的,且它们分别对应的类型相同。

5~8行定义了一个游标,命名为C1,实际在内存生成一个表,包含EMPNO、NAME、AGE、SEX4个列。

9行打开游标C1,如果要对其中元组操作,必须要先打开。

10行是例外处理。

11~15行是一个循环语句,对游标中的每一个元组逐个取出并打印,第12行语句是把指示器所指的一个元组的4个列的值分别装入主变量empno、name、age、sex中,由于是使用SQL的FETCH语句,故在前面都冠以冒号":";第13行不是SQL语句,而是C语言的打印语句,虽然它们使用了empno、name、age、sex 4个变量,但它们的前面都不能加冒号":",随着循环的执行,指示器逐个下移,使每个元组都打印出来,直到游标中的所有元组都处理完。

16行关闭游标,这是在处理完游标中全部元组后规定要执行的操作。

4.4 数据库应用系统的设计

数据库应用系统的设计就是要面向一个给定的应用环境创建一个能满足用户使用要求的数据库,并基于该数据库开发一套能完成用户业务活动中各种事务处理的应用程序。这些程序和数据库要在一个所选用的DBMS支持下工作,并且具有良好的性能。这里核心的问题是数据库的设计,数据库设计的优劣将直接影响应用系统的质量和效果。设计一个结构优良的数据库是对数据进行有效管理和产生正确信息的保证。数据库系统的复杂性和它与应用环境的密切联系,要求设计者既要懂计算机专业知识,又要懂应用领域的专业知识,涉及的因素很多。因此,数据库设计是一个困难、复杂和费时的过程。

4.4.1 数据库设计内容及特点

数据库设计包括结构特性设计和行为特性设计两方面的内容。

结构特性设计是指构造数据库的数据模型。该模型要反映实际应用中需要的数据及数据间的联系,并在满足不同用户数据需求的前提下尽可能减少冗余,实现数据共享。结构特

性是静态的,一旦形成就相对稳定,轻易不再变动。当然,也不可能一成不变,要留有扩充余地,以满足发展的需要。

行为特性设计是指确定用户的业务活动,也就是用户对数据库应用的行为和动作。具体体现在应用程序中,用户通过应用程序访问和操作数据库,用户的行为与数据库结构紧密相关,所以行为特性设计主要是应用程序的设计。

数据库设计过程比较复杂且工作量较大,是一项细致且技术性很强的软件工程,应采用软件工程的方法和工具。一般的软件工程比较强调行为特性的设计,而在数据库设计中,由于数据库结构是一个相对稳定的并为所有用户共享的数据基础,结构设计成为关键,所以更强调结构特性的设计。

数据库设计是一个"反复探寻,逐步求精"的过程,一般不可能一次实现。在数据库设计过程中,结构特性设计和行为特性设计通常是相互参照进行的。图 4.4.1 描述了结构特性设计和行为特性设计的各个步骤及两者的关系。

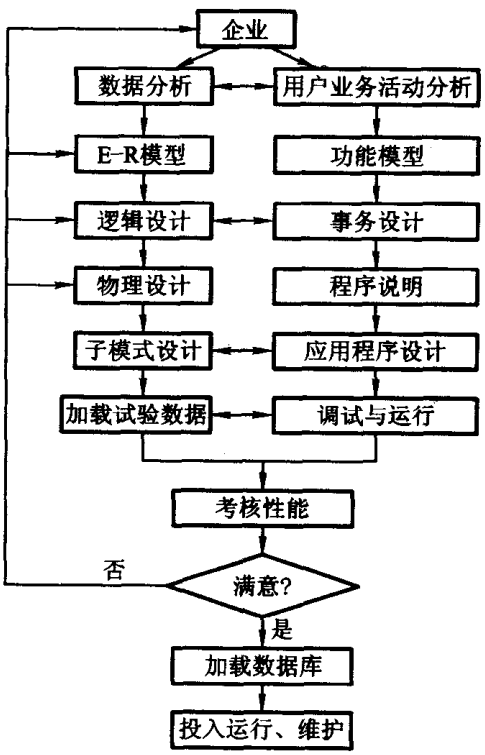


图 4.4.1 数据库设计中两种特性的对照

4.4.2 数据库设计步骤

在此,以关系数据库为基础讨论数据库应用系统设计。一般将数据库及其应用系统设计分为以下6个阶段,如图4.4.2所示。

(1) 需求分析。即准确了解与分析用户需求,这是整个数据库设计的基础。

(2) 概念结构设计。对用户的需求进行综合、归并与抽象,形成一个独立于具体DBMS的概念模型。

(3) 逻辑结构设计。将得到的概念模型转换为关系模型,即按关系数据库的要求转化。

(4) 物理结构设计。选择数据库使用的物理结构,包括存储结构和存取方法等。

(5) 数据库实施。根据逻辑设计和物理设计的结果建立数据库,编制与调试应用程序,组织入库,并进行试运行。

(6) 数据库运行和维护。数据库系统正式运行中的日常维护。

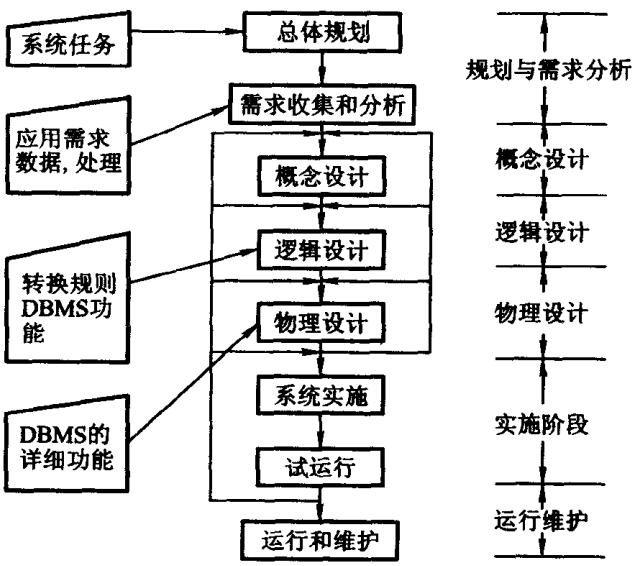


图4.4.2 数据库设计步骤

4.4.3 需求分析

需求分析是整个数据库设计过程中最重要的步骤之一,是后续各阶段的基础。它对系统的整个应用情况进行全面、详细的调查,收集支持系统总设计目标的基础数据和对这些数据的处理要求,确定用户的需求。这一步犹如盖房子打基础,质量如何将直接关系到数据库设计的成败,必须予以高度重视。它包括收集资料、分析整理、数据流图、数据字典、用户确

认等几项工作。

进行需求调查的基本方法是听取数据库应用部门工作人员的报告，并与之座谈；请专人介绍，询问，设计调查表请用户填写；设计人员亲自查阅原始资料及跟班作业等。

1. 收集资料

耐心细致地了解现行业务处理流程、对新系统的要求、收集全部数据资料，如报表、合同、档案、单据、计划等。

用户需求主要包括以下 3 方面：

(1) 信息需求。即用户要从数据库获得的信息内容。信息需求定义了新系统应该提供的所有信息，应描述清楚系统中数据的性质及其联系。

(2) 处理需求。即完成什么处理功能和采用什么处理方式。处理需求定义了新系统数据处理操作，应描述操作执行的场合、频率、操作对数据的影响等。

(3) 数据约束。即应用对数据的特殊要求，主要是以下几个方面。

① 安全保密性要求。主要指各个用户对各类不同数据所拥有的不同操作权限。

② 数据完整性约束。主要指数据正确性的约束范围和验证准则，以及一致性保护的要求。

③ 响应时间要求。主要指某些特定应用对数据存取时间的限制。

④ 数据恢复。主要指数据转储及恢复的时机与范围等要求。

2. 分析整理

对前面调查的结果进行分析整理，确定以下内容。

(1) 系统目标(总的目的，战略性目标与具体要求)

建立此系统的目的、任务，对数据处理的要求以及各外部要求，如数据保密性、使用范围、查询速度、人机交互的要求等。

(2) 系统功能结构

从组织机构、各部门职责、业务处理等方面出发，分析各个部门完成什么工作，输入和使用什么数据，如何加工处理这些数据，输出什么信息，输出到什么部门，输出结果的格式是什么等，以确定系统的基本功能结构。

(3) 系统支持范围

确定哪些功能由计算机完成或准备让计算机完成，数据库应支持哪些应用功能。该范围的确定应尽可能地考虑较广泛的应用部门以及有效地利用计算机设备和数据库系统的能力，同时还应适当考虑以后扩展的要求。

分析的过程中还要对所收集到的数据进行抽象。抽象是对实际事物或事件的人为处理，抽取共同的本质特性，忽略细枝末节，并精确地加以描述，组成某种模型。

数据分析与抽象是数据库设计的基础，数据分析和抽象可以同时进行，并可从局部入手。例如，在库存清单中有下面几个数据：

零件号 零件名称 数量 单价 总价

其中,总价 = 单价 × 数量,所以总价是通过计算得到的二次数据,不是基本数据元素,不必作为基本数据项在数据库里存储。在下面的数据流图和数据字典中出现的数据,都可在这里作初步的分析整理,或者结合在数据流图和数据字典中进行。像这样对所收集到的数据逐个进行筛选和认定需要很大的工作量。

了解、分析了用户需求后,还需要进一步表达用户的需求。在众多的分析和表达用户需求的方法中,结构化分析(Structured Analysis,简称 SA 方法)是一个简单实用的方法。SA 方法用自顶向下,逐层分解的方式分析系统,用数据流图(Data Flow Diagram,简称 DFD)描述事务处理过程,借助数据字典(Data Dictionary,简称 DD)描述系统中的数据。

3. 数据流图

数据流图可以形象地描述事务处理与所需数据的关联,便于用结构化分析方法,自顶向下,逐层分解,步步细化。在数据流图中,用命名的箭头表示数据流,用圆圈表示处理,用矩形或其他形状表示数据存储。

按照 SA 方法,可将任何系统抽象为图 4.4.3 所示的数据流图。

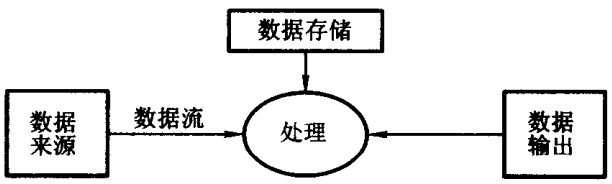


图 4.4.3 数据流图

图 4.4.3 给出的只是最高层次抽象的系统概貌,要反映更详细的内容,可将处理功能分解为若干子功能,每个子功能还可以继续分解,直到把系统工作过程表示清楚为止。在逐步分解的同时,它们所用到的数据也逐级分解,形成若干层次的数据流图。

4. 数据字典

数据流图给出了系统数据流向和加工等动态情况,但其各个成分的具体含义仍然不清楚或不明确。因此,在实际中常采用数据字典这一基本工具对其作进一步的详细说明。

数据字典和数据流图密切配合,能清楚地表达数据处理的要求 。数据字典用于对数据流图中出现的所有成分给出定义,它使数据流图上的数据流名字、加工名字和数据存储名字具有确切的解释。

数据字典将从原始的数据资料中分析整理出下述数据信息:数据元素的名称、同义词、性质、取值范围、提供者(来源)、使用者(去向)、控制权限、保密要求、使用频率、数据量、数据之间联系的语义说明、各个部门对数据的要求及数据处理要求。通过细致的调查研究,对组织的各种数据需求进行详细描述,并由此产生这个组织的数据字典。

数据字典通常包括数据项、数据结构、数据流、数据存储和处理过程5个部分。其中数据项是数据的最小组成单位，若干个数据项可以组成一个数据结构，数据字典通过对数据项和数据结构的定义来描述数据流、数据存储的逻辑内容。可见，数据字典是关于数据库中数据的描述，即元数据，而不是数据本身。

(1) 数据项

数据项是不可分割的数据单位。对数据项的描述通常包括以下内容：

数据项描述 = {数据项名，数据项含义说明，别名，数据类型，长度，取值范围，取值含义，与其他数据项的逻辑关系，数据项之间的联系}

其中“取值范围”、“与其他数据项的逻辑关系”(例如该数据项等于另几个数据项的和，该数据值等于另一数据项的值等)定义了数据的完整性约束条件，是设计数据检验功能的依据。

(2) 数据结构

数据结构反映了数据之间的组合关系。一个数据结构可以由若干个数据项组成，也可以由若干个数据结构组成，或由若干个数据项和数据结构混合组成。对数据结构的描述通常包括以下内容：

数据结构描述 = {数据结构名，含义说明，组成：{数据项或数据结构}}

(3) 数据流

数据流是数据结构在系统内传输的路径。对数据流的描述通常包括以下内容：

数据流描述 = {数据流名，说明，数据流来源，数据流去向，组成：{数据结构}，平均流量，高峰期流量}

其中“数据流来源”是说明该数据流来自哪个过程。“数据流去向”是说明该数据流将到哪个过程去。“平均流量”是指在单位时间(每天、每周、每月等)里的传输次数。“高峰期流量”则是指在高峰时期的数据流量。

(4) 数据存储

数据存储是数据结构停留或保存的地方，也是数据流的来源和去向之一。它可以是手工文档或手工凭单，也可以是计算机文档。对数据存储的描述通常包括以下内容：

数据存储描述 = {数据存储名，说明，编号，输入的数据流，输出的数据流，组成：{数据结构}，数据量，存取频度，存取方式}

其中“存取频度”指每小时或每天或每周存取几次、每次存取多少数据等信息。“存取方式”包括是批处理还是联机处理；是检索还是更新；是顺序检索还是随机检索等。另外，“输入的数据流”要指出其来源，“输出的数据流”要指出其去向。

(5) 处理过程

处理过程的具体处理逻辑一般用判定表或判定树来描述。数据字典指只需要描述处理过程的说明信息，通常包括以下内容：

处理过程描述 = {处理过程名,说明,输入:{数据流},输出:{数据流},处理:{简要说明}}

其中,“简要说明”主要说明该处理过程的功能及处理要求。功能是指该处理过程用来做什么(而不是怎么做),处理要求包括处理频度要求,如单位时间里处理多少事务、多少数据量、响应时间要求等。这些处理要求是后面物理设计的输入及性能评价的标准。

现以学生学籍管理子系统为例,简要说明如何定义数据字典。该子系统涉及很多数据项,其中“学号”数据项可以描述如下。

数据项: 学号
含义说明: 唯一标识每个学生
别名: 学生编号
类型: 字符型
长度: 8
取值范围: 00000000 ~ 99999999
取值含义: 前两位标识该学生所在年级,后六位按顺序编号
与其他数据项的逻辑关系: ……

“学生”是该系统中的一个核心数据结构,它可以如下描述。

数据结构:学生
含义说明:是学籍管理子系统的主体数据结构,定义了一个学生的有关信息
组成: 学号,姓名,性别,年龄,所在系,年级

数据流“体检结果”可如下描述。

数据流: 体检结果
说明: 学生参加体格检查的最终结果
数据流来源: 体检
数据流去向: 批准
组成: ……
平均流量: ……
高峰期流量: ……

数据存储“学生登记表”可如下描述。

数据存储: 学生登记表
说明: 记录学生的基本情况
流入数据流: ……

流出数据流： ……
组成： ……
数据量： 每年 3 000 张
存取方式： 随机存取

处理过程“分配宿舍”可如下描述。

处理过程： 分配宿舍
说明： 为所有新生分配学生宿舍
输入： 学生，宿舍
输出： 宿舍安排
处理： 在新生报到后，为所有新生分配学生宿舍。要求同一间宿舍只能安排同一性别的学生，同一个学生只能安排在一个宿舍中，每个学生的居住面积不小于 3 平方米。安排新生宿舍的处理时间应不超过 15 分钟。

为节省篇幅，这里省略了数据字典中关于其他数据项、数据结构、数据流、数据存储、处理过程的描述。

5. 用户确认

需求分析阶段形成的文档资料是需求说明书，主要内容包括：数据库应用功能目标、不同用户视图范围、数据量统计、数据约束、各项业务的数据流程图及有关说明、对各类数据描述的集合，即数据字典。数据字典在需求分析阶段建立，在整个数据库设计的各个阶段将不断修改、充实和完善。

需求分析得到的需求说明书必须返回用户，并且用非专业术语与用户交流。在反馈时，设计者与用户一起检查与修改那些没有如实反映现实世界的错误或遗漏、修改 DFD 图、补充数据字典等，与用户交流的过程可能需要反复多次，最终要得到用户的认可。

4.4.4 概念设计

这一阶段要把需求分析得到的用户需求抽象为信息世界的结构，即设计出概念模型。通常用 E－R 模型来表达。其步骤是先设计出局部 E－R 图，然后集成为全局 E－R 图。

1. 局部 E－R 图的设计

局部 E－R 图可从数据流图出发确定实体、属性和标识实体的码，并根据数据流图中表示的对数据的加工，确定实体之间的联系及其类型。

(1) 选择实体

选择实体的任务是在确定的局部范围内选择一些合适的信息单位为局部信息结构的基本实体。这些实体应能满足该局部范围内的基本应用需求。

通常，以实际工作中习惯使用的信息单位为选择实体的基础。因此，数据字典中的数据结构是选择实体的直接依据。如在通常的经营业务管理中，设计人员可以选择买方、卖主、

商品、合同及仓库作为信息结构的初始实体。当然,基于习惯应用所选择的实体可能是不合适的,这在后续的设计过程中可逐步得到修正。

实体选择时最大困难是如何区别实体与属性。在应用环境中,从不同的应用观点出发,同一个信息单位,有的视为实体,有的视为属性。此时,应根据应用的各种不同需求,分清主次,灵活处理。例如,买方和卖方可作为合同实体中的属性,也可单独作为实体。如果应用中不仅仅需要买方和卖方的名称,还需要诸如地址、电话、法人及银行账号等多种信息,且日常应用中经常查询这些信息,则应将买方和卖方独立作为实体,并将这些信息归入买方和卖方实体之中。

进行实体选择时,若某些实体的信息内容及主要处理共性较大,则可将它们合并为一个实体。如前面设计的买方、卖方实体,其基本信息内容和使用一般差别不大,则可合并为一个命名为"供销单位"的实体。如果应用时需要区别买方与卖方,则可在实体选择时另设一标识属性来区别。

实体确定后,应给每一实体命名。在同一局部信息中的实体名称应具有唯一性。

(2) 确定实体的关键字属性(即码)

实体的存在依赖于其关键字的存在。实体关键字属性确定后,实体的非关键字属性也就容易确定了,因为它们之间总存在某种应用上的联系和依赖性。选择关键字属性除了考察其是否能唯一标识实体外,还应注意实际应用中的习惯与方便。

(3) 确定实体的属性

一般来说,"属性"必须是不可分的数据项,不再具有需要描述的特性属性,可分为标识属性和说明属性两类。标识属性用作实体的关键字,说明属性用作描述实体的一般特征。标识属性已在前述步骤中确定。这里的任务是确定各实体内的说明属性,并给它们命名。

决定说明属性的基本原则是,说明属性应从功能上从属于标识属性。这就是说,属性分配到某实体内是否合理,取决于该属性是否可通过其所在实体的关键字找到,并且它们之间在应用中具有某种联系。如它们往往在一起使用或用于生成同一种输出。

说明属性应是单值的,即不允许嵌套属性和重复组的现象出现在实体中。

实施属性按功能从属的分配办法,有可能发生同一说明属性从属于多个实体的标识属性的情况。此时,可根据数据流图,将属性分配到其使用频率最高的实体中去。

依照上述方法,进行反复的综合调整,各实体及其属性基本确定后,数据字典中的数据项中仍可能剩余少量数据项。此时,这些剩余数据项若仍有使用价值,则可进一步按照功能从属的方法将它们分配到依赖性较强的那些实体中去,或者去定义新的实体及联系,从而充分保证信息的完整性。

(4) 确定联系

数据库系统的重要特征之一是,它不仅管理数据本身,而且还管理数据间的联系。这种联系必须在概念设计时由设计人员定义。

一种常用的分析实体间是否存在联系的方法是，将局部范围内的实体逐一取出，与该范围内的其他实体试匹配，考察能否找到与参与试匹配的两个实体都有关的问题，或同一任务同时使用到参与试匹配的两个实体。若存在这种情况，则认为它们之间存在某种联系。这种方法可称之为试匹配法。

若两实体间发现存在一个联系实体，则可确定联系的类型(1:1，1:n，n:m)并给联系命名。

根据以上确定的信息，就可以画出局部 E－R 图了。图 4.4.4 表示图书管理中几种应用对应的 E－R 图。

构造局部 E－R 图过程，还要逐一考察它们是否能满足数据流图对数据的要求。例如，图 4.4.4(a)一个实体“图书”就已经能满足查询图书的要求；类似的，图 4.4.4(b)中的一个实体“读者”就可以满足办理借书证的处理要求；图 4.4.4(c)反映了“读者”与“图书”两个实体间的联系，根据这种联系就能满足“查询过期读者名单”及“读者所借图书”等处理要求。

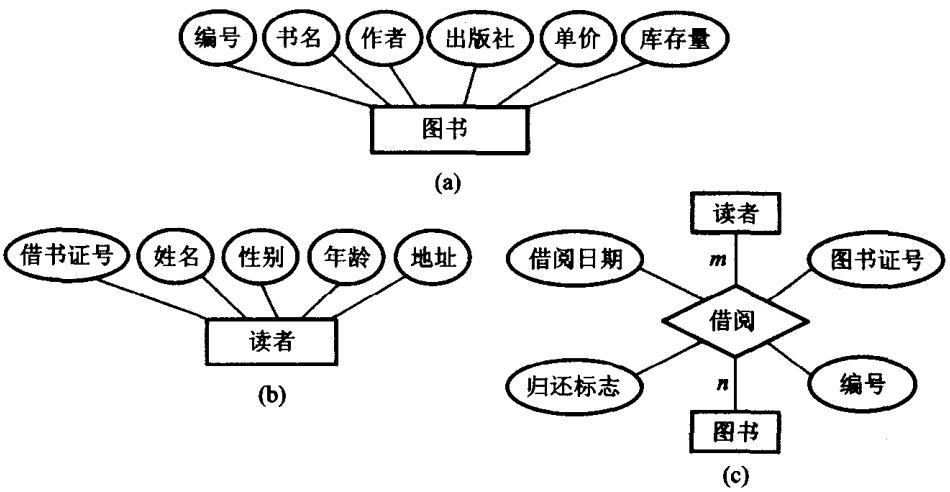


图 4.4.4 局部 E－R 图

2. 全局 E－R 图的设计

各个应用的局部 E－R 图设计好以后，对它们加以综合汇总来产生全局 E－R 图。由于局部 E－R 图面向的应用不同，且往往由不同人员设计，必然存在许多冲突，如属性冲突、命名冲突、结构冲突等。因此综合汇总不能把局部 E－R 图简单堆积到一起，而是要致力于解决这些冲突，并且尽量消除不必要的冗余数据和联系，以形成一个所有用户都理解和接受的统一概念模型。

冲突主要表现在命名冲突、属性冲突及结构冲突 3 个方面。

(1) 命名冲突是指不同的局部信息中用同一名称表示不同数据对象，或用不同的名称

表示同一个数据对象，即存在同名异义或异名同义冲突。

如对科研项目，财务科称为项目，科研处称为课题，生产管理处称为工程等，这是异名同义；又如在图书馆管理子系统中，都叫“名称”，但是对书籍应改为“书名”，对部门应改为“部门名称”，以免相重，这属于同名异义。

命名冲突可能发生在实体、联系上，也可能发生在属性一级上。其中属性的命名冲突更为常见。命名冲突的处理方法是经过讨论、协商，对其重新命名。

（2）属性冲突是指不同局部信息结构中对相同属性的数据类型、值域要求不同。如属性“时间”，有的要求用字符串，有的要求用整数，有的要求用日期型，有的要求精确到年、月、日，有的要求精确到年或月等。

属性冲突的解决方法通常是：取尽可能包含较多局部信息结构要求的数据类型或值域作为该属性的数据类型或值域。如上述的属性“时间”可采用日期型，它可以精确到年、月、日，少数不同应用要求的用户可以通过数据库管理系统提供的语句或函数实现数据类型及值域的转换。

（3）结构冲突主要有两种类型。其一是同一数据对象，有的作为实体，有的作为属性；其二是相同实体间的联系在不同信息结构中联系方式不同。

对于第一类结构冲突，可以统一为实体或属性。通常的方法是，若为实体的一方仅含一个同对方冲突的属性，而不含任何其他属性，则可将该实体转化为与对方相同的属性；若其中还含有其他非冲突属性，则可将对方中与实体一方发生冲突的那个属性转化为实体。

例如，设一局部应用中有“部门”实体，另一局部应用中有一“职员”实体，“职员”实体中有一“部门”属性。此时，若“部门”实体中仅含一个“部门”属性，则将“部门”实体转化为一个“部门”属性而统一为“职员”实体中的“部门”属性。若“部门”实体中除了一个“部门”属性外，还有其他属性，则可将“职员”实体中的“部门”属性抽出来统一为“部门”实体。当然，是否如此处理，还需要考虑到“部门”、“职员”实体与其他实体可能存在的联系的影响。

对于第二类结构冲突。应基于一对多包含一对一、多对多包含一对多的概念处理。

假设图 4.4.4 所描述的图书管理中还应包括“图书采购”的信息，那么在设计好该应用的局部 E－R 图后，再与图 4.4.4 中的局部 E－R 图合并汇总，将得到图 4.4.5 所示的全局 E－R 图（为简便起见，图中省略了各实体的属性）。

图 4.4.5　全局 E－R 图

4.4.5 数据库逻辑设计

其任务就是把前面设计好的基本 E－R 图转换为与选用的具体机器上的 DBMS 产品所支持的数据模型相符合的逻辑结构。由于各种 DBMS 产品一般均具有许多不同的限制，提供不同的环境与工具，因此分为下面几个步骤。

① 将 E－R 图向一般关系、层次、网状模型转换。

② 进一步将得到的结构向特定的 DBMS 支持下的数据模型转换。

③ 依据应用的需求和具体的 DBMS 的特征进行调整与完善。

1. E－R 图向关系模型的转换

在此，以 E－R 图向关系模型的转换为例进行讨论。E－R 图向关系模型的转换要解决的问题是如何将实体和实体间的联系转换为关系模式，如何确定这些关系模式的属性和码。

对于实体，问题很好解决：将每个实体转换成一个关系，实体的属性即为关系的属性，实体的码即为关系的码。

对于实体间的联系，可根据不同的情况讨论。

（1）若实体间的联系是 1∶1

可以在两个实体转换成的两个关系中选择任意一个关系的属性，加入另一个关系的码。

例如：若某工厂中每个车间生产一种产品，那么车间实体与产品实体间的联系是 1∶1 的联系。在将其转换为关系模型时，车间与产品各为一个关系。

车间（车间号，车间名，车间主任代码，电话号码）各属性中，车间号为关系的码。

产品（产品号，产品名，规格）中产品号为此关系的码。

由于两者是 1∶1 联系，可以在任一关系中加入另一关系的码。应该在哪个关系中加入另一关系的码呢？如果经常要在查询车间关系时查询此车间生产的产品代码，就可以将产品号放入车间关系，以减少查询时的连接操作。反之，若查询产品时经常要查询生产该产品的车间号，则可将车间号放入产品关系。当然，如果没有上述两种情况，则在任一关系中加入另一关系的码均可。现在在车间关系中加入产品号，则车间关系为：

车间（车间号，车间名，车间主任代码，电话号码，产品号）

（2）若实体间联系是 1∶n

可以在 n 端实体转换成的关系中加入一端实体转换成的关系的码。

例如：一个部门有多名职工，一个职工只属于一个部门，那么，部门实体与职工实体之间联系为 1∶n，将两实体转化得到的关系为：

部门（部门号，部门名，电话号码）

职工（职工号，职工名，性别，年龄，部门号）

（3）若实体间的联系是 n∶m

可以将联系也转换成一个关系，关系的属性为各个实体的码加上联系具有的属性，而关

系的码则为诸实体的码的组合。

例如一个产品由多个零件组成,每个零件可用于多个产品,则产品实体与零件实体间为 $n:m$ 的联系,转换可得到如下 3 个关系。

产品(产品号,产品名,规格)

零件(零件号,零件名,库存量)

构成(产品号,零件号,数量)

2. 向特定的 DBMS 支持的数据模型转换

形成了一般的数据模型之后,下一步就是将得到的结构向特定的 DBMS 规定的模型转化。设计人员需要熟悉所用的 DBMS 的功能、特性与限制。

3. 数据模型的调整和完善

数据库逻辑设计的结果不是唯一的。在逻辑设计基本完成之后,再根据应用需要对设计结构进行适当的修改和调整,提高应用系统的性能。这种修改和调整包括两个方面:

(1) 数据库结构的调整

根据数据字典中记载的对信息查询的响应时间要求和查询频度要求进行数据库结构上的调整。

① 增加必要的冗余数据,这通常是提高查询速度的有效措施。在逻辑设计的最后阶段可作一次检查,看是否有必要增加冗余。

② 如果经常要做的查询是两个关系的连接,则可以考虑是否能够将两个关系合并为一个关系。当然,这种修改需要仔细考虑对相关实体与实体间联系的影响,然后再修改。

(2) 设计用户子模式

根据局部应用需求,结合 DBMS 特点设计用户子模式。在前面介绍 SQL 语句时曾提到视图的概念。利用视图可以达到以下目标。

① 使用更符合用户习惯的别名

对于数据库系统来说,同一关系和属性必须使用惟一的名字,但对于用户来说,这样的名字往往不习惯。此时可以利用定义视图来保持与习惯的一致。

② 定义不同的视图

可以对不同级别的用户定义不同的视图,以满足系统对安全性的要求。如关系模式:产品(产品号,产品名,规格,单价,生产车间,生产负责人,产品成本,产品合格率,质量登记)。可以在此关系上建立两个视图。

为一般顾客建立视图:产品 1(产品号,产品名,规格,单价)

为销售部门建立视图:产品 2(产品号,产品名,规格,单价,车间,生产负责人)

两个视图的用户都只能查看各自的属性,这样可以防止用户非法访问本来不允许他们查询的数据,保证了系统的安全。

(3) 简化用户对系统的使用

如果某些局部应用中经常要使用某些很复杂的查询,为了方便用户,可以将复杂查询定义成视图,用户每次只对定义好的视图进行查询,这样就可以大大简化了用户的使用。

4.4.6 数据库物理设计

数据库在物理设备上的存储结构与存取方法称为数据库的物理结构,它依赖于实际的计算机系统。对于前面做好的逻辑设计选择一个最符合应用要求的物理结构就是数据库的物理设计。

通常物理设计的内容主要包括以下几个方面。

(1) 确定数据的存储安排

对于这个问题,主要是从提高系统性能方面考虑,例如:

- 将表和索引分别放在不同的磁盘上,在查询时,由于两个磁盘驱动器分别在工作,因而可以保证物理读写速度比较快。
- 将比较大的表分别放在两个磁盘上,可以加快存取速度,特别是在多用户环境下。
- 将日志文件和数据库对象(如表、索引等)分别放在不同的磁盘上可以改进系统的性能。

由于各个系统所能提供的对数据进行物理安排的手段、方法不尽相同,因此开发人员应仔细了解给定 DBMS 在这方面的特点,针对应用环境的要求,对数据进行适当的物理安排。

(2) 存取路径的选择与调整

一般来讲,数据库系统是多用户共享的,对同一数据存储要建立多条存取路径才能满足多用户的多种应用要求。物理设计的任务之一就是要确定建立哪些存取路径。例如:将哪些属性列作为次码建立次索引,根据应用要求还要对哪些属性列建立组合索引;是否还需要根据数据库系统提供的功能建立其他类型的索引等。

(3) 确定系统配置

许多 DBMS 产品都设置了系统配置变量,例如:控制内存分配的参数,使用系统的用户数和其他影响系统性能的参数,供设计者和 DBA 对数据库进行物理优化。初始情况下,系统都已经设置了合理的缺省值。大多数情况下可以不用调整。但是这些值不一定适合每一种应用环境。因此,在物理设计时,可以对其重新赋值,以改善系统性能。

通常情况下,这些配置参数包括:同时使用数据库的用户数,同时打开的数据库对象数,使用的缓冲区长度、个数,时间片大小,数据库大小,装填因子,锁的数目等。这些参数值将影响存取时间和存储空间的分配,在物理设计时就要根据应用环境确定这些参数值,以使系统性能最优。

当然,有可能在系统运行时还要根据实际运行情况对此参数进行调整。所以,本步骤的调整只是初步的。

4.4.7 应用程序设计与系统的运行和维护

(1) 应用程序设计

经过前面的逻辑设计、物理设计,接下来就是应用程序设计并投入运行和维护。由于它与一般的程序设计并无本质的不同,因此也可采用一般程序设计的方法,例如:

- 自顶向下,或者结合由下而上灵活运用。
- 程序按功能模块化。
- 使模块便于组装和调用。
- 追求程序可读性,不过多地采用难以理解的技巧。

应用程序设计包括编码和调试工作。编码就是具体程序设计,所用的语言取决于DBMS语言系统。编写应用程序要求具有一定的数据库知识,有一定程序设计能力,需要一个实践锻炼的过程。调试工作相当耗费精力,一般来说静态工作越细致,出错可能越小。有时在调试中会出现意想不到而又一时难以发现的问题,这就需要步步跟踪,逐条分析,可在程序中设置几个断点,考察断点处的状态。通过程序调试,能发现所隐藏的问题,特别是前面逻辑设计的问题。程序初步调试成功,并不说明大功告成,必须经过试运行,对数据库施加增、删、改、查等各种操作,看是否还隐含有问题,同时考察其性能。

(2) 投入运行和维护

① 加载数据入库

系统运行和维护之前先要进行数据加载,即用具体DBMS提供的数据定义语言和方法将逻辑设计和物理设计的结果严格地描述出来,建立实际的数据库。加载一般是通过系统提供的实用程序和自编的专门录入程序进行的。

输入数据时,一方面系统对数据有完整性控制,另一方面在应用程序中也对数据有一个合理性的要求。

② 运行与维护

数据库投入运行标志着开发任务的基本完成和维护工作的开始。所谓数据维护,就是整理数据的存储。在数据库运行阶段,对数据库经常性的维护工作主要是由DBA完成的,主要包括以下几个方面。

(a) 数据库的转储和恢复

定期对数据库和日志文件进行备份,以保证一旦发生故障,能利用数据库备份及日志文件备份,尽快将数据库恢复到某种一致性状态,并尽可能减少对数据库的破坏。

(b) 数据库的安全性、完整性控制

DBA必须对数据库安全性和完整性控制负起责任。根据用户的实际需要授予不同的操作权限。另外,由于应用环境的变化,数据库的完整性约束条件也会变化,也需要DBA不断修正,以满足用户要求。

(c) 数据库性能的监督、分析和改进

目前许多 DBMS 产品都提供了监测系统性能参数的工具,DBA 可以利用这些工具方便地得到系统运行过程中一系列性能参数的值。DBA 应该仔细分析这些数据,通过调整某些参数来进一步改进数据库性能。

(d) 数据库的重组织和重构造

数据库运行一段时间后,由于记录的不断增、删、改,会使数据库的物理存储变坏,从而降低数据库存储空间的利用率和数据的存取效率,使数据库的性能下降。这时 DBA 就要对数据库进行重组织,或部分重组织(只对频繁增、删的表进行重组织)。

数据库的重组织不改变原设计的数据逻辑结构和物理结构,只是重新安排存储位置,回收垃圾,减少指针链,提高系统性能。DBMS 一般都提供了供重组织数据库使用的实用程序。

当数据库应用环境发生变化,会导致实体及实体间的联系也发生相应的变化,使原有的数据库设计不能很好地满足新的需求,从而不得不适当调整数据库的模式和内模式,这就是数据库的重构造。DBMS 都提供了修改数据库结构的功能。

重构造数据库的程度是有限的。若应用变化太大,已无法通过重构数据库来满足新的需求,或重构数据库的代价太大,则表明现有数据库应用系统的生命周期已经结束,应该重新设计新的数据库系统,开始新数据库应用系统的生命周期了。

4.4.8 编写技术文档

按照软件工程的要求,数据库应用系统设计应编写以下 3 类技术文档。

(1) 系统说明书

数据流图和数据字典是构成系统说明书的两份主要技术资料,用于表示系统数据和对数据所进行的加工,它们是设计人员设计和编程的基础。

(2) 技术说明书

技术说明书主要应包括在设计过程中所采用的技术手段和实现措施,对每一个环节的技术资料进行归纳、整理、存档。它包括以下几个主要方面:

① 数据流图和数据字典分析。

② 概念模式设计阶段产生的各局部 E - R 图,综合后的总体 E - R 图。

③ 逻辑设计阶段产生的各种记录类型及其联系;指出记录关键字,给出函数依赖关系,说明规范化的过程,说明采取规范式等级的依据。

④ 数据存取是否采用索引结构,根据什么建立索引,说明原因。

应用程序设计主要应给出程序总体结构、主要模块的程序框图,并附上全部程序清单。

(3) 使用说明书

使用说明书中应说明使用方法、操作步骤和维护工作中应注意的事项。这些资料,在系

统正式运行后应该交给用户。

4.4.9 数据库应用系统设计实例

本节将按照数据库应用系统的开发步骤(需求分析,系统设计,系统实施,系统运行与维护),采用结构化的设计方法,来完成一个学生信息管理系统的开发工作。

1. 需求分析

(1) 系统请求

随着学校规模的不断扩大,学生数量急剧增加,有关学生的各种信息量也成倍增长。面对庞大的信息量,过去纯粹的手工管理方式已很难适应现代学生培养管理的需要。因此,实现学生管理工作的科学化、规范化和信息化已成为教育管理改革中的一个重要内容。

(2) 可行性研究

对于学生信息管理系统,因为系统规模小,人工系统的构成也较简单,因此可行性研究也很简单,但必须详细了解原人工系统的硬件资源,尤其是外存容量,它是限制新系统目标的重要条件之一。例如,新系统中存储学生的情况,但是否存储学生的社会关系、历史材料、简历、奖惩情况等,则要兼顾用户的需要和可能两方面,以确定新系统的目标。

结合本实例的具体情况,分析确定新系统的目标是储存一个系(如计算机系)学生的必要信息,并能随时查询和编辑这些信息,打印出规定的报表等,即完成数据录入、维护、查询、打印报表等功能。

(3) 系统调查

在系统调查时主要了解以下几个方面:

① 学生入学登记、班级管理及课程管理等业务处理涉及到的所有信息。

② 在业务流程中对所有原始数据、报表、数据文件等所进行的加工处理、形成规律等,即全面了解信息流程。

③ 外部要求。包括数据保密性、使用范围、查询速度、人机交互、输入/输出格式等方面的要求。

(4) 整理文档

根据上述工作整理需求分析阶段的文档。

2. 系统设计

(1) 应用功能设计

本系统开发的总体任务是实现学生信息管理的系统化、规范化和自动化。为实现该目标,通过调查、收集资料,进行系统的功能分析,可以确定学生信息管理系统应具备以下基本功能:

① 学籍信息的输入、删除、修改、查询,包括学生基本信息、所在班级、所学专业、所属院系等。

② 班级信息的输入、删除、修改、查询，包括班级设置、所属院系等。

③ 学校基本课程信息的输入、删除、修改、查询。

④ 学生成绩、学分、奖惩等信息的输入、删除、修改、查询、统计。

对上述各功能进行集中、分块，按结构化程序设计的要求，得到学生信息管理系统的功能模块图如图 4.4.6 所示。

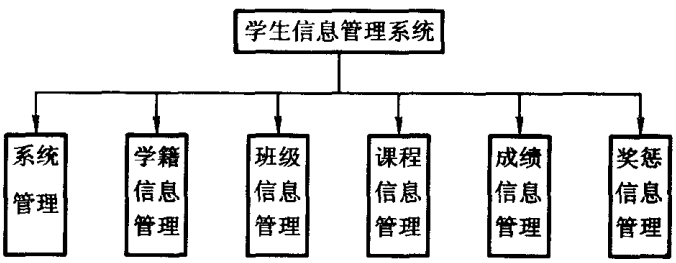


图 4.4.6 功能模块结构图

(2) 数据流图

遵循画数据流图的原则，可以得到图 4.4.7 所示的学生信息管理系统的数据流图。其中成绩、奖惩子系统对学籍管理子系统有一定的依赖关系，即必须先建立学生学籍，才能进行成绩、奖惩等信息的管理，可以通过定义相应主码与外码的约束来实现这种依赖关系。

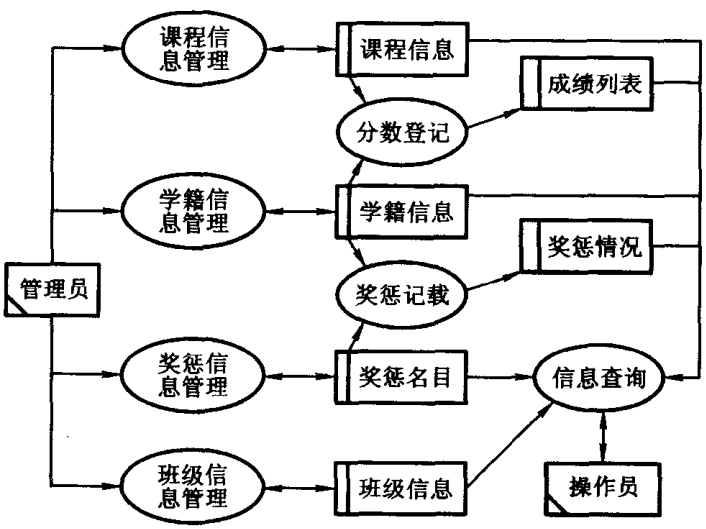


图 4.4.7 数据流图

(3) 数据字典

在数据流程图的基础上，可以从原始的数据资料中分析整理出数据字典的内容：数据

项、数据结构、数据流、数据存储以及处理过程。限于篇幅，在此略去数据字典的分析结果。

（4）建立 E－R 模型

根据前面分析得到的数据流程图和数据字典，可以设计出如图 4.4.8 所示的局部 E－R 图。

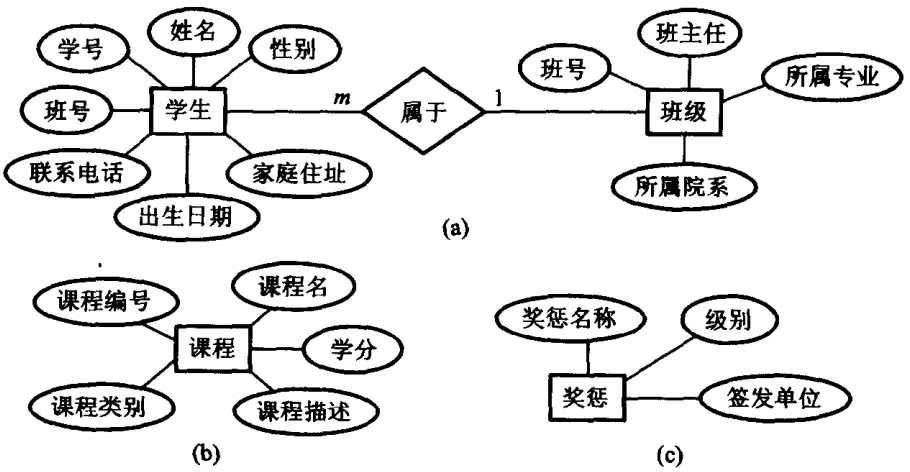


图 4.4.8　局部 E－R 图

把局部 E－R 图加以综合，并且合理消除属性冲突、命名冲突和结构冲突，便可以得到如图 4.4.9 所示的全局 E－R 图（为方面起见，图中略去了属性）。

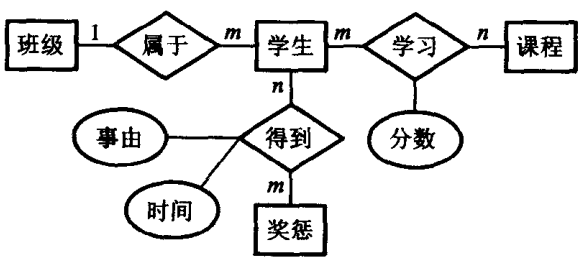


图 4.4.9　全局 E－R 图

（5）数据库模式设计

① 将 E－R 图转换为一般关系模型

将 E－R 图向一般关系模型转换，结果如下：

班级（班号，班主任，所属专业，所属院系）

学生（学号，姓名，性别，出生日期，联系电话，家庭住址，班号）

课程（课程编号，课程名，学分，课程类别，课程描述）

成绩(学号,课程编号,分数)

奖惩(奖惩名称,级别,签发单位)

奖惩情况(学号,奖惩名称,级别,事由,时间)

② 数据模型的调整和完善

上面的设计结果并不是唯一的,根据应用的需要,还可以做一些适当的修改和调整,以提高应用系统的性能。一种方法是适当增加冗余。如实际应用中经常需要查询学生的成绩,为了使查询结果清楚地反映学生姓名,可将成绩实体改为:

成绩(学号,课程编号,姓名,分数)

还可以根据需要创建视图。如,很多时候需要查询一个班的各科成绩,那么就可以创建一个包含班号的视图(view):

班级成绩(学号,课程编号,姓名,班号,分数)

3. 系统实施

(1) 数据库定义

数据库定义必须要采用特定数据库管理系统提供的数据定义语言来进行详尽描述,本实例中尽量使用标准 SQL 语言,并在 SQL Server 2000 下运行通过。

创建数据库(采用默认设置)

```
CREATE DATABASE SIMS
```

创建表格(USE SIMS)

```
CREATE TABLE 班级(
班号 INT NOT NULL,
班主任 CHAR(32),
所属专业 CHAR(64) NOT NULL,
所属院系 CHAR(64) NOT NULL,
CONSTRAINT pk_班号 PRIMARY KEY(班号)
)
CREATE TABLE 学生 (
学号 NUMERIC(15) NOT NULL,
姓名 CHAR(16) NOT NULL,
性别 CHAR(2) NOT NULL,
出生日期 DATETIME NOT NULL,
联系电话 CHAR(15),
家庭住址 CHAR(128),
班号 INT NOT NULL,
CONSTRAINT pk_学号 PRIMARY KEY(学号),
```

```
    CONSTRAINT chk_性别 CHECK(性别 IN ('男','女')),
    FOREIGN KEY (班号) REFERENCES 班级(班号)
)
CREATE TABLE 课程 (
    课程编号 INT NOT NULL,
    课程名 CHAR(32) NOT NULL,
    学分 INT NOT NULL,
    课程类别 CHAR(16) NOT NULL,
    课程描述 CHAR(256),
    CONSTRAINT pk_课程编号 PRIMARY KEY(课程编号)
)
CREATE TABLE 成绩 (
    学号 NUMERIC(15) NOT NULL,
    课程编号 INT NOT NULL,
    分数 INT NOT NULL DEFAULT 0,
    CONSTRAINT pk_学课 PRIMARY KEY(学号,课程编号),
    CONSTRAINT chk_分数 CHECK(分数 >= 0 AND 分数 <= 100),
    FOREIGN KEY (学号) REFERENCES 学生(学号),
    FOREIGN KEY (课程编号) REFERENCES 课程(课程编号)
)
CREATE TABLE 奖惩 (
    奖惩名称 CHAR(128) NOT NULL,
    级别 CHAR(16) NOT NULL,
    签发单位 CHAR(128) NOT NULL,
    CONSTRAINT pk_奖级 PRIMARY KEY(奖惩名称,级别)
)
CREATE TABLE 奖惩情况 (
    学号 NUMERIC(15) NOT NULL,
    奖惩名称 CHAR(128) NOT NULL,
    级别 CHAR(16) NOT NULL,
    事由 CHAR(1024) NOT NULL,
    时间 DATETIME NOT NULL,
    CONSTRAINT pk_学奖级 PRIMARY KEY(学号,奖惩名称,级别),
    FOREIGN KEY (学号) REFERENCES 学生(学号),
```

FOREIGN KEY（奖惩名称,级别）REFERENCES 奖惩(奖惩名称,级别)

)

此外还可创建索引来提高系统的查询效率,创建视图来满足各种应用的需要,创建存储过程以及触发器来确保数据的完整性和一致性,本节并不一一列出,有兴趣的读者可以自己研究。

(2) 数据库装载

就本例来说,学生学籍信息是在新生入学之初录入系统,而课程信息、班级信息以及奖惩信息均是人工收集完成或由相应的输入程序实现。如某校 26 班学生入校后的部分学生信息数据库表如图 4.4.10 所示。

	学号	姓名	性别	出生日期	联系电话	家庭住址	班号
1	2002612100008	杨东亮	男	1979-10-15 00:00:00.000	027-87554632	广东东莞	26
2	2002612100029	孙艳	女	1979-03-25 00:00:00.000	027-87553260	江苏扬州	26
3	2002612100036	胡超	男	1977-08-01 00:00:00.000	027-87554689	湖南衡阳	26
4	2002612100072	李劲伟	男	1978-12-31 00:00:00.000	027-87554697	四川乐山	26
5	2002612100075	张泉	男	1980-04-26 00:00:00.000	027-87554682	湖北武汉	26
6	2002612100093	方晨静	女	1979-06-18 00:00:00.000	027-87553264	河南信阳	26
7	2002612100161	万金华	男	1978-10-28 00:00:00.000	027-87554697	湖北黄冈	26
8	2002642100119	唐琼	女	1980-05-17 00:00:00.000	027-87553260	湖北武汉	26
9	2002642100168	冯涛	男	1978-02-22 00:00:00.000	027-87554697	湖北十堰	26
10	2002642100199	王荣	女	1980-12-25 00:00:00.000	027-87553296	云南西双版纳	26
11	2002642100249	张金红	女	1980-11-06 00:00:00.000	027-87553296	浙江温州	26

图 4.4.10 部分学生数据信息

(3) 应用程序设计

应用程序设计包括编制程序和调试程序。

应用功能总体结构图是系统总体设计阶段中应用功能设计所得结果,它是本阶段应用程序设计的主要依据之一。应用程序设计的第二依据是数据库定义。开发人员应综合分析这两部分内容,用数据库管理系统提供的编程语言,设计和编制应用程序。

应用程序设计可以采用两种方法:结构化程序设计方法和面向对象程序设计方法。

结构化程序设计方法是 80 年代通用的程序设计方法,该方法的实质是自顶向下逐步求精。它用在系统总体设计阶段,可以指导划分模块;用在应用程序设计阶段,将考虑一个程序如何用下面 3 种控制流程来编写。

- 顺序:程序中的代码按出现的先后顺序逐条执行。
- 分支:根据条件的成立与否,选择程序操作。
- 循环:当条件成立(或不成立)时,反复执行相同的程序段。

应用程序设计的第二项工作是调试程序，调试程序应遵循先局部再全局的原则，每做好一个程序，都要进行调试，再在局部范围联调，最后是系统联调。调试程序应尽量使用系统提供的调试工具。

4. 系统运行与维护

系统运行之前应经过试运行，确认系统无故障或暂未发现故障时，系统才能正式投入运行。

系统在运行中有可能暴露出设计中存在的问题，或用户又提出新的需求，更多的是随着数据的不断增、删、改而使系统的物理存储结构变坏或存取效率下降，因此都需要在系统运行阶段做好维护工作，如定期或不定期地进行数据转储、系统故障发生时及时恢复处理、实施安全与完整性控制、重组和重构数据库等。

对于用户的新需求、也应提出改进和扩展方案并付诸实施，以使应用系统尽可能满足用户的要求。

4.5 实用数据库技术简介

4.5.1 数据库技术的发展

1. 数据库技术的发展

数据库技术20世纪60年代中期产生，70年代蓬勃发展，80年代后则更加向纵深发展，并不断取得辉煌成就。

社会信息化和信息社会化给数据库技术的应用不断地提出了新的要求，使得传统的数据库技术无法满足，促使数据库技术与其他计算机技术交叉结合研究，如分布式处理技术、人工智能、并行处理技术、互联网技术、移动通信技术、Web数据库技术等，产生了一系列新的数据库系统。例如，将各个分散的管理信息系统联系起来，协调配合提供全局数据的分布式数据库系统；根据已知数据和逻辑关系进行推理，从而获得数据库中并不存在但客观上正确有用数据的演绎数据库；将图形、图像、文字、声音等非常规的复杂对象与数字、字符一类常规信息一并处理的多介质数据库系统等。20世纪80年代，人们对信息的需求上了一个新的台阶，这就迫使数据库专家们把眼界从数据库领域扩大到更加广泛的范围，形成了以数据库技术为核心，综合运用人工智能、分布式技术、图像识别、图形处理等多种学科领域的知识和技术，而逐步发展成高等的数据库系统。这些高等数据库的代表有：分布式数据库、演绎数据库、知识数据库、智能数据库、专家数据库、多介质数据库、工程数据库、面向对象数据库、主动数据库、实时数据库、空间数据库、并行数据库、嵌入式数据库等。

20世纪90年代,数据库系统也广泛地采用了流行的用于解决分布问题的新型计算机体系结构——客户机/服务器(Client/Server)结构,将原集中在主机完成的数据库管理系统功能和应用处理功能分离开来,由服务器和客户机分别负担,实现了客户机请求和服务器响应这样一种计算模式。服务器上专门运行数据库管理系统,客户机上则运行数据库管理系统的应用开发工具和用户的应用程序,它们通过网络连在一起。客户机要存取数据库中的数据时,向服务器提出请求,服务器响应这些请求,进行分析,作相应服务后将结果或状态信息返回给客户机。每个客户自己干自己的事,服务器向所有的客户提供数据库方面的服务,且服务对客户来说是透明的。这样即形成了所谓的客户机/服务器结构的数据库管理系统。该结构的基础构件由客户机/服务器以及连接它们的中间件组成,涉及硬件和软件两个方面。

2. 数据库技术发展的相关因素

数据库技术之所以能得到迅速地发展,是与许多因素密切相关的。

(1) 日益增长的应用需求

数据库系统的用户剧增,对信息的要求越来越高,越来越迫切。是促进数据库技术发展的直接动力,数据库技术的发展也使得计算机应用渗透到国民经济各个领域。

(2) 相关技术高速发展

高性能计算机、大容量的新存储介质、网络技术、数字化技术、通信技术等先进技术给数据库发展提供了强大的物质基础和技术支持。

(3) 数据库设计方法研究取得进展

初期的数据库设计方法,主要凭直觉和经验,工程规范程度不高,缺乏理论指导,设计的好坏在很大程度上取决于主要设计者个人的知识和实践经验。方法复杂,难于掌握,系统性能难以得到保证。数据库设计方法研究受到高度关注并取得进展。研究的内容包括:

① 数据模型设计方法。

② 数据存储与访问方法。

③ 数据库的管理与保护。

(4) 数据库规范化理论的研究成为一个独立的理论分支

(5) 数据库标准化技术

如异构数据库开放式互连,数据库语言标准化(如SQL)等促进系统向更加统一、标准化、通用化的目标迈进。

4.5.2 当前流行的数据库管理系统

当前流行的数据库管理系统产品,主要有Oracle、DB2、Sybase、Microsoft SQL Server等。限于篇幅,这里只对它们进行简单介绍,以便对更进一步的学习和选用数据库产品有所帮助。

(1) Oracle公司的Oracle

Oracle数据库系统支持多种系统平台(如Linux、HPUX、SUN OS、OSF/1、VMS、Win-

dows、WindowsNT、OS/2),具有面向制造系统的管理信息系统和财务系统的应用系统;当发展到 Oracle 7.1 版本服务器时,已经可以支持 1000 ~ 10000 个用户了。

1997 年,Oracle 公司推出了 Oracle 8,它增加了对象扩展和许多新特性及管理工具。

1999 年又推出了 Oracle 8i,它是世界上第一个全面支持 Internet 的数据库,是唯一具有集成式 Web 信息管理工具的数据库,也是世界上第一个具有内置 Java 引擎的可扩展的企业级数据库平台。它具有在一个易于管理的服务器中同时支持数千个用户的能力,可以帮助企业充分利用 Java,以满足其迅速增长的 Internet 应用需求。它通过支持 Web 高级应用所需要的多媒体数据来支持 Web 繁忙站点不断增长的负载需求。Oracle8i 提供了在 Internet 上运行电子商务所必需的可靠性、可扩展性、安全性和易用性,从而广受用户青睐。

(2) IBM 的 DB2

DB2 家族产品适合于各种硬件平台、可移植性好,它们能满足不同用户的需求,可分为 5 类:DB2 数据库服务器、网关、数据复制、数据库系统管理和 DB2 客户机产品。

现在市面上较为普遍的是 DB2 7. x 版,它支持标准的 SQL 存储协议语言,而且提供了相关的工具和服务,用户可以轻松地实现从 Oracle、Microsoft、Sybase 的数据库向 DB2 通用数据库移植。DB2 通用数据库 7. x 版可以运行于 UNIX、Linux 及 Windows 等 20 余种通用平台之上,并成为业界第一个将 Internet 检索与关系数据库的可扩展性和可用性有效集成的数据管理系统。目前,全球 30 万个企业中的 4 000 万个用户已经将 DB2 通用数据库用于客户关系管理、决策支持和业务分析等领域。总的来说,DB2 和 Oracle 功能十分强大,比较适合于开发大型数据库管理系统,而且在市场上也更受欢迎。

(3) Sybase

美国 Sybase 公司的名称是由 System(系统)和 Database(数据库)相结合的含义,该公司的最大功绩是促进了客户/服务器的迅猛发展。自 1992 年进入我国以来,十分重视中国市场的发展,引进了最新的技术和研究手段,促进了中国软件产业的发展。下面主要介绍 Sybase服务器系列产品。

① Sybase SQL Server

该产品是一个面向联机事务处理的可编程服务器,具有良好的性能和高可靠性,其最新产品是 Sybase SQL Server 11. x 版。

② Sybase Replication Server(Sybase 复制服务器)

Sybase 复制服务器能尽可能地快速工作,以保证整个企业范围数据的精确和一致。

③ Sybase Secure SQL Server(安全 SQL 服务器)

安全 SQL 服务器通过在不限制实际的联机应用所需求的关系数据库管理系统功能和特性的情况下,而提供严格的安全控制措施。

④ Sybase Navigation Server (Sybase 导航服务器)

Sybase 导航服务器允许在分布环境中多个 SQL Server 共同工作,对所有查询和事务提

供并行处理。它支持大规模并行处理和对称多处理环境中的巨大数据库,并且独立于网络和硬件。

(4) INFORMIX

INFORMIX 是美国 INFORMIX 软件公司的产品,INFORMIX 的取名来自 INFORMATION + UNIX。不难理解,刚开始时 INFORMIX 运行在 UNIX 平台,不过现在它也支持 SUN OS、HPUX、ALFAOSF/1 和 Windows 等。该数据库系统采用双引擎机制,占用资源小,简单易用,适用于中小型数据库管理。它的主要产品有:

- INFORMIX - Universal Server
- INFORMIX - DataBlade Modules
- INFORMIX - Online Extended Parallel Server
- INFORMIX - Online
- INFORMIX - SE

(5) INGRES

INGRES 数据库系统的多项技术直接采用了伯克利大学最新研究成果。技术上一直处于领先水平。INGRES 数据库不仅能管理数据,而且还能管理知识和对象(对象是指数据与操作的结合体,计算机把它们作为整体处理)。INGRES 产品分为 3 类:

- 第一类为数据库基本系统,包括了数据管理、知识管理和对象管理。
- 第二类为开发工具。
- 第三类为开放互联产品。

但是 INGRES 数据库系统学术价值大于实用价值。即在学术方面掌握领先技术,在产品服务上相对较薄弱。

4.5.3 SQL Server 系统及其使用简介

Microsoft SQL Server 脱胎于 Sybase SQL Server。1988 年,Sybase 公司、Microsoft 公司和 Ashton - Tate 公司联合开发的 OS/2 系统上的 SQL Server 问世。后来,Ashton - Tate 公司退出了 SQL Server 的开发,而 Microsoft 公司和 Sybase 公司签署了一项共同开发协议。

到 1992 年,将 SQL Server 移植到 Windows NT 操作系统上之后,Microsoft 公司和 Sybase 公司取消合同,各自开发自己的 SQL Server。Microsoft 公司致力于 Windows NT 平台的 SQL Server 开发,而 Sybase 公司则致力于 UNIX 平台的 SQL Server 开发。

1996 年,Microsoft 公司推出了 SQL Server 6.5 版本,1998 年,又将 SQL Server 升级到 7.0 版。

2000 年 4 月,Microsoft 公司推出了 SQL Server 2000 Beta2 版,又于同年 8 月推出了 SQL Server 2000 正式版,其中包括企业版、标准版、开发版和个人版 4 个版本。

SQL Server 2000 是可收缩的、高性能的、基于 SQL 和客户/服务器体系结构的关系数据库管理系统服务器软件,也是微软推出的新的数据库管理系统产品。下面简单介绍 SQL Server 2000 企业版的使用。

1. SQL Server 2000 的安装步骤

SQL Server 2000 的安装步骤如下：

(1) 放入 SQL Server 2000 光盘，自动安装程序启动，屏幕上出现的画面为图 4.5.1 所示。

(2) 选择“安装 SQL Server 2000 简体中文企业版”之后，进入的画面为图 4.5.2 所示。

(3) 选择“安装 SQL Server 2000 组件”之后，进入另外一个画面，然后再选择“安装数据库服务器”，进入 SQL Server 2000 企业版安装向导。

(4) 在安装向导对话框中单击“下一步”，进入计算机名对话框。

图 4.5.1　自动安装界面

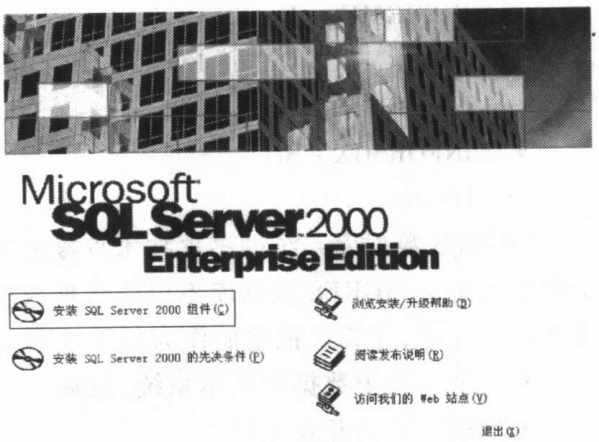


图 4.5.2　选择安装中文企业版

(5) 选择“本地计算机”，单击“下一步”，进入安装选择对话框。

(6) 选择创建新的 SQL Server 实例，单击“下一步”，进入用户信息对话框。

(7) 输入用户信息，单击“下一步”，进入安装定义对话框。

(8) 选择服务器和客户端工具，单击“下一步”，进入实例名对话框。

(9) 输入实例名，单击“下一步”，进入安装类型选择对话框。

(10) 选择典型安装，进入服务账号设置对话框。

(11) 选择对每一个用户使用同一个账号，自动启动服务器，单击“下一步”，进入选择身份验证模式对话框。

(12) 选择 Windows 身份验证模式。单击“下一步”，进入开始复制文件对话框，单击“下一步”，进入选择许可模式对话框，选择处理器许可证，单击“继续”，开始复制文件。

(13) 文件复制完毕后，进入安装完毕对话框，单击“完成”，系统安装完毕。

2. SQL Server 2000 的工具简介

(1) 启动服务管理器

在对数据库进行操作之前，先要启动服务管理器。步骤是：“开始”→“程序”→Microsoft SQL Server→“服务管理器”。然后进入如图 4.5.3 所示的画面，按照图 4.5.3 所示进行

操作。

（2）SQL Server 2000 查询分析器

查询分析器是一个十分重要的工具，所有SQL语言命令都需要在查询分析器中输入、编辑运行。进入步骤是："开始"→"程序"→Microsoft SQL Server→"查询分析器"。

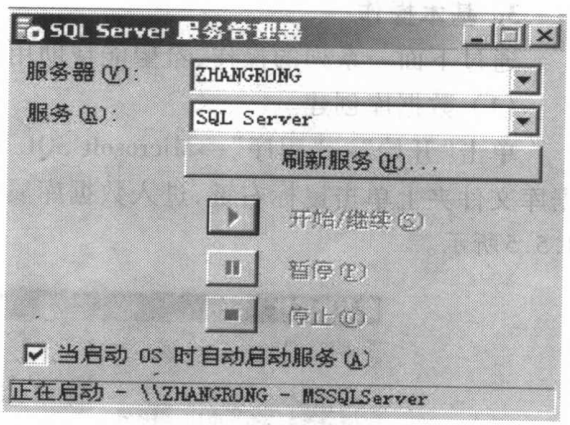


图 4.5.3 启动服务器

（3）SQL Server 2000 企业管理器

单击"开始"→"程序"→Microsoft SQL Server→"企业管理器"，进入企业管理器。重复单击控制台根目录下面的"+"，进入数据库文件夹，其中数据库 master、model、msdb、tempdb 是系统数据库，pubs、Northwind 是 SQL Server 自带的样本数据库，有大量的实验数据，可以作为 SQL 语言实验的基础数据，如图 4.5.4 所示。

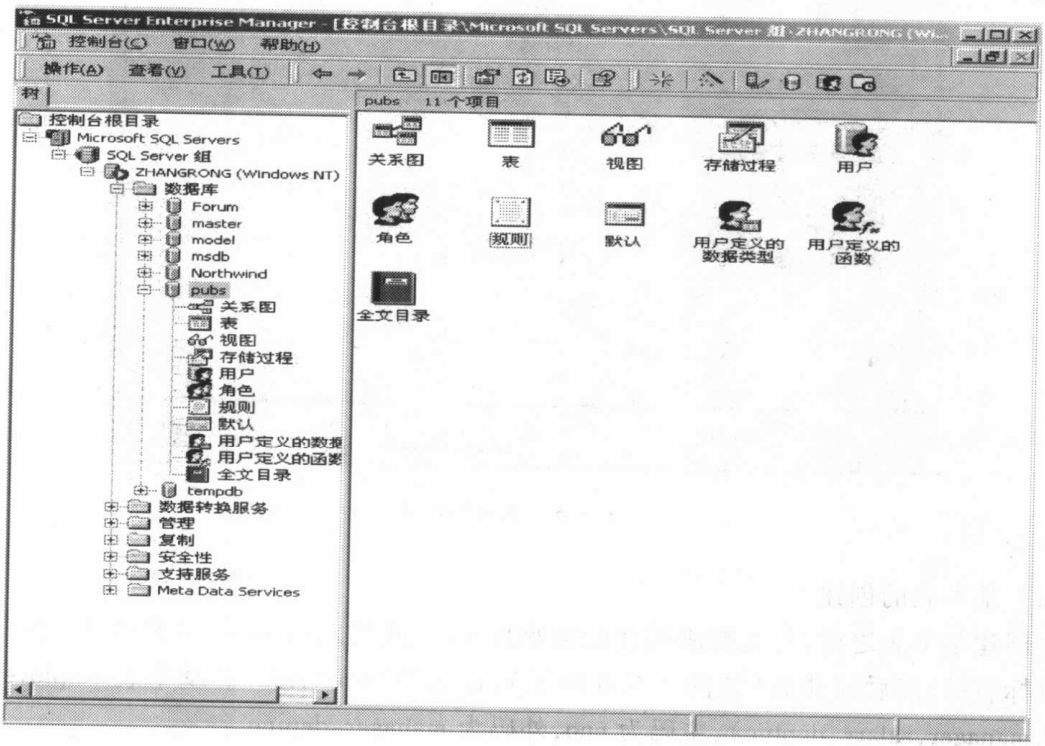


图 4.5.4 进入企业管理器

3. 基本操作

通过下面一系列的操作，希望能帮助用户基本掌握数据库、表和视图创建。

(1) 数据库创建

单击“开始”→“程序”→Microsoft SQL Server→“企业管理器”，进入企业管理器后，在数据库文件夹上单击鼠标右键，进入数据库属性对话框，新建数据库，命名为“ems”，画面如图4.5.5所示。

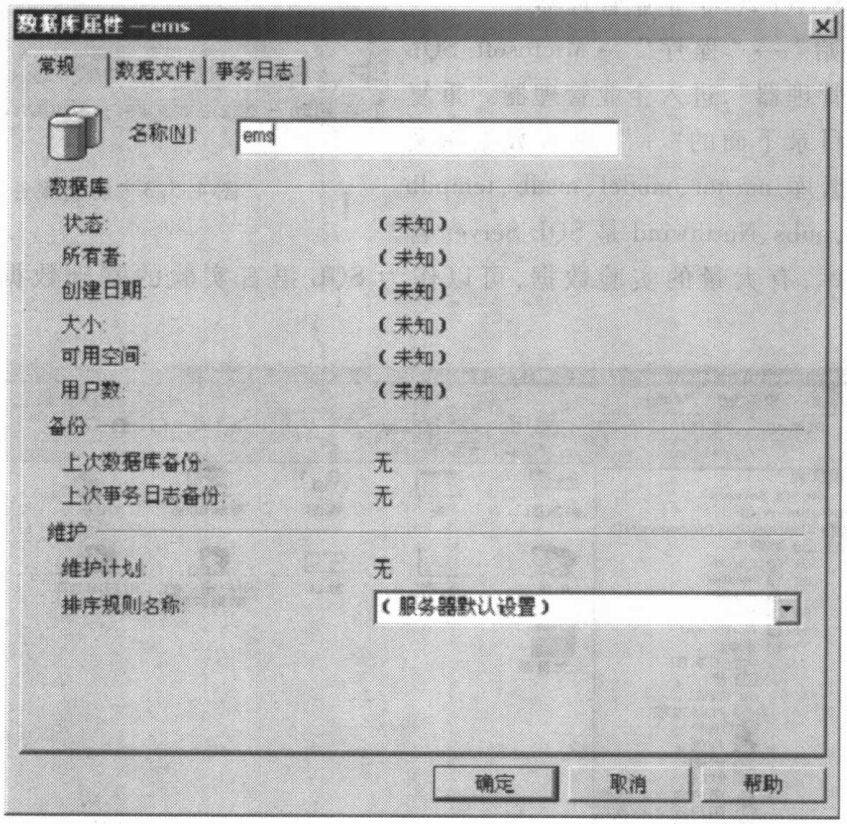


图 4.5.5 创建数据库

(2) 基本表的创建

在创建基本表之前，先选择刚创建的数据库 ems，然后双击 ems，在子目录中的“表”上单击鼠标右键，然后新建表(如图 4.5.6 所示)，命名为“employee”。基本表 employee(eno, ename, manager, salary, deptno)，主码为 eno，外码为 manager、deptno。

(3) 视图的创建

与上面创建基本表类似，在视图上单击鼠标右键，然后在新视图中输入如图 4.5.7 所示的信息，然后单击工具栏中的 ! 按钮，并把该视图存为 annualsal。

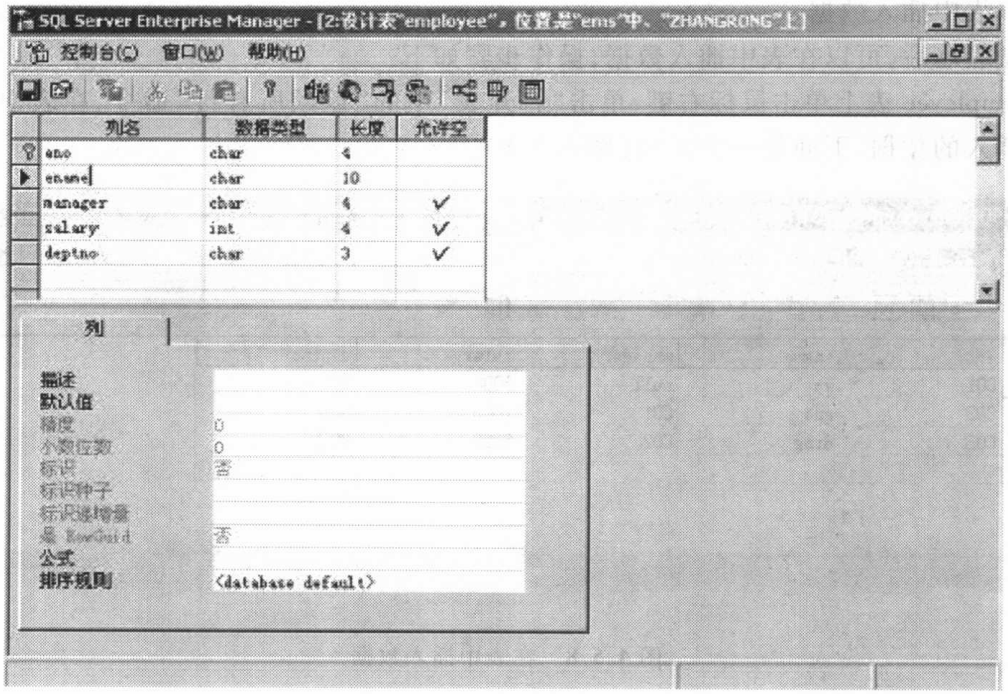


图 4.5.6 创建表 employee

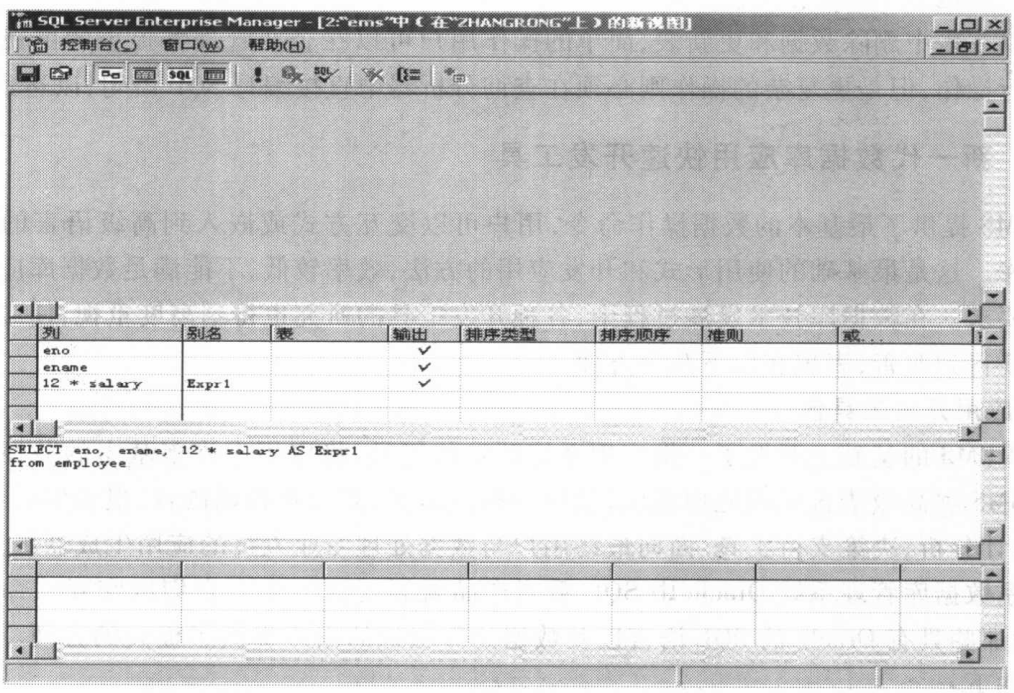


图 4.5.7 创建视图

(4) 表中插入数据

建立表之后,可以在表中插入数据,操作步骤如下:

在 employee 表上单击鼠标右键,单击“打开表”中的“返回所有行”命令,在表中插入用户需要插入的数据,下面是一个示例(图 4.5.8)。

图 4.5.8 在表中插入数据

(5) 其他

可以在表中删除数据和更新表,简单的操作用户可以在企业管理器提供的可视化界面直接进行操作,但是更复杂的操作则必须在查询分析器中直接编写 SQL 语句,以达到目的。

4.5.4 新一代数据库应用快速开发工具

DBMS 提供了最基本的数据操作命令,用户可以交互方式或嵌入到高级语言的方式存取数据库。这是最基础的使用形式和开发应用的方法,效率较低,不能满足数据库应用迅猛增长的要求。在数据库技术发展过程中,各种开发工具的研究也得到高度重视,使应用开发环境不断得到改善,表现在以下几个方面。

1. 应用开发工具包

在 DBMS 的基础上开发了一批应用系统开发的工具,将应用中许多基本的共性的工作利用 DBMS 和高级语言做成通用程序,给用户提供工具,如交互查询语言、报表生成器、图形语言、统计分析、决策支持工具、面向最终用户与数据处理专业人员的应用生成器等。

关系数据库管理系统 Oracle 中 SQL 系列产品就属于这一类。它们以交互式命令方式驱动,为那些具有 Oracle 的 SQL 语言以及数据处理知识的用户提供了极大的方便。其中部分工具如下:

(1) SQL * PLUS 。它是交互式命令接口,是一种增强型 SQL 语言,利用它可以进行数

据库定义、存储、检索、操纵与维护，还可生成格式化数据报表，提供求助、编辑命令以及调用操作系统命令等。

(2) SQL * FORM 。是一个应用生成器，对交互式应用它能自动生成并运行。

(3) SQL * Report。报表生成器，用户可用 SQL 语句查询数据库内容，并将查询结果和附加的正文标题等信息按所需的格式在终端或打印机上输出格式化的报表。

(4) SQL * Menu 。是一个动态菜单生成和管理工具，它提供了在应用系统中通过菜单方式执行不同的程序及操作系统命令的手段。

(5) SQL * Graph。是一个交互式图形工具，能将数据库中的信息以饼图、直方图和折线图的形式输出，能支持各种图形终端、绘图仪和打印机。

还有很多就不一一介绍，不同系统提供的工具有所差别，但大都简单易学，使用方便，能大大缩短开发周期，降低成本，为用户减轻负担。

2. ODBC 接口

ODBC 的中文名称为开放数据库互联，是通用的数据库接口。

在前面所讲的环境下，数据库应用程序要么基于具体 DBMS 的开发包，要么基于嵌入式 SQL 语言，用这种接口编制的应用程序，显然是与具体的数据库管理系统分不开的，不同 DBMS 的开发包肯定不会完全兼容，嵌入式 SQL 语言依赖于具体的预编译程序，要想实现应用程序与数据库无关，要想实现应用程序在不同的 DBMS 之间移植和转换，都是非常困难的。

为了达到不同数据库之间的互连互访，ODBC 定义了所有数据库系统公用的函数接口和 SQL 语法，基于 ODBC 的应用程序可以与具体的 DBMS 无关。目前大多数数据库开发商都支持 ODBC 接口，而且，许多新的应用程序开发工具，如 PowerBuilder，，Delphi 等，都在 ODBC 之上开发了若干通用的、简单的数据库构件，从而使用户根本不用考虑具体的 DBMS 就能开发数据库应用程序。应用开发环境得到了进一步的改善。

3. 新一代数据库应用快速开发工具

客户/服务器数据库自 20 世纪 90 年代流行起来后，数据库的应用开发环境获得了更大的改善。应用开发人员的主要工作是在客户机端用基于图形用户界面(GUI)的工具迅速地开发应用。除了像 Oracle Designer 2000 和 Developer 2000 一类工具外，还有一些当前广泛采用的主要用于开发的工具，如 PowerBuilder、Delphi、Visual Basic、Visual C++ 、Uniface、SQL Windows、Unify Version 等以及主要用于设计、分析的工具 ERwin、S - Design 等。这些工具的共同特点是：具有 GUI 可视化图形开发环境，适应客户/服务器结构，支持多平台、通过它们可以方便地存取后端服务器上多种关系数据库中的数据，广泛采用了面向对象技术，内部模块的可重用性好，开发效率高，维护成本低。

(1) Visual Basic

自 Windows 95 推出后，Microsoft 公司迅速推出了开发工具 Visual Basic 4.0，它可以建

立图形化的数据库存取应用,在该工具中有数据存取对象,微软 Jer 3.0 数据引擎(Data Engine),远程自动监控,对象数据库连接(ODBC),ODBC API 以及远程数据对象等。

(2) Borland 公司的 Delphi

Delphi 是一个 Windows 快速开发工具,由美国 Borland International Inc. 研制开发。它兼有 Visual C ++ 的强大功能和 Visual Basic 的易用性,具有可视化编程和面向对象相结合的优点,很多界面设计就像是在"画"而不是在写,大大提高了开发效率。

Delphi 在数据库支持方面具有明显的优势,它支持桌面数据库、SQL 数据库查询软件以及 Oracle、Sybase 和 Interbase 等数据库。由于 Borland 公司缺少自己有影响力的数据库产品,故它在支持别的数据库时就下了更多的功夫。它的 BDE 引擎功能强大,可以支持本地数据库,客户/服务器形式和 ODBC 方式的数据库平台,也可以使用第三方工具或者自己创建数据库访问类来进行数据库程序的编写。应用程序通过 Delphi 的 BDE(Borland Data Engine)可以连接多种数据库,如 Oracle、Sybase、INFORMIX、SQL Server、DB2 等。

现在比较流行的版本是 Delphi 6.0,最近又推出了较新的版本 Delphi 7.0,它们支持数据库开发的功能不断得到增强。

(3) PowerBuilder

PowerBuilder 是著名的数据库应用开发工具生产厂商 Sybase Inc. 的子公司 PowerSoft 于 1991 年 6 月推出的数据库应用开发工具,历经了多次升级换代,是目前具有代表性的数据库前端开发工具之一。它以其优良的性能和普及率领导着数据库应用技术的发展潮流。

PowerBuilder 采用的是图形化的界面和可视化的编程方法,通过引入独具特色的数据窗口对象,使得开发人员可以可视化地完成对数据库的操作。PowerBuilder 提供了对目前流行的大型数据库和桌面数据库的支持,如 Oracle、Microsoft SQL Server、DB2 和 Access 等,同时它自身也附带一个数据库管理系统 Adaptive Server Anywhere,几乎具备大型数据库的一切特征。

PowerBuilder 支持多种软硬件平台,是一个跨平台的图形开发环境,支持 Windows 3.x、Windows 95/98/NT/2000/XP、UNIX 和 Macintosh,不仅跨平台共享程序中的各种对象,还支持应用程序的跨平台开发和分布,极大地减轻了程序员在不同平台上移植程序的工作量。

现在市面上比较流行的版本是 PowerBuilder 7.0 和 PowerBuilder 8.0 版。7.0 版是一种面向 Web 与分布式环境的产品,从传统的客户/服务器两层蜕变为一层,以适应多层次客户/服务器体系以及电子商务的发展。相比 7.0 版而言,8.0 版有更强大的 Web 应用开发能力,开发系统更简洁、更高效。

除了上数据库快速开发工具之外,还有一些其他工具,在此不再详细介绍,用户有兴趣可以进一步阅读相关的资料。

本章小结

本章第1节全面介绍了数据库的基本概念，并通过对数据管理发展情况的介绍，说明了数据库系统的优点。阐述了数据库系统结构，即三级模式两级变换，由此使系统获得了数据独立性。在第2节介绍了数据模型的基本知识和3种主要的数据模型以及E－R模型。掌握好这些基本概念和基本知识，有利于进一步学习下面部分。

第3节介绍的关系数据库知识是本章的重点之一。在介绍了有关关系数据库的基本概念后，重点介绍了关系数据库语言——SQL，包括SQL的特点、它的数据定义功能和数据操纵功能，并结合实例讨论了如何运用SQL创建数据库、表，并对数据进行各种查询、插入、删除等操作。另外还介绍了视图的概念、对视图的操作以及视图的优点，并简要介绍了嵌入式SQL的基本概念和基本方法。用户最好在读书学习的同时进行上机实习，在上机的过程中进一步理解掌握有关SQL的知识。

第4节介绍的数据库应用系统的设计是本章的重点之二。在讨论数据库设计的方法和步骤时，列举了较多的例子，详细介绍了数据库设计各阶段的目标、方法和应注意的事项。其中重点是概念结构的设计（即E－R图的设计）和逻辑结构的设计，这也是数据库设计过程中最重要的两个环节。学习这一部分，要努力掌握书中讨论的基本方法，还要能在实际工作中运用这些思想，设计符合应用需求的数据库应用系统。接下来给出了一个较为完整的设计实例供用户参考。

在本章的最后，也就是第5节，还对当前实用的数据库技术做了介绍。先讲述了数据库技术的发展，然后介绍了当前流行的数据库管理系统，如Oracle、Sybase、DB2等，特别还对较实用的SQL Server的用法做了一些介绍。最后，讲述了新一代数据库应用快速开发工具，主要包括应用开发工具包、ODBC接口以及广泛使用的PowerBuilder，Delphi，Visual C++等工具。

习 题 四

1. 解释下列术语。

实体，属性，模式，子模式，内模式，数据库，数据库系统，数据库管理系统，数据模型，数据独立性，数据一致性，数据完整性，数据安全性，关系模式，关系数据库模式，元组，候选关键字，关键字，外键。

2. 已知有关系*R1*和*R2*如表1和表2所示。

表1　关　系　*R1*

A	*B*	*C*
3	2	4
3	5	6
7	2	6

表2　关　系　*R2*

A	*B*	*C*
3	5	6
7	2	4
3	8	4

求 $R1 \cup R2$，$R1 - R2$ 和 $R1 \cap R2$。

3. 已知关系 R，S，T 如表3～表5所示。

表3　关　系　*R*

A	*B*	*C*
a	*b*	*c*
d	*e*	*f*
g	*h*	*i*

表4　关　系　*S*

A	*B*	*C*
a	*b*	*c*
l	*m*	*n*

表5　关　系　*T*

A	*B*	*C*
d	*e*	*f*
g	*h*	*i*

已知 T 是 R 和 S 经过关系运算所得的结果，给出该关系运算的关系代数表达式。

4. 设有如表6～表8所示的3张表。

表6　学　　生

学号	姓名	性别	年龄
2003001	吴影	女	21
2003002	陈达	男	20
2003005	甘露	女	22
2003008	马跃	男	23

表7　课　　程

课程号	课程名	学时数
01	数学	180
02	英语	200
03	数据库	70
04	*C* 语言	60

表8　成　　绩

学号	课程号	分数
2003001	01	83
2003001	02	92
2003002	02	72
2003002	03	85
2003005	01	76
2003005	04	95
2003008	01	55
2003008	03	88
2003008	04	70

完成下列工作：

（1）写出成绩由高到低排列出学生姓名、性别、课程名、分数的 SQL 语句。

（2）给出下列 SQL 语句的汉语意思及其执行的结果。

```
SELECT 姓名,性别,课程名,分数
FROM 学生,课程,成绩
WHERE（学生.学号 = 成绩.学号）AND（课程.课程号 = 成绩.课程号）AND 成绩.分数 =
```

(SELECT MAX(成绩.分数) FROM 成绩)

5. 设有如下3个关系:(假定所有老师的姓名均不相同)

学生(学号,姓名,性别,年龄)

课程(课程号,课程名,任课教师)

选课(学号,课程号,成绩)

试用关系代数表达式完成下列查询语句:

(1) 查询所有年龄大于22岁的男生的信息。

(2) 查询学号为"200264210"的学生所学课程的名称、成绩以及任课教师。

(3) 查询至少选修"吴哲"老师所授课程中一门课的女生姓名。

(4) 查询"张敏"同学不学的课程的课程名。

(5) 查询所有至少选修两门课程的学生学号。

(6) 查询全部学生都选修的课程的课程号与课程名。

(7) 查询选修课程包含"吴哲"老师所授课程的学生学号。

6. 在5题所给的关系基础上,将所有查询用SQL语句实现,另外还需完成下列查询:

(1) 统计有学生选修的课程门数。

(2) 求选修C4课程的学生的平均年龄。

(3) 求"吴哲"老师所授每门课程的学生平均成绩。

(4) 统计每门课程的学生选修人数,要求输出超过10人选修的课程号和选修人数,查询结果按人数降序排列,若人数相同,按课程号升序排列。

(5) 查询姓"张"的所有学生的信息。

(6) 求年龄大于女同学平均年龄的男生的姓名和年龄。

7. 简要回答下列问题:

(1) 数据管理技术的发展经历了哪几个阶段?其特点分别是什么?

(2) 举例说明什么是数据冗余?它可能产生什么后果?

(3) 数据的物理独立性以及逻辑独立性的含义分别是什么?

(4) 说明数据字典的组成及其作用。

(5) 试述DBMS的主要功能。

(6) 模式与子模式在构成与作用上分别有什么不同?

(7) 什么是E-R图?构成E-R图的基本要素是什么?

(8) 关系数据库如何实现数据之间的联系?

(9) 什么是实体完整性?如何实现实体完整性?

(10) 什么是参照完整性?如何实现参照完整性?

8. 设要建立一个企业管理数据库(EMDB),它要求:每一企业有若干部门和生产工程;每一部门有许多职工,他们分别为一定的生产工程工作;每一工程生产一定的产品,并由与企业有合同关系的供应商提供各种零件;每一产品由一定的生产工程用相关的零件来生产。

完成如下设计:

(1) 构造符合上述条件的E-R模型(适当自定义有关的属性)。

(2) 将该 E-R 模型转换成关系模型,并用下划线标出关键字。

9. 现有一教学管理系统,它管理下列各种表:

学生:学号,姓名,性别

课程:课程号,课程名

专业学生:专业名,学号,姓名

排课表:课程号,学期,时间,教室,教师名

学生选课:课程号,学号,学期

教材使用:课程号,书名,学期

讲课教师:课程号,教师名

周课时:课程号,周学时

要建立一个数据库系统来实现这个教学管理系统,应完成如下设计:

(1) 按通常的语义给出一个完整的 E-R 图。

(2) 将该 E-R 图转换成关系模型,并用下划线标出关键字。

10. 设一个海军基地要建立一个舰队管理信息系统,它包括如下两个方面的信息:

舰队方面:

舰队:舰队名,舰队号,基地地点,舰艇数量。

舰艇:舰艇号,舰艇名,舰队名称。

舰艇方面:

舰艇:舰艇编号,舰艇名,武器名称。

武器:武器号,武器名称,类别。

官兵:官兵证号,姓名,职务。

其中,一个舰队有多艘舰艇,一艘舰艇属于一个舰队;一艘舰艇有多个官兵,一个官兵只属于一个舰艇;一艘舰艇可以安装多种武器,一种武器可安装在多艘舰艇上。

用户应完成如下设计:

(1) 分别设计舰队和舰艇两个局部 E-R 图。

(2) 将上述两个局部 E-R 图合并成为一个全局 E-R 图,合并时存在哪些冲突? 该如何处理这些冲突?

(3) 将该全局 E-R 图转换成关系模型,并用下划线标出关键字。

11. 某运动会历经数届,每届在不同地点举行,设有多项体育比赛项目,有若干运动队参加,每个运动队有许多运动员,一个运动员仅可为一个运动队的队员,可参加多届运动会的多个体育项目的比赛。现在要建立一个数据库完成下列任务:

(1) 登记和检索历届运动会参与的运动队和运动员的信息。

(2) 登记和检索运动员参加历届运动会的各种比赛项目及成绩(名次)。

用户应完成如下设计:

(1) 按通常语义拟定实体、属性后,分别构造上述应用的局部 E-R 图。

(2) 将上述两个局部 E-R 图合并成为一个全局 E-R 图,合并时存在哪些冲突? 该如何处理这些冲突?

(3) 将该全局 E-R 图转换成关系模型,并用下划线标出关键字。

第五章　计算机网络及应用

计算机网络是计算机技术和通信技术紧密结合的产物，它涉及计算机和通信两个领域。计算机网络在当今社会经济中起着极其重要的作用，对人类社会的进步做出了巨大贡献。从某种意义上讲，计算机网络的发展水平标志着一个国家的综合国力及现代化的程度。

在本章将介绍计算机网络的基本概念、计算机网络的协议和服务、网络互联与 Internet、计算机网络的应用和网络安全的基本知识。

5.1　计算机网络的概念

5.1.1　计算机网络的定义

计算机网络是计算机技术与通信技术结合的产物，是互联起来的计算机的集合。连接是物理的，由硬件实现。连接的介质（有时也叫信息传输介质、媒体）可以是普通的双绞线、同轴电缆和光纤电缆等“有形”物质，也可以是激光、微波或卫星信道等“无形”物质。

在计算机网络环境中的计算机既是相互独立自治的，又能相互传递信息。人们利用计算机网络，可以在一个部门、一个城市乃至全球范围内实现对计算机系统资源的共享。

在上述的定义中之所以强调入网计算机的“独立自治”，是为了将计算机网络与主机加多台设备构成的主从系统区别开来，如一台计算机带多台终端和打印机，这种系统通常称为多用户系统，而不是计算机网络，由一台主机控制多台从属机构组成的系统是多机系统，也不是计算机网络。

计算机与通信相结合表现在两个方面：通信网络为计算机之间的数据传递和交换提供了必要的手段；数字计算技术的发展渗透到通信技术中，又提高了通信网络的各种性能。当然，这两方面的发展和进步离不开微电子技术的高速发展和辉煌的成就。

5.1.2　信息时代中的计算机网络

人类进入的 21 世纪是一个以网络为核心的信息时代，其特征是数字化、网络化和信息化。计算机及计算机网络几乎渗透到人类生产和生活的所有领域，促进世界经济从工业经济转向知识经济。

知识经济中两个重要的特点就是信息化和全球化。要实现这个目标，就必须依靠完善的网络。因此，网络现在已经成为信息社会的命脉和发展知识经济的重要基础，对经济的发展已经产生了不可逆转的影响。

上述网络包括电信网络（主要业务是电话，但也有其他业务，如传真、多媒体、数据等）、有线电视网络（即单向电视节目的传送网）和计算机网络。虽然这 3 种网络在信息化过程中都起到了重要的作用，但其中发展最快并起核心作用的是计算机网络。

在 20 世纪 90 年代，以 Internet 为代表的计算机网络得到飞速的发展，已经从最初的教育科研网络逐步发展成为商立网络和服务网络，正在影响和改变人们的生活和工作的各个方面，它已经给很多国家（尤其是 Internet 的发源地——美国）带来巨大的经济利益和社会效益，加速了全球信息革命的进程，促进了全球经济的发展。

近几年出现的网格（grid）是将广域范围的各类计算资源（包括 CPU、存储器、数据库等）通过高速的 Internet 组成充分共享的资源集成，从而提供一种高性能计算、管理及服务的资源能力。人们用这些资源就像用电源一样，不必计较这些资源的来源和负载情况。

5.1.3　计算机网络的发展过程

世界上第一台数字电子计算机自 1946 年问世后，在最初的 10 年内，计算机和通信没有什么关系。当时计算机以“计算中心”的服务模式工作。直至 1954 年，一种收发器（transceiver）的终端制造出来后，人们才首次使用这种终端通过电话线路将数据发送到远地的计算机。此后，计算机开始与通信结合，计算中心的模式逐渐转变为计算机网络的服务模式。

计算机网络的发展大致可以分为以下 4 个阶段。

1. 面向终端的计算机通信网

用一台计算机专门进行数据处理，用一个通信处理机或前端处理机（FEP，Front End Processor）通过 Modem 与远程终端相连，如图 5.1.1 所示。通信处理机完成全部通信任务，ModemM 将终端或计算机的数字信号变换成可以在电话线路上传输的模拟信号，或者把电话线路上传输的模拟信号转换成计算机和终端使用的数字信号。由于前端处理机可采用廉

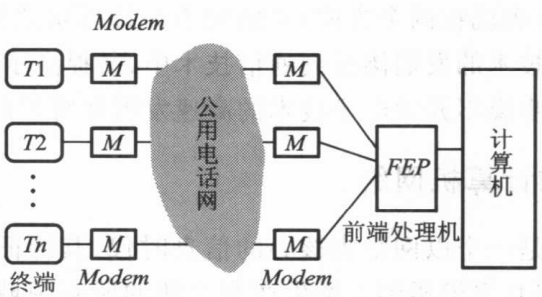


图 5.1.1　面向终端的计算机通信网

价的小型计算机，所以从20世纪60年代初起一直被广泛使用。这种联机系统称为面向终端的计算机通信网。也有人把这种最简单的计算机网络称为第一代计算机网络，这是一种以单主机为中心的星形网，如终端通过通信线路共享主机的软件和硬件资源。

2. 分组交换网

分组交换(packet switching)也称为包交换，它是现代计算机网络的技术基础。在有线电话出现不久，人们就认识到在所有用户之间架设直达线路，不仅线路投资太大，而且没有必要，可以采用交换机实现用户之间的联系。一百多年来电话交换机从人工转接发展到现在的程控交换机，经过多次更新换代，但交换方式始终没有改变，都是采用电路交换(circuit switching)，即通过交换机实现线路的转接，在两个用户之间建立起一条专用的通信线路。用户通话之前，先要申请拨号，待建立一条从发端到收端的物理通路后，双方才能互相通话。在通话的全部时间内，用户始终占用端到端的固定线路，直到通话结束，挂断电话(释放线路)为止。这种通信系统不适合传送计算机或终端的数据，原因有以下几个方面。

(1) 计算机的数字信号是不连续的，它与打电话传送的连续语音信号不同，具有突发性和间歇性，传送这种信号真正占用线路的时间很少，往往不到10%甚至1%，在绝大部分时间里通信线路是空闲的。但是，对电信局来说，只要用户占用了通信线路，就得收费，而不管传输了多少数据。

(2) 电路交换建立通路(即呼叫过程)的时间为10～20秒，对打电话来说时间并不长，但对只需传送半秒的计算机数据来讲就太长了。

(3) 电路交换很难适应不同类型、规格、速率的终端和计算机之间的通信，除非采取一些措施，如在终端与计算机之间经过缓冲器暂存一下，经过适当变换后再发送或接收。但这样就有别于电路交换。

(4) 计算机通信对可靠性要求很高，如需要在传输过程中进行差错控制，以电路交换方式难以做到。

因此，必须寻求适合计算机通信的交换技术，才能使计算机网络得以发展。1964年8月，巴兰(Baran)首先提出了分组交换的概念。1969年12月美国的分组交换网ARPANET投入运行，从此计算机网络进入一个崭新的发展阶段，标志着现代通信时代的开始。

图5.1.2为分组交换网的示意图，图中结点A、B、C、D、E和连接这些结点的链路AB、BD、DC、CE组成分组交换网，通常称为通信子网，结点上的计算机称为结点交换机。在ARPANET中结点交换机曾被称为接口报文处理机(Interface Message Processor，IMP)。图中的H1～H6都是一些独立的并且可以进行通信的计算机，称为主机(host)。

当主机H1向主机H6发送数据时，首先将数据划分成为一系列等长的单位(例如1 024 b)，称为分组。同时附上一些有关目的地址等信息，然后将这些分组依次发给与H1相连的结点A。这时除链路H1—A外，网内其他通信链路并不被目前通信的双方所占用，即使链路H1—A也只当分组正在该链路上传送时才被占用，在各分组传送的空闲时间，仍可用于

传送其他主机发来的分组。结点 A 收到分组后,先将收到的分组存入缓冲区,再根据分组所带的地址信息按一定的路由算法,确定将该分组发往哪个结点。由此可见,各结点交换机的主要作用是负责分组存储、转发及路由选择。

图 5.1.2 是分组交换网的示意图,一个分组交换网可以允许很多主机同时进行通信,而一个主机中的多个进程(即正在运行的多道程序)也可以各自与不同主机中的不同进程进行通信。

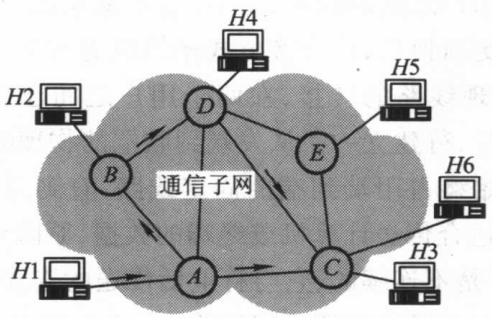


图 5.1.2 分组交换网的示意图

综上所述,存储转发分组交换技术采用的策略是断续(或动态)分配传输通路。因此,非常适合传输突发式的计算机数据,极大地提高了通信线路的利用率,降低了用户的使用费用。

ARPANET 采用分组交换技术取得成功,使计算机网络的概念发生了根本的变化,由以单个主机为中心的面向终端的计算机网转变成以通信子网为中心的分组交换网。而主机和终端则处于网络的外围,用户不仅共享通信子网的资源,还可以共享连网的计算机系统资源(包括硬件和软件)。这种以通信子网为中心的计算机网络通常称为第二代计算机网络,它的功能比第一代计算机网络扩大了很多,当今著名的全球性网络 Internet 就是在此基础上形成的。

分组交换网可以是专用的,也可以是公用的,一些工业发达国家已建立了不少公用分组交换网,与公用电话网相似,为更广大的用户服务。我国公用分组交换网(简称 CNPAC)于 1989 年 11 月建成。

3. 形成计算机网络的体系结构

计算机网络是一个非常复杂的系统,需要解决的问题很多,如何进行设计? 早在 ARPANET设计时,就提出了“分层”的方法,即将庞大而复杂的问题分为若干较小的易于处理的局部问题。1974 年美国 IBM 公司按照分层的方法制定了系统网络体系结构(System Network Architecture,SNA)。现在 SNA 已成为世界上较为广泛使用的一种网络体系

结构。

网络体系结构出现后,使得一个公司所生产的各种设备都能很容易地互连成网。这种情况显然有利于一个公司垄断自己的产品。用户一旦购买了某个公司的网络,当需要扩大容量时,就只能再购买原公司的产品。如果购买了其他公司的产品,那么由于网络体系结构不同,就很难互相连通。

然而,全球经济的发展使得不同的网络体系结构的用户迫切要求能够互相交换信息。为了使不同体系结构的计算机网络都能互连,国际标准化组织 ISO 于 1977 年成立了专门机构研究该问题。1978 年 ISO 提出了异种机连网框架结构,即著名的开放系统互连参考模型 OSI/RM(Open Systems Interconnection Reference Model)。OSI 得到了国际上的承认,成为其他各种计算机网络体系结构仿效的标准,大大地推动了计算机网络的发展。从这以后,就开始了所谓第三代计算机网络的新纪元。

在计算机网络发展的过程中,另一个重要的事件就在 20 世纪 70 年代末出现了局域网。局域网可使在一个局部的范围内(一个企业、学校、部门)的许多小型(或微型)计算机互连在一起,用于信息交换和资源共享。局域网连网简单,只要在小型(或微型)计算机中插入一个接口板(目前通称为网卡),就能接上电缆实现连网。由于局域网价格便宜,传输速率高,使用方便,在 20 世纪 80 年代得到了很大的发展。与此同时,微型计算机的大量普及对局域网的发展也起到了很大的推动作用。

OSI 从一张白纸开始,试图达到全世界的计算机网络都遵循这个统一的标准,因而全世界的计算机都能够很方便地进行互连和交换数据。在 20 世纪 80 年代,许多大公司甚至一些国家的政府机构都纷纷表示支持 OSI。当时看来在不久的将来全世界一定会都按照 OSI 制订的标准来构造自己的计算机网络。然而到了 20 世纪 90 年代初期,虽然整套的 OSI 国际标准都已制定出来了,但由于 Internet 已经覆盖了相当大的范围,而与此同时却几乎找不到有什么厂家生产出符合 OSI 标准的商用产品。因此人们得出这样的结论:OSI 事与愿违地失败了。

现在规模最大,覆盖全世界的计算机网络 Internet 并未使用 OSI 标准。OSI 失败的原因可归纳为:OSI 的专家们缺乏实际经验,他们在完成 OSI 标准时没有商业驱动力;OSI 的协议实现起来过于复杂,而且运行效率很低;OSI 标准的制定周期太长,因而使得按 OSI 标准生产的设备无法及时进入市场;最后,OSI 的层次划分并不太合理,有些功能在多个层次中重复出现。

4. 高速网络技术

从 20 世纪 80 年代末开始,计算机网络开始进入第四代,其主要标志可归纳为:网络传输介质光纤化,信息高速公路建设;多媒体网络及宽带综合业务数字网(BISDN)的开发应用;智能网络的发展;分布式计算机系统及集群(cluster)、计算机网格(grid)的研究促进了高速网络技术的发展和广泛的应用,并相继出现了高速以太网(所谓的千兆网)、光纤分布

式数据接口 FDDI、快速分组交换技术,包括帧中继和异步转移模式等。

5.1.4 计算机网络的构成

按照网络的地域覆盖范围,人们把计算机网络分为局域网(Local Area Network,LAN)、都市网(Metropolitan Area Network,MAN)、广域网(Wide Area Network,WAN)和网间网(Internet,又叫互联网)。无论是哪种网络,都可划分成两部分:主机(HOST)和子网(SUBNET),如图 5.1.3 所示。

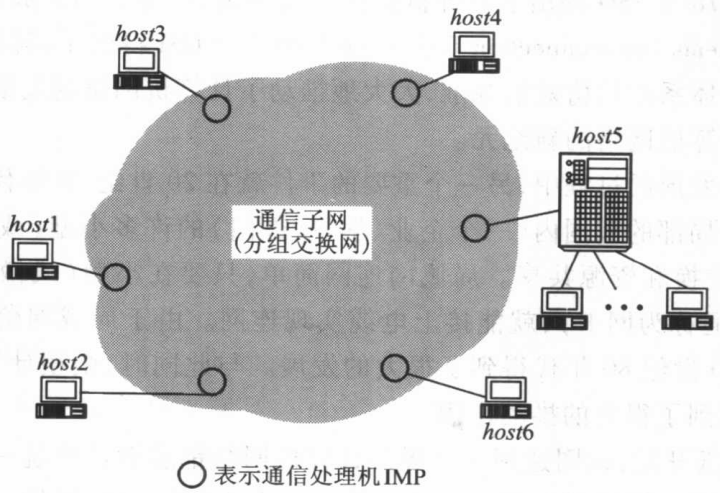


图 5.1.3 计算机网络的概念结构

主机是组成网络的独立自主的计算机系统,用于运行用户程序(即应用程序),也有些文献把它称为末端系统(End System,ES)。子网,严格地说,应叫通信子网(Communication Subnet),是将入网主机连接起来的实体。子网的任务是在入网主机之间传递信息,以提供通信服务。

在网络中,把纯通信部分的子网与应用部分的主机分离开是网络层次结构思想的重要体现,使得网络的设计与分析大为简化。

上述网络概念结构来自 ARPANET。ARPANET 是最早出现的重要网络之一,也是产生 TCP/IP 技术和最早应用 TCP/IP 技术的网络。ARPANET 关于计算机网络概念的划分有一个明显的缺陷,就是没有把网络结构与网络协议层次结合起来,这样容易造成一个误解,似乎在计算机网络中,主机不参与任何通信操作,这显然是不符合实际情况的。

为了克服上述缺点,不妨从另一个角度来讨论这个问题。从上述讨论已知,计算机网络是计算机技术与通信技术相结合的产物,它的主要目的是提供不同的计算机系统和用户间资源的共享。换句话说,在计算机网络中通信只是一种手段。在这个意义上,可以把计算机

网络划分为通信服务的提供者和通信服务的使用者两部分。通信服务的提供者包括网络层及以下各层,通信服务的使用者包括传输层及以上各层(尤其是应用层)。

为了便于讨论,将上述通信子网的概念加以拓展,把它作为通信服务的提供者,从通信子网的定义出发是完全合理的。那么,关于主机的概念就必须加以修改。因为在计算机网络中,任何一台主机都包括网络协议的全部层次(当然也包括通信子网的一部分),因此主机也包含通信服务的提供者和通信服务的使用者两部分。

在物理上,通信子网是由哪些部件组成的呢?不同类型的网络,其通信子网的组成各不相同。最简单的是局域网,其子网由传输介质(又叫传输线、线路、信道)和主机网络接口板组成。在以太网(Ethernet)中,传输介质可以是标准以太网电缆、双绞线和宽带电缆等。网络接口板可以是3Com公司的3C50X系列以太网卡,或者是NOVELL公司的NE2000以太网卡等。

在广域网中,通信子网除包括传输介质和主机网络接口板外,还包括转发部件。转发部件是一种专用计算机,连接两条或多条传输线,负责主机间的数据转发,相当于电话系统中的程控交换机。描述转发部件的术语很多,常见的有分组交换节点(Packet Switch Node,PSN)、中间系统(Intermediate System,IS)等。在ARPANET中把转发部件叫做接口报文处理机(IMP)等。在TCP/IP网中,转发部件相当于网关(Gateway)。关于转发部件在网络中的价值,用户不妨想象一下,假如在一个大型电话网中没有交换机或接线员,会出现什么样的情况呢?

5.1.5 计算机网络的分类

计算机网络可以从不同的角度进行分类。

(1) 按网络的拓扑结构分类

按网络的拓扑结构分类,计算机网络可分为星型网、树型网、总线网、环型网、网状网等。

(2) 按网络的使用范围分类

按网络的使用范围分类,可以划分为公用网和专用网。公用网(Public Network)一般是国家邮电部门建造的网络,为全社会的人提供服务。专用网(Private Network)是为某部门特殊业务工作的需要而建设的网络,不向外单位的人提供服务。例如军队、铁路等系统的网络均为专用网。

(3) 按网络的分布范围

按网络的分布范围的大小分类,可分为如下3类。广域网(Wide Area Network,WAN),分布范围通常为几十至几千公里。例如,一个国家或洲际网。广域网有时也称为远程网(Long Haul Network),传输速率往往在每秒钟几千比特以上。

局域网(Local Area Network,LAN),一般分布在较小范围内(如1 000 m左右),或为一个建筑物、一个工厂、一个单位内,为一个单位所有。它一般用微型计算机通过高速线路相

连,传输速率在 1 Mbps 以上。

城域网或市域网(Metropolitan Area Network,MAN),其分布范围在广域网和局域网之间,例如,范围是一个城市,其作用距离约为 5 ~ 10 km,传输速率也在 1 Mbps 以上。

不同的广域网、城域网或局域网还可以根据需要互相连接,形成规模更大的国际网。

5.1.6 Internet

(1) Internet 的概念

Internet 是个全球性的、最成功的、世界上最大的计算机网络。它分布于全球 100 多个国家,将几千个计算机网络连接在一起,拥有几千万个用户。中国是第 71 个国家级 Internet 网络成员。

Internet 使几千万用户遵守共同的协议,共享资源,由此形成了"Internet 文化",它是全人类最大的知识宝库。

Internet 是指通过网络互连设备把不同的网络和网络群体连接起来形成的大网络,也称网际网。而现在人们通称的 Internet 是指美国人建立起来的,目前已连接世界各国的一个特定的、被国际社会认可和广泛使用的网络。

(2) Internet 的形成

自 1969 年 ARPANET 间世以后,其规模一直快速增长,到 1983 年就已连上了 300 多台计算机,供美国各研究机构和政府部门使用。在 1984 年 ARPANET 分解成两个网络。一个仍为 ARPANET,是民用科研网。另一个是军用计算机网络 MIINET。

目前,Internet 已经成为世界上规模最大和增长速率最快的计算机网络,没有人能准确地说出 Internet 究竟有多大。Internet 的迅速发展始于 20 世纪 90 年代。由于欧洲原子核研究组织 CERN 开发的万维网(World Wide Web,WWW)被广泛使用在 Internet 上,大大地方便了广大非网络专业人员对网络的使用,成为 Internet 的这种指数级增长的主要驱动力。

(3) Internet 对人类的影响

Internet 是在人类对信息资源需求的推动下发展起来的,随着人类社会的发展,信息已经成为人类社会最重要和不可缺少的赖以竞争、生存、发展的资源。计算机网络技术的不断发展和完善,不仅极大地满足了人类对信息的需求,更重要的是加速和推动了人类社会的发展,使人类社会发生根本性的变革。Internet 为人类带来了各种利益,主要包括:

(a) 促进经济增长和提高生产率;创造就业机会;保持技术领先地位;推动区域性、地方经济的发展;推动电子商务、电子政务的发展。

(b) 推动医疗保健制度的改革。

(c) 改善为公众利益服务的网络。

(d) 促进科学研究。

(e) 推动教育事业的发展。

Internet 是全人类的资源，是事实上的世界各国的信息基础结构设施，它为全人类提供各种形式的信息服务。该网将对社会产生以下 12 个方面比较明显的影响：

(a) 传播媒介。“大众传播媒介”显示衰落迹象。电视观众越来越少。广播电台将填补适当的市场。报纸将越来越失去影响，在读者数量和广告收入方面都将减少。

通信网络能使这些趋势加快，这是由于通信网络有利于一些非常个人化的、越来越交互的媒介。这些“离开大众”的媒介能使用户有更多的自由选择和安排，节目包括有线电视、小型磁盘、计算机电子公告板、电子邮件、电子报纸、传真、大型传媒装置、音乐合成装置、联机数据库、卫星转播的远程电视教学、电子游戏和录像节目等。

(b) 数据检索。各种数据库的联机使一般公民的事务信息量不断扩大，使人们有可能迅速而便利地检索到大量的数据资料。

(c) 数据失真。各种信息能够便利方便地传输和复制，为选出的数据材料进行重新加工开辟了道路。计算机的编辑功能使人们能够改变数字数据、声频数据。结果造成令人烦恼的安全问题，不可能确保信息的准确性。

(d) 计算机犯罪。

(e) 获得信息的难度增加。信息是宝贵的财富，它可以在市场上销售，这样就会出现“信息禁令”，因为很有价值的信息的拥有者们已认识到它的价值，并且为了经济利益而采用保护措施或者提高信息价格，结果导致较贫困的个人或机构和国家难以获得信息。

(f) 信息贫乏者受歧视。“信息贫乏者”由于缺少信息操作技术，从而限制了他们获得就业的机会。电信革命对穷人和未受过教育的人们可能是沉重的打击。

(g) 推进改革。通信网络加快了信息传播，使政治、经济、组织结构的变革加快。

(h) 超越国籍。正如信息超越国界将改进政治结构一样，信息也将改变社团、宗教制度和文化传统。

(i) “香蕉”美元。通信网络有可能使全世界各地的巨额货币进行电子汇兑。“香蕉美元”的大量流动能够迅速动摇除了最庞大的国民经济以外的所有经济体系。

(j) 工作人员能够通过他们的个人电脑在家里办公。

(k) 电信社会。网络通信增多意味着人们更少进行面对面地交往。然而，随着出现高清晰度、交互性更强的视频通信，网络交互作用的质量将会提高。

(l) 技术化。通信网络要求使用者和他们的需要必须适合网络的规定。为了存储信息的简便，许多网络都把人简化为一系列的数字，如社会保险、住址、电话号码和传真号码。

(4) Internet 功能

Internet 与其他大多数现有的商业网络不同，它不是为某些专用的服务设计的，它既通用又高效。Internet 能够适应计算机、网络和服务的各种变化，它能够提供多种信息服务。

因此，Internet 成为一种全球信息基础设施。其提供的服务主要包括：电子邮件、电子公告牌、文件传输、远程登录、信息浏览、高级浏览、自动标题搜索、自动内容搜索、声音和视频通信及全球数字化信息库等。

5.2　协议与体系结构

计算机网络中的计算机是独立自治的，它们之间要正确地交换数据，就必须遵守一些事先约定好的规则，这就好像两个用汉语进行交谈的人，他们必须遵守汉语的语法规则，否则就无法进行交谈。为进行网络中的交换而建立的规则、标准、或约定称为网络协议。一个网络协议主要由以下 3 个要素组成：

（1）语法。数据与控制信息的结构或格式。

（2）语义。需要发出何种控制信息，完成何种动作以及做出何种应答。

（3）同步。事件实现顺序的详细说明。

所以，网络协议实质上是计算机间通信时所使用的语法规则，是计算机网络不可缺少的组成部分。

5.2.1　网络拓扑结构

计算机网络的拓扑结构实际上是信道分布的拓扑结构，常见的网络拓扑结构有 5 种：总线型、星型、环型、树型和网状型，如图 5.2.1 所示。

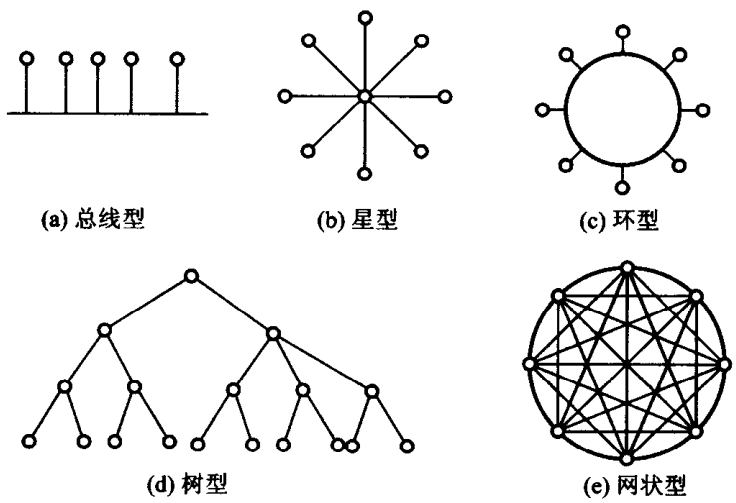


图 5.2.1　网络的拓扑结构

不同拓扑结构的信道访问技术、性能(包括各种负载下的延迟、吞吐率、可靠性以及信道利用率等)和设备开销都不相同,分别适用不同的场合。

尽管不同的拓扑结构的差别是很明显的,但总的来说可以分为两类:点到点信道和广播信道。

在点到点信道类型中,网络包含许多根电缆或租用的电话线路,每一根的两头连接一对 IMP。假如希望进行通信的两台 IMP 不在一条线上,那么它们必须经过其他 IMP 间接地进行通信。例如有 A、B 两个电话用户,他们的电话线不在一根电话线上,必须通过电信局的交换机转接才能通话。在计算机网络中,当一个用户经过某一台 IMP 发送一组相关信息到另一台 IMP 时,往往要经过多个中间 IMP 转发,每一个中间转发的 IMP 都将信息接收下来,并存放在 IMP 的内存中,直到所要求的线路空闲时再向前传送。按这种原理工作的子网叫点到点子网(或称存储转发子网)。采用点到点子网时就涉及网络的拓扑结构,图 5.2.1 中的环型、树型和网状型等网络拓扑结构适合点到点信道。

广播信道系统的固有特性是,任意一个 IMP 发出的信息(或报文)将被网上所有其他 IMP 所接收,在发出的信息中(或报文)必须有一些信息来说明它要发给谁,如果一台 IMP 收到的信息不是自己的,那么这台 IMP 只要不理睬它就可以了。图 5.2.1(a)图中给出的总线网络结构适应广播网。在总线网络中,任一瞬间只有一台 IMP 是总线控制者,它可以进行信息发送,而网上所有其他的 IMP 则禁止向网上发送信息。总线上必须有一种仲裁机构来解决多台 IMP 想要同时发送时引起的冲突。这种仲裁机构可以是集中的,也可以是分布的。

5.2.2 数据交换方式

交换(Switch)的概念最早来自电话系统,当用户发出电话呼叫时,电话系统中的交换机将在电话的呼叫者与接收者之间寻找一条客观存在的物理通路,用户的通路就建立起来。然后两端电话占有这条物理通路,直到用户挂断电话。这里的交换体现在交换设备(交换机)内部。

当交换机从一条输入线上接收到呼叫请求时,它首先根据被呼叫者的号码寻找一条空闲的输入线路,然后通过硬件开关将两者线路接通,假如一次电话呼叫要经过若干交换机,则所有交换机都要完成同样的操作。电话系统的交换方式叫线路交换。线路交换的外部表现是,通信两端一旦接通,便拥有一条实际的物理通道,双方占有此线路。

线路交换技术有两大优点:传输延迟短,唯一的延迟是电磁信号的传输时间;一旦线路接通,便不会发生冲突。第一个优点是,线路一旦连通,便不再需要连接开销,第二个优点是,双方独占物理线路。

线路交换的缺点首先是建立线路所需的时间很长,在数据传输之前,呼叫必须经过

若干中间交换机,最终才能传到被呼叫方,这个过程常常需要 10 s,甚至更长的时间。另外,线路交换独占线路造成信道的浪费。因为信道一旦建立起来,即便空闲也不能作为它用。当然这种浪费也有好处:对于独占信道的用户来说,其可靠性和实时响应的能力很优异。

另一种交换技术是报文交换(Message Switching)技术,报文交换技术不事先建立线路,当发送方有数据块要发送时,它把数据块作为一个整体(叫作报文,Message)交给交换设备(即 IMP),交换设备选择合适的空闲输出线,将数据块通过该输出线发送出去,在这个过程中,交换设备的输入线和输出线不建立物理连接,与线路交换一样,报文在传输过程中,也可能经过若干个交换设备。在每个交换设备处,报文首先被储备起来,然后在适当的时候转发出去。所以报文交换技术是一种存储转发技术(Store - and - Forward)。

另一种存储转发技术是分组交换(Packet Switching),上述报文交换技术没有对传输数据块的大小作限制,报文传输时,IMP 必须利用主机的磁盘作缓冲,单个报文可占用一对 IMP 线路长达几分钟的时间,这样显然不适应交互式通信。为了解决这个问题,分组交换技术严格限制数据块的大小,使分组可在 IMP 内部存放,保证任何用户都不能长时间(不超过几十毫秒)占用线路。因此非常适应于交互式通信。与报文交换相比,分组交换的另一个优点是吞吐率高,在具有多个分组报文中,第二个分组未接到之前,已经收到的第一个分组就可以往前发送。这样减少了时间延迟,提高了吞吐率。

分组交换除吞吐率较高外,还提供一定的程序校验和代码转换能力,它是绝大多数计算机网络所采用的技术。也有极少数计算机网络采用报文交换技术,但绝对不采用线路交换技术。

综上所述,交换方式的实质是交换设备内部将数据从输入线切换到输出线的方式,线路交换与存储转发的关键区别是:前者是静态分配线路,后者是动态分配线路。

通信子网绝大多数采用分组交换技术,根据通信子网内部机制的不同可以把分组交换子网分为两类:一类采用连接方式,一类采用无连接方式。在连接子网中,连接称为虚电路,类似电话系统中的物理线路;在无连接子网中,分组称为数据报(Datagram),类似于电信系统中的电报。

具体实现时,虚电路子网要求有一个建立虚电路的过程,并在各 IMP 上都有一个记录虚电路的 IMP 表,经过某 IMP 的虚电路在该 IMP 的 IMP 表中占一个表目,从信源(Source)到信宿(Destination)路径上所有的 IMP 表的相应表目串起来,便构成一条虚电路。这条虚电路在建立连接的过程中产生,在关闭连接时撤销。一对主机之间一旦建立了虚电路,分组即可按虚电路进行传输,不必给出显示的信宿地址,而且传输过程中也不必再为各分组单独寻址,所有的分组遵循既定的虚电路。除此之外,虚电路方式在内部还包括输入输出缓冲机制和可靠性机制。

数据报子网没有建立连接的过程,各数据报均带信宿的地址,传输时子网对各数据报单独寻址。

虚电路和数据报这两个概念不仅用于描述通信子网内部结构,实际上也是分组交换的两种形式,面向连接的分组交换技术通常称为虚电路,无连接的分组交换叫数据报,而虚电路一般用于描述通信子网,因为只有通信子网内虚电路才有所指,由一定的内部机制所支持;而数据报是不需内部支持。

综上所述,通信网络中的交换技术分为两类:线路交换和存储转发。存储转发又分为报文交换和分组交换,而分组交换又分为虚电路和数据报两种形式。

5.2.3 网络协议

(1) 网络协议

现代计算机网络是以高度结构化的方式进行设计的。为减少设计的复杂性,绝大多数网络是以一系列的层或级组成的,其中的每一层建立在前一层的基础上。层数、每一层的名称以及每一层的功能将随着网络的不同而不同。但是在所有的网络中,每一层的目的都是为高层提供一定的服务。

一台主机的第 N 层与另一台主机的第 N 层进行对话,在这种对话中使用的规则和约定的集合称为 N 层协议。图 5.2.2 所示为 ISO/OSI 7 层网络结构模型。

实际上,没有任何数据是从一台主机的第 N 层直接传送给另一台主机的第 N 层的(最底层除外),而是每层将数据和控制信息传递给紧接着它的下一层,直到最底层为止。在最底层实现与另一台主机的物理通信。它与高层之间进行的通信称为虚拟通信。在图 5.2.2 中虚拟通信用虚线表示,而物理通信(即最底层)用实线表示。

在每对相邻的层之间有一个接口,接口定义了较低层向较高层提供原始操作和服务,当网络设计者决定网络中应包含多少层和每层的任务时,最重要的是考虑每两层之间要有一个明确定义的接口。有了这个接口后,就可依次要求每一层执行一组已被充分理解的具体功能。除了减少在层间传递的信息外,明确划分的接口还易于用完全不同的一种方法来代替原来的那一层,例如所有的电话线路被微波线路所取代。

层和协议的集合称为网络体系结构,体系结构的说明必须包含足够的信息,以允许网络的实现者能编写完全符合协议的程序,至于执行细节和接口的规则都不是网络体系结构的范畴。网络结构和协议是计算机网络的基本研究课题。

(2) 国际标准化组织(ISO)参考模型

ISO(International Standard Organization)是国际标准化组织的缩写,是专门制定各种国际标准的组织。著名的开放系统互连参考模型就是 ISO 制定的有关通信协议的模型。设计此模型的目的是为设计协议簇的人们提供一个统一的结构。ISO/OSI 模型的结构如图 5.2.2 所示。

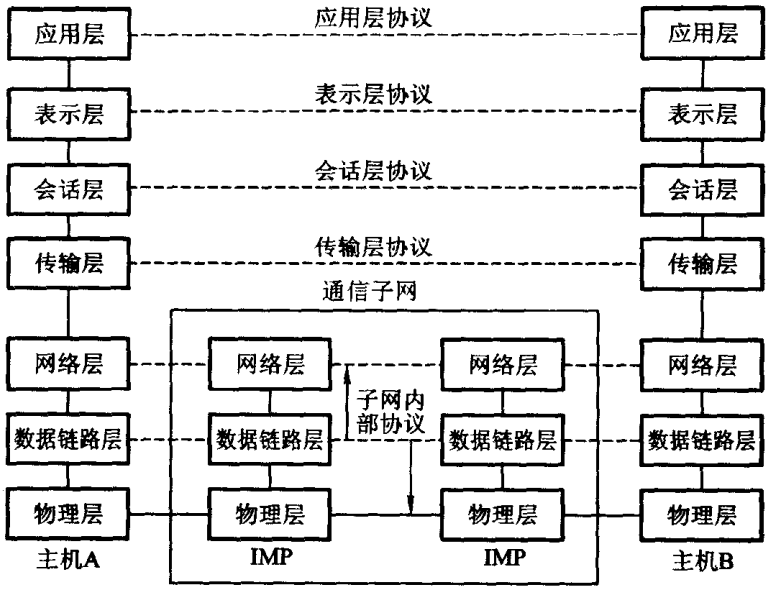


图 5.2.2 ISO/OSI 七层网络结构模型

以下依次讨论这 7 层协议。

① 物理层

物理层关系到在通信信道上传输的原始比特(bit)信号,在这一层中主要是处理与传输介质有关的机械的、电气的和与通信子网的接口问题。

② 数据链路层

数据链路层的任务是将一个原始的传送机构转换为对于网络层来说无传输错误的传输线。要完成这个任务,必须把输入数据分成若干个称为数据帧(或帧)的单位。每个数据帧包含要发送的数据、数据要达到的目的地址、源地址、帧类型和校验信息等。

数据链路层分为访问控制(MAC)和逻辑控制(LLC)两个子层,MAC 子层解决广播型网络中多用户竞争信道使用权的问题,LLC 的主要任务是将有噪声的物理通道变成无传输差错的通信通道,提供数据成帧、差错控制、流量控制和链路控制等功能。

③ 网络层

网络层(有时称通信子网层)负责将数据从物理连接的一端传到另一端,主要功能是路径的选择,以及与之相关的流量控制和拥塞控制等。

在这一层中通常还拥有一些会计的功能,记录每个用户发送了多少个包或多少个字或比特的信息,从而产生出有关收款的信息。

④ 传输层

传输层(又叫主机层)的主要功能是接收来自会话层的数据,把这些数据分成较小的单

元,再将它们传输到网络层,并保证所有数据块准确地到达另一端。

传输层通过向上提供一个标准的、通用的界面,使上层与通信子网(下三层)的细节相隔离,传输层的主要任务是提供进程间通信机制和保证数据传输的可靠性。

⑤ 会话层

若忽略某种数据变换的表示层,则会话层就是用户进入网络的接口。会话层主要针对远程终端访问,包括会话管理,传输同步以及活动管理等。会话一般是面向连接的,远过程调用(Remote Procedure Call,RPC)是个例外。

⑥ 表示层

主要功能是信息转换,包括信息压缩、数据加密、标准格式转换以及上述操作的逆操作等。

⑦ 应用层

向用户提供通用或常用的应用程序,例如电子邮件(E-mail)和文件传输。

(3) TCP/IP 协议

与国际标准化组织的 ISO 模型不同,TCP/IP(Transmission Control Protocol/Internet Protocol)不是作为标准制定的,而是产生于广域网的研究和应用实践中,并且是现在事实上的网络标准。在 5.3.3 节将详细介绍 TCP/IP 协议。

5.3 网络互连与 Internet

5.3.1 局域网技术

(1) 什么是局域网

局域网络技术是当前计算机网络技术领域中非常重要的一个分支。局域网作为一种重要的基础网络,在企业、机关、学校等各种单位和部门都得到广泛的应用。局域网还是建立互联网络的基础网络。

至今人们还很难给局域网一个严格的定义。但大多数人认为,局域网(简称 LAN,下同)是指在较小的地理范围内,将有限的通信设备互连起来的计算机网络。它只是一种通信网,仅有 OSI 参考模型的下三层,其覆盖范围处于广域网与多处理器计算机系统之间,约 10 m ~ 1 km 或更大一些。局域网具有以下几个特点:

① 共享传输信道。在局域网中几个计算机系统连接到一个共享的通信媒体上。

② 地理范围有限,用户数有限。通常局域网属于某个单位所有,只在一个相对独立的局部范围内联网,如在一座建筑物内、一个学校中等。其覆盖范围一般在 10 m ~ 1 km 内。

③ 传输速率高。局域网的传输速率一般在 1 Mbps ~ 1 000 Mbps,能支持计算机间的高速通信,所以延时较低。

④ 误码率低。因为是近距离的传输,所以误码率很低,一般在 $10^{-8} \sim 10^{-11}$ 之间,因而可靠性高。

⑤ 多采用分布式控制和广播式通信。在局域网中各结点是平等的而不是主从的关系。可以进行广播(一结点发,所有结点收)或组播(一结点发,多个结点收)。

(2) 局域网的体系结构

20 世纪 80 年代初期,美国电气和电子工程师学会 IEEE802 委员会首先制定出局域网的体系结构,即著名的 IEEE802 参考模型。其中包括 IEEE802.3(CSMA/CD)、IEEE802.4(令牌总线)、IEEE802.5(令牌环)。著名的以太网(Ethernet)就是 IEEE802.3 的一个典型产品,也是当前使用得最为广泛的一种计算机局域网。许多 802 标准现在都成为 ISO 国际标准。

由于局域网只是一个计算机通信网,而且局域网不存在路由选择问题,因此它不需要网络层,而只有最低的两个层次,物理层和数据链路层。然而局域网的种类繁多,其媒体接入控制方法也各不相同。为了使局域网中的数据链路层不过于复杂,就将局域网中的数据链路层划分成两个子层,即媒体访问控制 MAC (Medium Access Control)子层和逻辑链路控制 LLC(Logical Llink Control)子层。

(3) 局域网硬件的基本组成

局域网主要由网络服务器、用户工作站、网络适配器(网卡)、传输媒体等部分组成。

① 网络服务器。对局域网来说,网络服务器是网络控制的核心。一个局域网至少有一个服务器,特别是一个局域网至少要配一个文件服务器。文件服务器要求由高性能、大容量的计算机来担任,如微机局域网的文件服务器通常由配备大容量存储器的高档微机担任。文件服务器的性能直接影响整个局域网的性能。

② 工作站。在网络环境中,工作站是网络的前端窗口,用户通过工作站来访问网络的共享资源。在局域网中,工作站可以由计算机担任,也可以由输入输出终端担任,还可以用网络计算机(Network Computer)充当。对工作站的性能要求主要根据用户的需求而定。

③ 网卡。在局域网中,从功能的角度上说,网卡相当于广域网中的通信控制处理机,工作站或服务器连接到网络上,实现网络资源共享和互相通信都是通过网卡实现的。网卡对于局域网中的工作站和服务器来说是不可缺少的设备。

④ 传输媒体。传输媒体是网络通信的物质基础之一。传输媒体的性能特点对信息传输速率、通信的距离、连接的网络结点数目和数据传输的可靠性等均有很大的影响。因此,必须根据不同的通信要求合理地选择传输媒体。目前局域网中使用的传输媒体主要是双绞线,也可以采用同轴电缆、光缆。

(4) 网络操作系统

要使局域网满足用户的需求，必须有合适的高层软件，包括各种局域网系统软件和应用软件，本节将对局域网的系统软件——网络操作系统(NOS)的一些原理进行简单的介绍。

局域网的网络操作系统是网络用户与计算机网络之间的接口，网络用户通过网络操作系统请求网络服务。网络操作系统与一般概念操作系统的最大区别是，增加了对网络资源管理和提供用户请求网络服务的功能。网络操作系统除了处理机管理、存储管理、设备管理、作业管理、文件管理的功能外，还应具有网络设备管理和网络管理的功能。目前流行的操作系统如 Windows(9x/2000/XP)、UNIX、Linux 等都是网络操作系统。

5.3.2 网络互连

(1) 网络互连概念

局域网虽然给一个单位、一个部门的应用带来了极大的便利，但更加广泛地应用必然要求跨部门、跨地区甚至跨国界的网络互连，以便实现在更大范围内的信息共享，例如，无纸贸易、国际间的电子邮件、数据信息查询与检索。因此，不同类型的网络互连是网络发展的必然趋势。

ISO 提出的 OSI 参考模型，目的是解决世界范围内网络的标准化，旨在使一个遵守 OSI 标准的系统可以和位于世界任何地方，也遵守同一标准的其他任何系统进行通信。但世界上存在许多非 OSI 体系结构的网络。相同网络的互连是比较容易的，而异构网络的互连就要复杂得多，因此，讨论网络互连技术，实质上是异构网络互连。

网络互连的具体方法有很多种，但总的来说，进行网络互连时应当做到以下几点：

① 在网络之间至少提供 1 条连接的链路及链路控制规程。

② 在不同的网络之间提供合适的路由，以便交换数据。

③ 对用户使用互联网络提供计费服务，它记录着不同网络和不同网关的使用情况，并且维护这些状态信息。

在提供上述服务时，要求不改变原有网络体系结构，能适应各种差异：不同的寻址方案，不同的最大分组长度，不同的网络访问控制方法，不同的超时控制，不同的差错恢复方法，不同的状态报告方法，不同的路由选择方法，不同的用户访问控制，不同的服务(面向连接的服务和无连接的服务)，不同的管理与控制方式以及不同的传输速率等。

(2) 网络互连方案

网络互连时，一般不能简单地直接相连，而是通过一个中间设备互连。这个中间设备称为中继(Relay)系统。在两个网络的连接路径中可以有多个中继系统。如果某中继系统在进行信息转换时与其他系统共享共同的第 N 层协议，那么这个中继系统就称为第 N 层中继系统。这样就可以把中继系统划分成以下 4 种：

① 物理层的中继系统。中继器(或转发器)是用于同种网络的物理层的中继系统，对接收信号进行再生和发送，扩大一个网络的作用范围。

② 数据链路层的中继系统。网桥(桥接器)是在数据链路层对帧信息进行存储转发的中继系统。

③ 网络层的中继系统。路由器(router)在网络层存储转发分组。

④ 比网络层更高的层次上的中继系统网关。网关(Gateway),又称网间连接器、信关或联网机。网关是对传输层及传输层以上的协议进行转换,它实际上是一个协议转换器。它可以是双向的,也可以是单向的。网关是中继系统中最复杂的一种。

互连各种类型的网络,可采用两种方式:利用中继系统实现网络互连和通过互联网实现网络互连。

广域网互连,一般使用路由器或网关在网络层实现以下各层协议的转换,广域网互连方式分为无连接和面向连接两种。

当今流行的 Internet 就是采用无连接的网际互连方式。无连接互连方式是采用互联网协议 IP,通过 IP 路由器将网络互连起来。这里加上"无连接"是为了强调这种互连方式向传输层提供的是无连接服务。换言之,不论互联的各网络之间有多少差异,当上升到网络层,整个网络都按照同一个 IP 协议来工作,向传输层提供无连接服务。

5.3.3 TCP/IP 协议

与 ISO 的 OSI 参考模型不同,TCP/IP 不是作为标准制定的,而是产生于广域网的研究和应用实践中,并且是当今事实上的网络标准。目前流行的 Internet 就是采用 TCP/IP 协议。

TCP/IP 是一个完整的协议簇,TCP/IP 协议族主要包括网际协议 IP,传输控制协议 TCP、用户数据报协议 UDP、反向地址转换协议 RARP、网际控制报文协议 ICMP 等。其中 TCP、IP 是该协议簇中两个主要协议,所以人们把这个协议簇称为 TCP/IP 协议。

TCP/IP 开发较早,并不完全符合 ISO 的开放系统互连模型。大致上说,TCP 对应 ISO/OSI 的传输层,IP 对应网络层,TCP/IP 与 ISO/OSI 的对应关系如表 5.3.1 所示。

目前广为流行的微机操作系统 UNIX、Linux、Windows 都实现了 TCP/IP。下面介绍 TCP/IP 协议,注意它与 ISO/OSI 的差别。

表 5.3.1 ISO/OSI 与 TCP/IP 的对应关系

ISO	TCP/IP
应用层	Telnet, FTP, tftp
传输层	TCP 和 UDP
网络层	IP 和 ICMP
数据链路层	ARP,RARP 设备驱动程序
物理层	电缆或别的设备如(Ethernet 网板)

(1) 物理层

物理层是在协议模型的硬件级上,它负责处理电子信号。物理层协议发送和接收"包"(Packet)格式的数据,一个"包"包括源地址、要传送的数据和目的地址。TCP/IP 支持许多不同的物理层协议,其中包括以太网。

(2) 数据链路层

数据链路层关心的是物理级的地址,此层协议涉及各种通信控制器,以及这些控制器的芯片和缓冲区。在这层,TCP/IP 支持以太网,另外两个 TCP/IP 协议——ARP 和 RARP,可以看作是网络层和数据链路层之间的协议。ARP 是以太网的地址转换协议,它把已知的 IP 地址(32 b)映射成为以太网的地址(48 b),RARP(或 Reverse ARP)是 IP 地址转换协议,它把已知的以太网地址(48 b)转换为 IP 地址(32 b),恰恰与 ARP 相反。

(3) 网络层

IP 和 ICMP 是网络层上的两个协议,IP 提供主机与主机之间的通信,它通过主机接收的 IP 地址决定传输路径的方法来完成传输路径的选择,IP 还传输格式服务,它把传送的数据组装成数据报的格式。如果数据报是向外发送的(即是从较高层协议收到的),则把 IP 头(IP header)加到此数据报上。IP 头包含许多信息,其中包括发送和接收主机的地址。

网络控制报文协议 ICMP 允许主机发送控制报文(也叫差错报文)到其他主机,ICMP 为一台计算机的网络软件与另一台计算机的网络软件提供通信。ICMP 报文在 IP 数据报的数据部分之中,像所有其他流通信息一样,经过网络进行传输。ICMP 报文的最终报宿不是接收结点主机的一个用户,而是那个计算机上的进程。也就是说,当一个有错误的 ICMP 报文到达时,IP 软件模块就会处理这个错误,不会把错误的 ICMP 报文传送给相应的应用程序。

(4) 传输层

TCP/IP 的传输层协议能使运行在不同机器上的进程进行通信,这一层的协议有 TCP 和 UDP。

TCP 是传输控制协议,是面向连接的,因而可提供可靠的、按序传送数据的服务。TCP 传输数据首先在通信的双方之间建立连接,然后进行数据传输,数据传输完成后撤销连接。TCP 不提供广播和多播服务,由于 TCP 要提供可靠的数据传输服务,因此,TCP 就不可避免地增加了许多开销,包括链路的建立与撤销、应答、流量控制、定时器等。

UDP 是数据报协议。UDP 在数据传输之前不需要建立连接,远地的主机在接收到 UDP 数据报后,也不需要向发送方给出任何应答。相对 TCP 而言,UDP 是一种高效的通信协议。

(5) 应用层

应用层协议包含多种应用协议,主要包括:

- TELNET,简单远程终端协议。
- FTP,文件传输协议。
- SMTP,简单邮件传送协议。

5.3.4 Internet 编址与地址解析

(1) IP 地址

Internet 是通过网关(或 IP 路由器)将各种物理网络互连在一起的虚拟网络,而一个物理网络中各站点都有一个可识别的地址,这个地址称为物理地址。物理地址的格式、长度等由物理网络所采用的技术决定,如果对地址不进行统一管理,那么同一类型不同物理网络上的站点可能拥有相同的物理地址。Internet 对物理地址的"统一"是在 IP 层完成的,IP 协议提供整个 Internet 通用的格式,该地址格式采用分层结构,由网络地址和主机地址两部分组成,如图 5.3.1 所示。网络地址用来标识连入 Internet 的网络,主机地址标识特定网络中的主机(包括网关)。为了确保一个 IP 地址仅对应一台主机,其网络地址由 Internet 注册管理机构网络信息中心(Network Information Center,NIC)分配,而主机地址由网络管理机构负责分配。

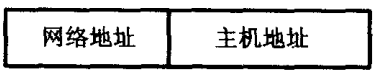


图 5.3.1　IP 地址结构

IP 地址长度为 32 bit,以 X.X.X.X 格式表示,X 为 8 bit,其值为 0 ~ 255,这种格式的地址称为点分十进制地址。

IP 地址分成 5 类,如图 5.3.2 所示。其中 A 类、B 类和 C 类为主类地址,D 类和 E 类为次类地址,用户可根据需要申请不同的 IP 地址。IP 地址的前 5 bit 用于标识地址的类型,例如,A 类地址的第一个比特为"0",B 类地址的前两个比特为"10"等。

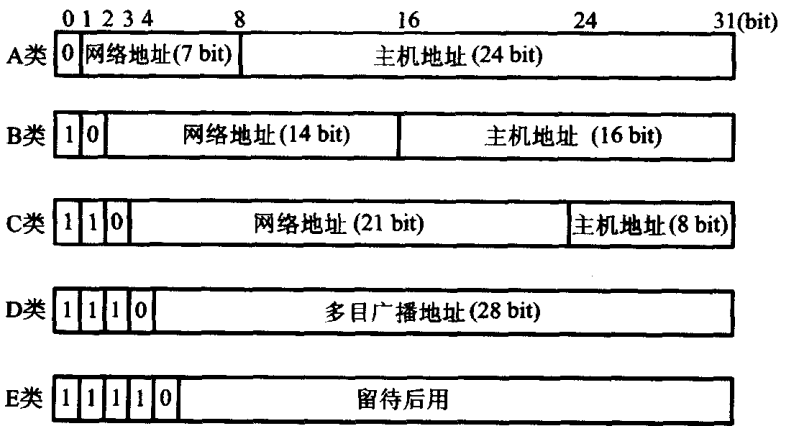


图 5.3.2　IP 地址类型

对于 A 类地址，网络地址空间 7 bit，允许 126 个不同的 A 类网络，起始地址为 1 ~ 126，0 和 126 两个地址用于特殊的目的。每个网络的主机地址数多达 2^{24}（16 000 000）个。也就是说主机的地址范围为：1.0.0.0 ~ 126.255.255.255，适用于有大量主机的大型网络。

B 类地址，网络地址空间为 14 bit，允许 2^{14}（16 384）个不同的 B 类网络，每个网络能容纳 2^{16}（65 536）台主机。地址范围是：128.0.0.0 ~ 191.255.255.255。这种编址适应于国际性大公司和政府机构等。

C 类地址的网络地址空间为 21 bit，允许多达 2^{21}（209 715）种不同的 C 类网络，地址范围为：192.0.0.0 ~ 223.255.255.255，每个 C 类网络能容纳 256 台主机。

D 类地址不标识网络，其起始地址从 224 ~ 239，主机地址范围为：224.0.0.0 ~ 239.255.255.255。用于特殊用途，例如多目广播。

E 类地址暂时保留。

从点分十进制地址便可识别 Internet 的地址类型，例如，126.1.1.1，从第一个十进制数“126”就知道它是一个 A 类地址，但 128.2.34.123 是一个 B 类地址。

IP 地址中包括网络地址，这种地址确定了主机连接在哪个网络上，这样，当网关连到两个网络上时，选择哪个网络地址作为网关 IP 的网络地址呢？Internet 的做法是网关所连的每一个网络都给网关分配一个 IP 地址，网关连接多少个网络，便拥有多少个 IP 地址，事实上同一网关的不同 IP 地址，对寻径来说是有利的。同理，有些主机也像网关一样具有两个或两个以上的物理连接，这种主机叫“多穴主机”，多穴主机有多个 IP 地址。每个 IP 地址对应一个物理连接。

在实际应用中，仅靠网络地址来划分网络会有许多问题，比如，A 类地址和 B 类地址都允许一个网络中包含大量的主机，但实际上不可能有这么多主机连接到一个网络中，这不但降低了 Internet 地址的利用率，还会给网络寻址和管理带来了很大的困难。解决该问题的办法是引入子网的概念，也就是将主机地址划分成子网地址和主机地址。通过灵活地定义子网地址的位数来控制每个子网的规模。这样，主机属于子网，若干个子网共享同一个 IP 网络地址。将一个大的 IP 网络划分成若干个既相对独立又相互联系的子网后，对外仍是一个单一的 IP 网络。IP 网络外部并不需要知道网络内部子网的划分细节，但 IP 网络内部各子网需要独立寻址和管理，子网间通过子网路由器相互连接，便于解决网络寻址和网络安全等问题。

判断两台主机在同一子网中，需要用到子网掩码。子网掩码的长度仍为 32 bit，只是网络部分（包括 IP 网络和子网）全为“1”，主机部分全为“0”。判断两个 IP 地址是否在同一个子网中，只要判断这两个 IP 地址与子网掩码做逻辑“与”运算的结果是否相同，若相同说明在同一个子网中，否则不在同一个子网中。

（2）地址解析

IP 地址统一了不同的物理地址，但是这种统一仅表现在自 IP 层开始的以上各层使用统

一格式的 IP 地址,将物理地址隐藏起来,实际上对各种物理地址并没有做任何改动。在物理网络的内部仍然使用各自的物理地址。由于物理网络技术的多样性,决定了物理网络地址规范的五花八门。这样,在 Internet 中就存在着 IP 地址和各物理网络的网络地址。要把这两种地址统一起来,就必须建立二者之间的映射关系。IP 地址与物理网络的物理地址之间的映射称为地址解析(resolution),包括两方面的内容:从 IP 地址到物理地址和从物理地址到 IP 地址的映射。为此,TCP/IP 专门提供了两个协议:地址解析协议 ARP(Address Resolution Protocol),用于从 IP 地址到物理地址的转换;逆向地址解析协议 RARP(Reverse Address Resolution Protocol),用于将物理地址转换成 IP 地址。

5.4 Internet 的应用

Internet 的应用已经渗透到人们当今生活的各个领域,正在改变人们的生活和工作习惯及方式。但从形式上可将 Internet 的应用分为工具、讨论、信息查询和信息广播等 4 类。工具类的应用包括远程登录(TELNET)、远程文件传输(FTP)、电子邮件(E-mail)、文件寻找(ACHIE);讨论类应用包括电子公告板(BBS)网络新闻论坛(NET - news 或 USENET)、实时在线交谈(IRC)、视频会议(MS Netmeeting);信息查询类的应用包括 Gopher 分布式文件查询系统、WAIS 广域信息服务、WWW 万维网超文本查询系统;信息广播包括在线语音和电视广播。

5.4.1 域名结构与域名系统

在 Internet 中为了屏蔽不同网络的物理地址的差别,在 IP 层使用 IP 地址来标识主机,但这种数字型的地址既难记忆也难理解。为了向用户提供直观的主机标识符,TCP/IP 专门设计了一种层次型名字管理机制,即域名系统 DNS(Domain Name System)。它包括字符型的层次式主机命名机制和名字与地址映射的分布式计算机的实现。

(1) 域名结构

Internet 允许用户为自己的计算机命名,用输入的名字来代替 IP 地址。但要求所命名的主机全局唯一,便于管理,方便映射。因此,任何连接在 Internet 上的主机或路由器都有一个层次结构的名字,称为域名。这里,"域"是名字空间中一个可被管理的划分。域还可以被划分为子域,如二级域、三级域等。域名结构由若干个分量组成,各分量之间用小数点隔开,例如:

三级域名.二级域名.顶级域名

各分量分别代表不同级别的域名。每级的域名都由英文字母和数字组成(不超过 63 个字符,并且不区分大小写字母),级别最低的域名写在最左边,而级别最高的域名(或称顶

级域名)写在最右边。一个完整的域名不超过 255 个字符。域名系统既不规定一个域名需要包含多少个下级域名,也不规定每一级域名代表的含意。各级域名由其上一级的域名管理机构管理,例如华中科技大学校园网的域名是 hust. edu. cn,如果要申请校园网内的域名,就得向华中科技大学校园网管理中心申请,而最高的顶级域名则由 Internet 的有关机构管理。用这种方法可使每一个名字都是唯一的,并且可以设计出一种查找域名的机制。注意:域名只是个逻辑概念,并不反映出计算机所在的地理地点。

现在顶级域名(TLD,Top Level Domain)有 3 类。

① 国家顶级域名 nTDL,采用 ISO3166 的规定。例如:cn 表示中国,us 表示美国等。

② 国际顶级域名 iTDL,采用 int。国际组织 int 下注册。

③ 通用顶级域名 gTDL,根据[RFC1591]规定,最早的顶级域名共 6 个,它们是:

com	表示公司企业
net	表示网络服务机构
org	表示非赢利性组织
edu	表示教育机构
gov	表示政府部门(美国专用)
mil	表示军事部门(美国专用)

由于 Internet 上用户急剧增加,现在又新增了 7 个通用顶级域名:

fire	表示公司企业
shop	表示销售公司企业
web	表示突出万维网活动的单位
arts	表示突出文化、娱乐活动的单位
rsec	表示突出消遣、娱乐活动的单位
info	表示提供信息服务的单位
nom	表示个人

表示国家顶级域名下注册的二级域名由该国自行规定,我国将二级域名划分为类别域名和行政区域名两大类。其中类别域名 6 个,分别为:ac 表示科研机构;com 表示工、商、金融等企业;edu 表示教育机构;gov 表示政府部门;net 表示互联网络、接入网络的信息中心(NOC);org 表示各种非赢利性的组织。区域域名 34 个,适用于我国各省、自治区和直辖市,例如:hb 表示湖北省,bj 表示北京市,sh 表示上海市等。

例 1:cs. hust. edu. cn

cs:计算机学院　hust:华中科技大学　edu:教育网　cn:中国

例 2:mail. hust. edu. cn

表示华中科技大学电子邮件系统的域名。

(2) 域名解析

域名解析就是将域名转换成 IP 地址的过程。是由 Internet 系统中的 DNS 完成的。Internet 的 DNS 设计成一个联机分布式数据库系统,并采用客户/服务器模式。DNS 使大多数名字都在本地映射,仅少量映射需要在 Internet 上通信,使得系统是高效的。DNS 也是可靠的,即使单个计算机出了故障,也不会妨碍系统的正常工作。名字到域名的映射是由若干个域名服务器程序组成的。域名服务程序在专设的结点上运行,而人们也常常把运行域名服务程序的计算机称为域名服务器。

将域名转换成 IP 地址的过程如下:当一用户进程需要将主机名映射为 IP 地址时,该应用程序就成为域名系统 DNS 的一个客户,并将待转换的域名放在 DNS 请求报文中,以 UDP 数据报方式发给本地的域名服务器(使用 UDP 是为了减少开销)。本地域名服务器在查找到域名后,将对应的 IP 地址放在应答报文中返回,应用进程获得目的主机的 IP 地址后即可进行通信。

若本地域名服务器不能回答该请求,则此域名服务器就暂时成为 DNS 中另一个客户,也将待转换的域名放在 DNS 的请求投放中,以 UDP 数据报的方式发给上一级的域名服务器。直到找到能够回答该请求的域名服务器为止。

5.4.2 远程登录 TELNET

TELNET 是一个简单的远程终端协议。用户使用 TELNET 就可以在其所在地通过 TCP 连接注册(或登录)到远地的另一个主机上(使用主机名或 IP 地址)。TELNET 能把用户的击键传到远地的主机,同时也能把远地主机的输出通过 TCP 连接返回到用户屏幕。这种服务是透明的,因为用户感觉好像键盘和显示器就连接在远地的计算机上。

5.4.3 文件传输协议 FTP

FTP 是 Internet 上使用最早和最重要的应用之一,采用客户/服务器工作方式。它的作用是在不同的计算机系统间传送文件,而与主机的类型无关,文件的种类不限。文件的传输既可以从远程的主机往本地的主机上传输,称为“下载”,也可以从本地主机往远程的主机上传输,称为“上传”。FTP 提供非常丰富、功能强大的命令集,其中最基本的是文件传输和文件管理命令。

FTP 的使用很简单,首先通过 ftp://iccc.hust.edu.cn 命令进入相应的 FTP 服务器,然后输入用户名和密码。系统核准后,就可以使用 FTP 的命令进行文件的传输和管理。FTP 的命令集与单机的文件操作命令几乎相同。

5.4.4 电子邮件

电子邮件是 Internet 上使用最多和最受欢迎的一种应用。电子邮件可以把邮件发送到收信人的邮箱中,收信人可随时收取。电子邮件不仅使用方便,而且费用低廉,现代的电子

邮件不仅可以传送文字信息,而且可以附上声音和图像。由于电子邮件的广泛使用,现在的人们很少通过邮局去发信和发电报。

用户如果希望使用电子邮件,可以申请一个电子邮件地址,例如向华中科技大学校园网申请一个电子邮件地址,就会得到一个邮件地址 XXXX@ mail. hust. edu. cn 和相应的密码。然后在用户的计算机上安装电子邮件系统,就可以使用电子邮件系统发送和接收邮件。也可以使用 Web 浏览器的方式来收发电子邮件。电子邮件地址中的 XXXX 是用户名(或称账号)。

5.4.5 WWW 超文本查询系统

万维网(WWW,world wide web)是 Internet 上的一个超文本信息查询工具。

超文本与传统的文本不同,传统的文本是线性结构,只能顺序阅读,没有选择余地,而超文本是一种非线性结构,制作超文本时,将写作素材按其内部的联系划分成不同层次、不同关系的信息单元,然后用超文本语言将其链接成一个网状结构的文件集合。

超文本由一些相对独立的信息单元和表达信息单元之间关系的链接组成。这些信息单元又叫结点或 Web 页(pages),结点的信息可以是文本、声音和图像,每个结点表达一个特定的主题,其大小视实际情况而定,通过超文本文件中的某些词或词组把相关的结点链接起来,每个链接有一个相应的指针(另一个文件的地址)。这样,链接使得结点或 Web 站点形成网状的信息网格。

结点、链接和信息网格构成超文本的三要素。其中链接是超文本的核心,功能直接影响结点信息的表现能力和信息网格的结构。当使用 WWW 阅读超文本文件时,屏幕上会出现许多不同的词或词组,这些被点亮了的词或词组就是链接,用鼠标单击被点亮了的词或词组之后,便显示词或词组所对应文件的内容。因此,浏览者可以有选择地阅读,这个阅读是有选择的非线性的,类似于联想思维模式,从而提高了人们获得知识的效率。

超文本标记语言(Hyper Text Markup Language,HTML)是一种专门用于 WWW 的编辑语言。用于创建 Web 文档,对 Web 的内容、格式及 Web 页中的超级链接进行描述。浏览器就是用于读取 Web 网点上的 HTML 文档,显示相应的 Web 页面。超级文本传输协议(Hyper Text Transfer Protocol,HTTP)则是浏览器与 Web 服务器之间进行通信的协议。

WWW 也采用客户/服务器的工作方式,它的客户端软件是 WWW 浏览器,WWW 服务器则运行着 WWW 服务程序。用户使用浏览器向服务器发出查询请求时,服务器则查询所有存储在服务器内的信息,如果所查询的信息没有在服务器上,那么这台服务器负责与其他服务器连接,并把结果通知浏览器,显示给用户。

目前流行的浏览器是微软公司的 Internet Explorer。

5.5 计算机网络安全

在计算机网络系统中,多个用户同处在一个大环境中,系统资源是共享的,用户终端可以直接访问网络和分布在各用户主机中的文件、数据和各种软件、硬件资源。随着计算机和网络的普及,政府、军队的核心机密和重要数据、企业的商业机密、甚至是个人隐私都存储在互连的计算机中,而因系统的原因和不法之徒千方百计地"闯入"、破坏,使有关方面蒙受巨大的损失。因此,网络安全已成为计算机领域中最重要的研究课题之一。

5.5.1 计算机网络面临的安全威胁

在计算机网络中,安全威胁来自以下 4 个方面。

① 截获。两个用户通过计算机网络进行通信时,如果不采取任何保密措施,那么网上其他的用户就可能"偷听"到他们的通信内容。

② 中断。当用户正在通信时,有意的破坏者可设法中断他们的通信。

③ 篡改。用户甲给用户乙发 1 份报文"命令你部 8 月 10 日 18 时到达 A 号地区",报文在转发过程中经过用户丙,用户丙将报文改为"命令你部 8 月 10 日 18 时到达 C 号地区"。这样,报文就被篡改了。

④ 仿造。当用户甲与用户乙用电话进行通信时,用户甲可以通过声音来确认对方。但用计算机进行通信时,若用户甲的屏幕上显示"我是用户乙"时,用户甲如何确定是用户乙而不是别人。又如用户甲是网络上的合法用户,一个非法的用户获得了用户甲的权限,也以用户甲的合法身份使用计算机网络中的信息。

以上 4 种对网络的威胁可以划分成两大类:被动攻击和主动攻击。

截获信息的攻击称为被动攻击,攻击者只是观察某一协议数据单元 PDU 而不干扰信息流,即使这些数据对攻击者来说是不易理解的,他也可能通过观察 PDU 的协议控制信息部分,了解正在进行的通信实体的地址和身份,研究 PDU 的长度和传输的频度,以便了解所交换的数据的性质。这种被动攻击又称为通信量分析。

更改信息和拒绝用户使用资源的攻击称为主动攻击,是指攻击者对某个连接中通过的 PDU 进行各种处理,如有选择地更改、删除、延迟这些 PDU(当然包括记录和复制);还可以在稍后的时间将以前记录下的 PDU 插入到这个连接(即重放攻击);甚至还可以将合成的或仿造的 PDU 送入到下一个连接中去。

所有主动攻击都是上述各种方法的某种组合。但从类型上看,主动攻击又可进一步分为 3 种。

① 更改报文流。更改报文流包括对通过连接的 PDU 的真实性、完整性和有序性的攻

击。对真实性的攻击可以是更改 PDU 中协议控制信息,这样报文就被送往错误的目的地。对完整性的攻击可以是更改 PDU 的数据部分,对有序性的攻击则可用删除 PDU 或更改 PDU 中协议控制部分的序号来实现。

② 拒绝报文服务。拒绝报文服务是指攻击者或者删除通过某一连接的所有 PDU,或者将双方或单方的所有 PDU 加以延迟。

③ 仿造连接初始化。攻击者重放以前已被记录的合法连接初始化序列,或者仿造身份而企图建立连接。

对于主动攻击,可以采取适当措施加以检测。但对于被动攻击,通常却是检测不出来的。根据这些特点,可以得出计算机网络安全的 5 个目标。

- 防止析出报文内容。
- 防止信息量分析。
- 检测更改报文流。
- 检测拒绝报文服务。
- 检测仿造初始化连接。

对于被动攻击可采用数据加密技术,而对付主动攻击,则需要加密技术和适当的鉴别技术。

还有一种特殊的攻击就是恶意程序(rogue program)攻击。恶意程序攻击种类繁多,对于网络安全威胁较大的有以下多种:

① 计算机病毒(computer virus)。一种会"传染"其他程序的程序,"传染"是通过修改其他程序来把自身或其变种复制进去完成的。

② 计算机蠕虫(computer worm)。一种通过网络的通信功能将自身从一个结点发送到另一个结点并启动运行的程序。

③ 特洛伊木马。一种程序,它执行的功能超出了其声称的功能。如编译程序除了执行编译任务外,还把用户的源程序偷偷地复制下来,则这种编译程序就是一种特洛伊木马。将一恶意程序隐藏在某个程序中,当用户运行这个程序时,恶意程序也启动运行,对用户的计算机硬件、软件和数据进行破坏。

④ 逻辑炸弹。一种当运行环境满足某种特定条件时才启动运行的恶意程序。如隐藏在 Word 中的某个逻辑炸弹,在条件不具备时,Word 运行时系统相安无事,当某个特定的条件满足时,逻辑炸弹程序开始运行,或者删除系统中的某些甚至全部文件,或删除、篡改数据库中的数据,或破坏系统的软、硬件设备。

这里讨论的计算机病毒是狭义的,也有人把所有的恶意程序泛指为计算机病毒。由于篇幅的限制,本书只给出这些最基本的概念。

5.5.2 计算机网络安全的内容

(1) 安全与保密

计算机网络安全是指网络系统中用户共享的软、硬件资源是否安全,不受到有意和无意的各种破坏,不被非法侵用等,研究计算机网络安全问题必然要涉及到保密问题。尽管计算机网络安全不仅仅局限于保密,但在研究计算机安全问题时,针对非法侵用、盗窃机密等方面的安全问题,要用保密技术加以解决。

保密是指为维护用户自身的利益,防止非法侵用和防止盗窃资源,使非法用户不能使用和盗取不到,或非法用户即使盗得资源也无法识别。保密一直是数据处理系统和通信系统中一个重要的研究课题,它涉及物理方法、存取数据的管理和控制、数据加密解密等数据安全机构。密码技术是实现保密与安全的有效办法。

密码技术分加密和解密两部分。加密是把需要加密的报文按照以密码密钥(简称密钥)为参数的函数进行转换,并产生密码报文。解密是按照密钥参数将相应的密文转换成明文的技术。目前常用的密码方法有 3 种:代码转换法、转换密码法和数据加密标准 DES。其中 DES 是由 IBM 公司研制,于 1997 年被美国联邦政府确定为标准,在国际上引起了极大的重视。DES 加密标准采用多层次的复杂的数据替换,使加密后的密文几乎不能被破译。

加密的方法分为通信加密和文件加密两种。

① 通信加密

通信加密是对通信过程中传输的数据进行加密,它分为以下 3 种。

(a) 结点加密。在相邻结点之间对传输的数据进行加密。其原理是,在数据传输整个过程中,发送结点对要发送数据的明文进行加密,然后以密文发出,中间转发结点先接收密文并解密成明文,然后按照本结点的密钥进行加密,并以密文发出,直到达到数据的目标结点。

(b) 链路加密。在通信链路上对传输的数据进行加密,这种加密方法主要通过硬件来实现。其加密原理是明文每次从某个发送结点发出,经过通信站时,利用通信站进行加密形成密文,然后再进入通信链路传输;当密文经通信链路传输到某个相邻中继结点或目标结点时,先经过通信站对密文进行解密,然后结点接收明文。

(c) 端对端加密。这种加密方式的加密是在报文传输初始结点上实现,在数据传输整个过程中,报文都是以密文方式传输,直到报文到达目标结点才进行解密。

② 文件加密

文件加密是对存储的文件进行加密。文件加密分为单级加密和多级加密两种,在控制上一方面与用户或用户组相关,另一方面与数据有关。单级数据信息加密是对需要进行保密的数据信息一视同仁,不对这些数据信息进行保密级别分类的保密方式。多级数据信息加密是对需要保密的数据信息按其重要程度分成若干个保密等级的方式。这两种方式中多

级加密方法的实现较为困难。

(2) 安全协议设计

设计出安全的计算机网络是人们急切盼望的。因此,计算机网络安全协议的设计是目前计算机安全研究的主要问题。但不幸的是,网络安全是不可判断的,目前在安全协议的设计方面,主要是针对具体的攻击设计安全的通信协议。但如何保证所设计出的协议是安全的?保证协议的安全性通常有两种方法,其一是用形式化的方法来证明,另一种是用经验来分析协议的安全性。形式化证明的方法是人们所希望的,但一般意义上的协议安全性也是不可判定的,只能针对某种特定类型的攻击来讨论其安全性。对复杂的通信安全性,形式化证明比较困难,所以主要采用找漏洞的分析方法。

(3) 存取控制

存取控制也叫访问控制,必须对接入网络的权限加以控制,并规定每个用户的接入权限。由于网络是一个非常复杂的系统,其存取控制比操作系统的存取控制机制要复杂得多,尤其在高安全性级的多级安全性情况下更是如此。

5.5.3 防火墙技术

随着 Internet 的飞速发展,越来越多的企业通过 Internet 上的信息服务拓宽业务,为客户提供在线服务、技术支持。随之而来的是网络安全问题,因为 TCP/IP 很少考虑网络安全性,在 Internet 环境下的主机可能受到来自 Internet 上的任何一个地方的攻击,一个薄弱的环节被攻破,将会殃及其他主机。为了保护整个网络的安全,出现了防火墙技术。

内部网(Intranet)是近年来发展非常快的一种企业内部网络。内部网通常采用一定的安全措施与企业外部的 Internet 用户相隔离,这种安全措施就是防火墙。

防火墙就是一种由软、硬件构成的系统,用来在两个网络之间实施存取控制策略。注意:这里的存取控制策略是由使用防火墙的单位制定的。这种安全策略应当最适合本单位的需要。防火墙的功能有两个:一个是阻止,另一个是允许。阻止就是阻止某种类型的通信通过防火墙(从外部网络到内部网络,或者是相反)。允许的功能与阻止恰好相反。但大多数防火墙的主要功能是阻止。

但是绝对地阻止不希望的信息是很难做到的,而简单地购买一个商用防火墙往往不能得到所需要的保护。不过,正确使用防火墙则可以将风险降低到一个可以接受的水平。

本章小结

本章主要讲解计算机网络的概念,计算机网络是计算机技术与通信技术相结合的产物,是互连起来的计算机的集合。在计算机网络中的计算机是独立自治的。连接是物理的,由

硬件实现,连接的介质有:双绞线、同轴电缆、光纤电缆、微波等。

计算机网络的发展经历了面向终端的计算机网络、分组交换网络、计算机网络体系结构和高速网络技术4个时代。

计算机网络由主机和通信子网两部分组成。主机(host)是组成计算机网络的独立自主的计算机系统,用于运行用户程序和网络管理程序。通信子网(Communication Subnet),是将入网主机连接起来的实体,由接口报文处理机和传输介质组成。子网的任务是在入网主机之间传递信息,以提供通信服务。

计算机网络按分布范围可分为广域网、局域网、城域网和互联网。局域网应用广泛,也是互联网的基础网络。Internet是全球性的、最成功的当今世界上规模最大的计算机网络。Internet的广泛应用加速和推动了人类社会的发展和进步。

计算机网络协议是计算机通信的规则的集合,是计算机网络不可缺少的、极其重要的组成部分。组成计算机网络协议的三要素是语法、语义和同步。本章简要地介绍了ISO的OSI和TCP/IP协议。

数据交换技术包括线路交换、报文交换和分组交换。虚电路和数据报是分组交换的两种形式。面向连接的分组交换技术称为虚电路,而分组交换称为数据报。

在本章中还重点介绍了网络互连和Internet的概念、应用和网络安全的概念。

习 题 五

1. 什么是计算机网络?
2. 计算机网络的发展分哪几个阶段?
3. 计算机网络由哪两大部分组成?
4. 按网络的分布范围进行分类,计算机网络有哪几种?
5. 什么是网络协议?
6. 网络协议的三要素是什么?
7. 什么是分组交换?
8. 什么是网络的体系结构?
9. 局域网硬件由哪些基本组成部分?
10. 什么是IP协议?
11. 什么是TCP协议?
12. 什么是UDP协议?
13. 试给出IP地址的结构。
14. 有IP地址:218.199.17.56。试指出该IP地址是哪类IP地址,网络号是什么,主机号是多少?
15. 什么是地址解析?

16. 什么是域名？试给出您上网使用的域名。
17. 计算机网络面临的安全威胁来自哪些方面？
18. 计算机安全的内容包括哪些方面？
19. 什么是密码技术？
20. 什么是防火墙？

第六章　实验与指导

实验一　线性表的应用

一、目的和要求

1. 熟悉并掌握线性表的逻辑结构定义、特点。

2. 熟悉并掌握顺序表、单链表、单循环链表、双向链表的描述方法。

3. 熟悉并掌握顺序表、单链表的基本操作，包括表的建立与销毁、判别表是否空、求表长、插入与删除数据元素、查找数据元素等。

二、实验内容

编写下列程序，并上机调试运行。

1. 建立一个顺序表 L，然后删除表中所有值大于 $k1$ 且小于 $k2$ 的数据元素（$k1 \leqslant k2$）。

2. 建立一个顺序表 $L=(a_1,a_2,\cdots,a_n)$，利用原表的存储空间将顺序表$(a_1,a_2,\cdots,a_n)$逆置为$(a_n,a_{n-1},\cdots,a_1)$。

3. 输入一系列整数，以 0 为结束标志，以后进先出（先进后出）的次序建立一个带表头结点的单链表。

4. 访问单链表，依次输出表中的每个结点。

5. 输入一个整数，若单链表中存在一个含此整数的结点，则将此结点从表中删除，然后执行操作 4，若不存在这样的结点，则输出相应的信息。

三、实验环境

1. 硬件：PC 机。

2. 软件：DOS 3.30 以上版本，Turbo C 或 Borland C ++ 任意版本。

实验二　栈、队列

一、目的和要求

1. 熟悉并掌握栈、队列的逻辑结构定义、特点。

2. 熟悉并掌握栈、队列的顺序与链接两种存储方式。

3. 熟悉并掌握栈、队列的基本操作：栈、队列的建立、访问栈顶（队头）元素、进栈、出栈、入队、出队等。

二、实验内容

编写下列程序，并上机调试运行。

1. 判定一个表达式中开、闭括号是否配对。

2. 输入一个十进制整数，将其转换为八进制整数，然后输出。

3. 假设以带表头结点的循环链表表示队列，并且只设一个指针指向队尾元素结点（不设头指针）。试编写相应的队列初始化、入队、出队的算法。

三、实验环境

1. 硬件：PC 机。

2. 软件：DOS 3.30 以上版本，Turbo C 或 Borland C ++ 任意版本。

实验三　排　　序

一、目的和要求

熟悉并掌握教材中所介绍的几种排序方法。

二、实验内容

若输入序列为（16，12，24，16，4，8，20，2，30，18），编制程序，按下列排序方法对序列从小到大排序并输出。

1. 冒泡排序。

2. 插入排序。

3. 选择排序。

4. 快速排序。

5. 归并排序。

三、实验环境

1. 硬件：PC 机。

2. 软件：DOS 3.30 以上版本，Turbo C 或 Borland C ++ 任意版本。

实验四　查　　找

一、目的和要求

熟悉并掌握教材中所介绍的几种查找方法。

二、实验内容

1. 顺序查找序列(16,12,24,16,4,8,20,2,30,18)中值为12、45的元素,若查找成功,输出元素所在的位置,否则输出0。

2. 对序列(16,12,24,16,4,8,20,2,30,18)排序后,折半查找值为12、45的元素,若查找成功,输出元素所在的位置,否则输出0。

三、实验环境

1. 硬件:PC机。

2. 软件:DOS 3.30以上版本,Turbo C或Borland C++任意版本。

实验五 Windows系统的配置和用户管理

一、目的和要求

通过Windows系统提供的控制面板和系统工具的支持,学会定制自己所需的Windows的配置;掌握进行用户管理的方法,了解常用的系统工具的使用方法。

二、实验内容

1. 通过"控制面板"设置日期和时间的功能改变系统日期和时间的设置。

2. 通过"控制面板"提供的输入法功能,完成添加和删除一种输入法。

3. 通过"控制面板"提供的用户账户管理功能,完成:

(1) 创建、删除用户账户。

(2) 对已创建的用户账户设定密码。

(3) 更改已设定的密码。

4. 通过"控制面板"提供的添加/删除程序功能,完成某一应用程序的安装或删除(如Realplayer)。

5. 通过"附件"→"系统工具"→"磁盘碎片整理程序",完成自己的数据磁盘的整理工作。

三、实验环境

1. 硬件:PC机。

2. 软件:操作系统Windows 98或Windows XP。

四、实验指导

当完成了Windows安装后,安装程序会为用户提供一套默认的标准设置,如桌面、鼠标、键盘、"开始"菜单、任务栏等。对于一个用户而言,这一默认的设置未必适合自己。所以,用户可以根据自己的需要和爱好,借助Windows系统提供的"控制面板"、"系统工具"等功

能定义自己所在的 Windows 环境的各种属性。

由于 Windows 98 或 Windows XP 的桌面和使用方法有些区别，所以做如下说明。

1. 完成实验内容 2 中调用输入法功能时：

（1）在 Windows 98 中通过“控制面板”→“键盘”进入“键盘属性”对话框，选择“语言”即可完成添加和删除输入法的工作。也可通过“控制面板”→“输入法”直接进入“输入法”的添加和删除。

（2）在 Windows XP 中通过输入方法栏目中的“选项”→“设置”进入，即可完成添加和删除输入法的工作。

2. 实验 3 的内容是对用户账户进行管理。首先，对用户账户作有关的介绍。

Windows 2000/XP 是多用户操作系统，提供多用户运行环境，即使是 Windows 98 也允许将计算机设置为多人使用。多用户账户管理的基础是创建用户账户。创建用户账户可以赋予访问本地计算机的其他使用者一定的使用权限，如登录到计算机、新建文件或文件夹，但同时又不影响本机的使用安全。在创建用户账户时，要以“计算机管理员”的权限身份登录到 Windows 2000 / XP。在安装好 Windows 系统后，会要求用户设置计算机的使用账户，这一账户应为管理员账户。计算机管理员账户可以创建、更改和删除用户账户。

Windows XP 提供的用户账户功能比 Windows 98 强，并且使用更为方便。

（1）在 Windows 98 中通过“控制面板”→“用户”进入，进行后可看到如下显示信息：

Windows 系统允许将计算机设置为多人使用，……。

通过单击“下一步”按钮进行操作。

（2）在 Windows XP 中通过“控制面板”→“用户账户”进入，进行后可看到如下所显示的信息，通过选择可完成实验 3 的任务。

挑选一项任务

- 更改账户
- 创建一个新账户
- 更改用户登录或注销的方式

或挑一个账户做更改

实验六　绘制进程状态变迁图

一、目的和要求

掌握 Windows 环境下 Word 提供的绘图功能和 Windows 附件中的画图功能。

二、实验内容

1. 通过“附件”中“画图”程序绘制一个具有进程3个基本状态的进程状态变迁图。

2. 通过Word中“画图”工具提供的绘图功能绘制一个具有进程3个基本状态的进程状态变迁图。

进程状态变迁图中应具有进程的3个基本状态,每个状态以一个椭圆描绘,并区分不同的颜色,椭圆内用文字标明状态。状态之间的变化用带箭头的线表示。

三、实验环境

1. 硬件:PC机。

2. 软件:操作系统Windows 98或Windows XP;文字处理软件Word。

实验七 Linux系统的用户界面——基本操作命令

一、目的和要求

了解Linux系统的使用环境,掌握Linux系统的登录、退出和关机的方法;以及Linux系统的操作界面——基本键盘命令的使用方法。初步了解Linux系统用户账户的管理方法。

二、实验内容

1. 以root用户账户及口令登录进入Linux系统。

2. 通过Linux的用户管理工具:useradd、usermod、userdel完成以下工作:

(1) 用useradd命令增加一个新用户。

(2) 对已增加的用户设置口令。

(3) 用userdel命令删除新增的用户。

3. 用vi命令编辑一个文本文件test,存放到用户起始目录下。

4. 使用目录与文件的基本操作命令,完成以下工作:

(1) 用pwd命令显示当前目录。

(2) 在用户起始目录下建立一个sub子目录。

(3) 显示test文件的内容。

(4) 将test文件从用户起始目录移至sub子目录。

(5) 用ls命令列出用户起始目录和sub子目录的内容。

三、实验环境

1. 硬件:PC机。Linux支持的CPU十分广泛,从最老的386到最新的Pentium系列。对于Red Hat Linux而言,可运行的最小主存是8 MB。

2. 软件:Red Hat Linux。

实验八　Linux 内核代码结构与系统状态

一、目的和要求

通过 Linux 系统的基本操作命令，了解 Linux 系统的内核代码结构。了解 Linux 核心源代码所在的位置，并能显示核心文件的内容。掌握查看多用户、多任务信息的方法。

二、实验内容

1. 使用 pwd、ls、cd 命令查看 Linux 的目录树。

2. 找到 Linux 源代码所在的位置，用 ls 命令列出此目录下的所有源文件，找到进程调度程序所在的文件 sched. c。

3. 用 cat 命令和 more 命令看 sched. c 文件的内容，了解这两个命令的区别。

4. 使用 who 命令查看多用户信息。

5. 使用 ps 命令查看多任务信息。

三、实验环境

1. 硬件：PC 机。Linux 支持的 CPU 十分广泛，从最老的 386 到最新的 Pentium 系列。对于 Red Hat Linux 而言，可运行的最小主存是 8 MB。

2. 软件：Red Hat Linux。

实验九　需求分析与概念设计

数据库实验背景概述

某大型企业在全国各地有多个生产工厂，每个工厂有车间、产品和职工 3 种主要数据，其中：

① 一个车间有多个职工，但一个职工仅属于一个车间。

② 一个车间可以生产多种产品，一种产品也可在多个车间中生产。

③ 一个职工可负责加工多种产品，一种产品也可由多个职工负责加工。

为了更方便地实现全局的任务分配和调度，该企业欲建立一个产品信息数据库。现要求数据库能满足下列事务处理需求：

(1) 找出某种产品的生产地。

(2) 找出生产某种产品的所有车间。

(3) 列出某个职工负责生产的所有产品。

(4) 列出某工厂所有职工的详细信息。

此外,还应能对数据库进行正常维护,即各种增、删、改操作,并保持其完整性。

一、目的和要求

1. 熟悉并掌握进行需求分析的方法。

2. 熟悉并掌握以 E-R 图的形式表示概念数据库结构的方法。

二、实验内容

1. 分析应用环境,确定系统需要存储的有关数据的各种信息,包括静态的数据定义以及描述信息,和动态的关于数据的使用和操作的信息。从而得到系统的实体和实体之间的联系。

2. 画出局部 E-R 图。

3. 画出全局 E-R 图。

三、实验环境

1. 硬件:PC 机。

2. 软件:Microsoft Word 或任意版本的文档图形编辑软件。

四、实验指导

1. 本实验重点在掌握 E-R 图的画法。可参考 4.4.4 节所述理论以及 4.4.9 节实例。

2. 画 E-R 图首先应该确定实体、属性以及标识实体的码,然后确定实体之间的联系及其类型。

3. 合并局部 E-R 图成为全局 E-R 图时,要注意解决 3 类冲突。

实验十 数据库定义

数据库实验背景概述同实验九。

一、目的和要求

1. 熟悉并掌握概念模型向关系模型转换的方法。

2. 熟悉并掌握用 SQL 语言创建表格、插入数据等相关语法。

二、实验内容

1. 逻辑转换。将 E-R 图转换成一般的关系数据库模式,给出设计的各关系模式,并用下划线标识码。

2. 数据库定义。根据目标 DBMS 所提供的数据库定义语言和描述规则,创建相应的基本表。

3. 数据装载。它包括下列任务:

(1) 初始数据收集与整理。

（2）初始数据装入程序的设计。

（3）装入初始数据到数据库并输出核对。

三、实验环境

1. 硬件：PC 机。

2. 软件：Microsoft Word，Microsoft SQL Server 2000 或其他以 SQL 语言为核心的数据库管理系统。

四、实验指导

1. E－R 图向关系模式转换，可将实体转换成关系，而实体间的联系则区分为几种不同的类型进行转换，可参考 4.4.5 节所述内容以及 4.4.9 节实例部分。

2. 用 CREATE TABLE 命令创建基本表，用 INSERT 命令插入数据。

3. 请为每个基本表设计至少 3 条记录的数据。

实验十一　数据库操作

数据库实验背景概述同实验九。

一、目的和要求

1. 熟悉并掌握使用 SQL 语言进行增加、删除、修改、查询等各项操作。

2. 熟悉并掌握使用高级程序设计语言连接数据库并进行操作的方法（选作）。

二、实验内容

1. 该企业新建了一座工厂，用户应根据实际的语义写出相应的 SQL 语言，将该工厂的相关信息插入数据库。

2. 该企业决定以后不再生产某种产品，并解雇只负责生产这种产品的职工，用户应根据实际的语义写出相应的 SQL 语言，从数据库中删除该产品以及负责生产这种产品的职工的相关信息。

3. 该企业决定交换两种产品的生产负责职工，即原来负责生产 A 产品的转到负责生产 B 产品，而以前负责生产 B 产品的现在负责生产 A 产品，用户应根据实际的语义写出相应的 SQL 语言，修改数据库中的相关信息。

4. 查询该企业 50 岁以上女性职工负责生产的所有产品。

三、实验环境

1. 硬件：PC 机。

2. 软件：Microsoft SQL Server 2000 或其他以 SQL 语言为核心的数据库管理系统，C 或者 Java 等高级语言编译运行环境。

四、实验指导

1. 实验中的不定数据,例如“某种产品”,用户可根据“实验十”插入的记录进行相应的选择。

2. 实验内容3中,可用当前数据库中并不存在的产品C作为过渡。可先将生产A修改成生产C,然后将生产B修改为生产A,最后再将生产C修改为生产B。思考一下,为什么要这么做?

3. 在C语言或者其他高级语言(如Java)上,要用ODBC或者JDBC连接数据库完成相应操作,用ODBC或者JDBC连接数据库的方法可查阅相关参考书。

实验十二 数据库维护

数据库实验背景概述同实验九。

一、目的和要求

熟悉并掌握使数据库正确、高效运行的维护方法。

二、实验内容

1. 数据库转储(备份)。周期性的把数据库转储到其他的存储介质上。

2. 删除。删除过期数据,以节约磁盘空间。

3. 恢复。当现有数据库发生故障时,恢复到前一个备份的状态。

三、实验环境

1. 硬件:PC机。

2. 软件:Microsoft SQL Server 2000或其他以SQL语言为核心的数据库管理系统。

四、实验指导

1. 可参考相关数据库产品的用户手册,查找相关的命令或者操作步骤,如SQL Server 2000中的BACKUP DATABASE命令。

2. 相应的还有RESTORE DATABASE命令。

实验十三 网络配置

一、目的和要求

1. 熟悉并掌握网络环境。

2. 熟悉并掌握局域网环境的网络配置。

3. 熟悉并掌握拨号上网配置。

二、实验内容

1. 微机网络配置。

2. Windows 环境下局域网的网络配置。

3. Windows 环境下拨号上网配置。

三、实验环境

1. 硬件:PC 机、网卡、Modem。

2. 软件:Windows 2000/XP Me 9x,网卡驱动程序和 Modem 驱动程序。

四、实验指导

1. 在微机上安装 Windows 2000(或 XP、Me)操作系统。

2. 局域网络环境:配置网卡,并安装网卡驱动程序(随购置网卡附带的网卡驱动程序)。

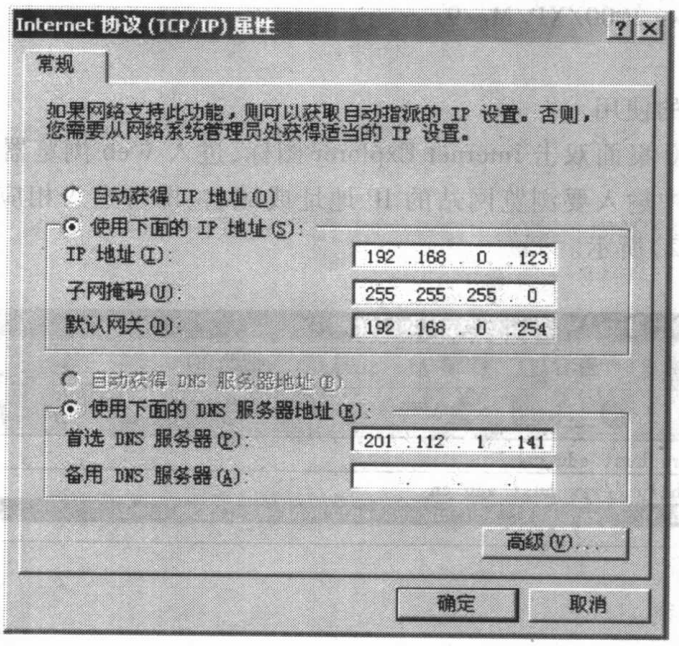
3. 配置局域网环境下的网络步骤如下:

① 在 Windows 2000 的桌面右击“网上邻居”,单击“属性”。这时桌面上出现“网络拨号与连接”对话框,单击“本地连接”。

② 在本地连接状态框中单击“属性”,桌面上出现本地连接属性对话框。

③ 在本地连接属性框中单击“Internet 协议(TCP/IP)”选项。

④ 在 Internet 协议(TCP/IP)属性对话框中输入如图(实验十三 图 1)所示的内容,包括 IP 地址、子网掩码、网关地址、DNS 地址。然后单击“确定”即可完成局域网络环境下的网络



实验十三 图 1　配置网络

配置。注意:该配置适应于局域网络环境,也适合家庭宽带网络环境。在 Windows 9x/Me 环境下将重新启动系统。

4. 拨号上网按配置的 Modem 的讲明书和相应的服务网站提供的说明书配置上网环境。

实验十四　网 络 应 用

一、目的和要求

1. 熟悉并掌握 Web 浏览器的使用。
2. 熟悉并掌握文件传输系统 FTP 的使用。
3. 熟悉并掌握电子邮件的使用。

二、实验内容

1. Web 浏览器的使用。
2. 文件传输系统 FTP 的使用。
3. 电子邮件的使用。

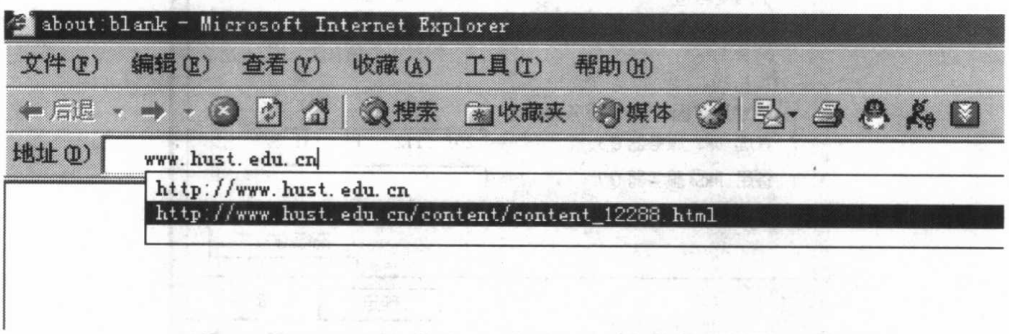
三、实验环境

1. 硬件:PC 机、网卡(或 Modem)。
2. 软件:Windows 2000/XP/Me/9x。

四、实验指导

1. Web 浏览器的使用

在 Windows 2000 桌面双击 Internet Explorer 图标,进入 Web 浏览器,如图(实验十四 图 1)所示。在“地址”中输入要浏览网站的 IP 地址或域名,即可进入相应网站的浏览器主页,如图(实验十四 图 2)所示。

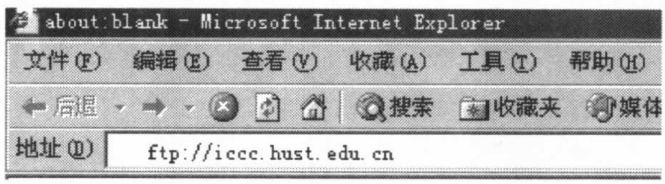


实验十四 图 1　Internet Explorer

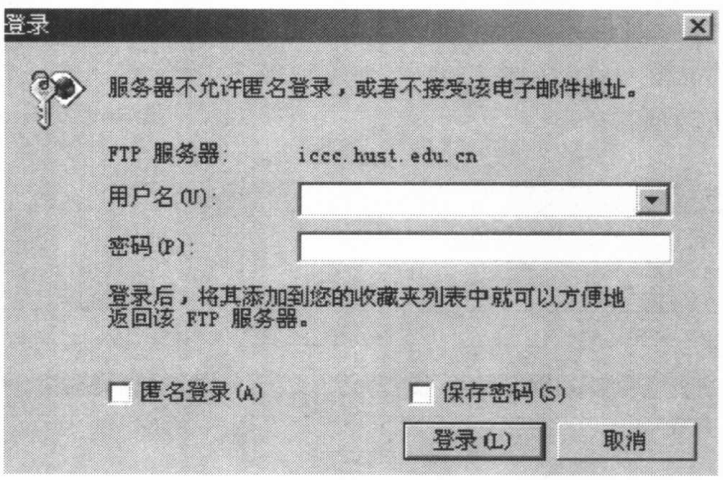
实验十四 图2　华中科技大学主页

2. FTP 应用

首先在 Windows 2000 桌面双击 Internet Explorer 图标，进入 Web 浏览器，如图(实验十四图 3 (a))所示，然后输入 FTP://文件传输服务器的 IP 地址或域名，即可进入相应的文件传输服务器的“登录”对话框，如图(实验十四图 3 (b))所示。



(a)



(b)

实验十四 图3　FTP 应用

在登录对话框中输入相应的用户名和密码后就可以进入相应的文件服务器。在文件服务器中可实现文件的上传和下载。也可以匿名的方式进入相应的文件传输服务器，只要选择图(实验十四图 3(b))中的“匿名登录”，就可以进入相应的文件传输服务器。

3. 电子邮件的使用

电子邮件的使用有多种方式。一类是各种电子邮件应用程序，如 Outlook Express。另一类是 Internet Explorer 浏览器方式。

参考文献

1 庞丽萍,印旻.软件开发基础——面向对象技术与操作系统虚拟机.北京:高等教育出版社,1999

2 董荣胜,古天龙.计算机科学与技术方法论.北京:人民邮电出版社,2002

3 严蔚敏,吴伟民.数据结构(C语言版).北京:清华大学出版社,1997

4 徐孝凯.数据结构实用教程.北京:清华大学出版社,1999

5 庞丽萍.操作系统原理.第3版.武汉:华中理工大学出版社,2000

6 尤晋元,史美林等.Windows操作系统原理.北京:机械工业出版社,2001

7 刘云生,卢正鼎,卢炎生.数据库系统概论.第2版.武汉:华中理工大学出版社,1997

8 冯玉才.数据库系统基础.第2版.武汉:华中理工大学出版社,1993

9 萨师煊,王珊.数据库系统概论.第3版.北京:高等教育出版社,2000

10 Abraham Silberschatz, Henry F. Korth, S. Sudarshan. Database System Concepts. 北京:机械工业出版社,2000

11 [美]M. F. Arnett, M. coulombe. 万玉丹,忻宏杰,徐旭东译. TCP/IP实用技术指南.北京:清华大学出版社,1997

12 尚晓航,陈强.计算机局域网与Windows NT实用教程.北京:清华大学出版社,1999

{
  "filename": "MTE4NDI3Mzguemlw",
  "filename_decoded": "11842738.zip",
  "filesize": 33659576,
  "md5": "d4db5b55e8d3006ff5a56fbe2fe250a0",
  "header_md5": "4b3e05da2a8ee3065c9c9bd6359fbaf6",
  "sha1": "82be2449579a591d8496d2bd3f8aed8f26fd1246",
  "sha256": "1a3712b884f542cadfd4831c2deacc33f4a548d0d35fef721495bff602997463",
  "crc32": 2855751821,
  "zip_password": "",
  "uncompressed_size": 35255082,
  "pdg_dir_name": "\u00ed\u2562\u255e\u2552\u2550\u00bf\u2555\u2580\u2561\u255a\u255c\u2560\u2559\u00b2\u00ed\u2591\u2569\u00ab\u256c\u03c3\u00ed\u2592\u2563\u00b7\u255d\u2565\u255d\u2562\u2563\u00b5\u2557\u00ab\u255c\u2560\u2593\u2500\u255d\u255e\u2566\u03c0\u2557\u00b7\u255a\u03c6\u255d\u25a0\u255d\u255d\u2569\u2321\u2561\u255d\u252c\u2588\u00ed\u2556_11842738",
  "pdg_main_pages_found": 361,
  "pdg_main_pages_max": 361,
  "total_pages": 373,
  "total_pixels": 2213793792,
  "pdf_generation_missing_pages": false
}