

湖南省计算机学会规划教材
高等院校计算机类系列教材

计算机操作系统

主编 胡志刚 谭长庚
副主编 朱艳辉 陈利平

中南大学出版社

总摇序

21世纪,人类社会已经步入信息时代,信息产业推动着全球经济的蓬勃发展,改变着人类的联系与交换方式,从某种意义上说,信息革命是人类历史上又一次深刻的社会变革。无疑,在以信息产业为基础的知识经济社会中,计算机科学与技术具有举足轻重的地位。有鉴于此,当今世界各国皆把培养高素质的创新型计算机科学与技术专业人才作为一项重要的战略任务来抓。早在 1982年,邓小平同志就强调指出:“计算机的普及要从娃娃抓起”,从此开启了中国信息革命的征程。经过 30多年的努力,我国的计算机教育虽然取得了令人瞩目的成就,但离知识经济社会的要求还有很大的差距。据 2009年信息产业部的数据显示,我国的信息化人才资源指数仅为 0.40,每年短缺信息化专业人才达 150万之多。因此,快速培养和造就一大批高素质的计算机与信息人才,乃是我国高等教育所面临的一项严峻挑战。为此,我们必须改革和完善现有计算机与信息技术学科的教学计划和课程体系,优化课程结构,精炼教学内容,拓宽专业基础,强化实践环节,注重学生的知识、能力和综合素质的培养。

为了适应计算机科学与技术学科发展和教育的需要,湖南省计算机学会,参照《中国计算机科学与技术学科教程 2006》,组织了一批长期从事计算机科学与技术专业教学与科研的学者参与编撰了这套由中南大学出版社出版的《高等院校计算机系列教材》,希望在教材中及时反映学科前沿的研究成果与发展趋势,以高水平的科研促进教材建设,以优秀教材促进教学质量的提高。该系列教材具有如下特点:

1. 教材参照《中国计算机科学与技术学科教程 2006》建议的教学大纲、知识领域、知识单元和知识点,结合作者多年教学与科研经验来编写,注重基本理论、基础知识的梳理、推演与挖掘,注意知识的更新,跟踪新技术、新成果的发展,并将之吸收到教材中来,力求开阔学生视野,逐步形成“基础课程精深,专业课程宽新”的格局,努力提高教材质量。

2. 注重理论联系实际,注意能力培养。力图通过案例教学、课堂讨论、课程实验设计与实习,训练学生掌握知识、运用知识分析并解决实际问题的能力以满足学生今后从事科研和就业的需要。

3. 在规范教材编写体例的同时,注重写作风格的灵活性:每册的每个章节包括教学目的、本章小结、思考题与练习题,每门教材都配有 300页电子教案,并做到层次分明、逻辑性强、概念清楚、图文并茂、表达准确、可读性强。

这套教材的编写吸纳了广大计算机科学与技术教育工作者多年的教学与科研成果，凝聚了作者们的辛勤劳动，也得到了湖南省各高等院校相关专业领导和专家的大力支持。我相信这套教材的出版，对我国计算机科学与技术专业本科教学质量的提高将有很好的促进作用。

由于编委和作者们水平与时间的限制，教材中难免还有不足之处，恳请广大读者批评指正。

A handwritten signature in black ink, reading '陈海明' (Chen Haiming). The characters are written in a fluid, cursive style.

國曆年 苑月

前 言

操作系统是现代计算机系统的重要组成部分，它作为重要的系统软件，在整个计算机系统中占有中心地位。操作系统的主要作用就是控制和管理计算机硬件和软件资源、合理地组织计算机工作流程以及方便用户使用计算机系统。随着计算机的发展，计算机系统的硬件和软件资源也越来越丰富。为了提高这些资源的利用率，增强系统的处理能力和改善用户与计算机间的接口，需要功能更完善的操作系统，从而出现了满足不同要求的操作系统，操作系统的功能也越来越强。

本教材以国际知名、使用广泛的 哉驛戴操作系统为线索，阐述了操作系统的基本概念、工作原理和实施技术。本书在编写过程中，注意到了理论和实践的结合以及培养学生分析和设计操作系统的能力。在讲述操作系统的一般概念和工作原理时，尽可能地通过图表进行说明，以方便教师讲解和学生理解。

本书的内容安排如下：第 员章介绍操作系统的功能、发展、分类等基本概念。第 圆章对操作系统中的一个重要概念——进程做了介绍，包括进程状态及转换、进程控制、进程通信、线程等。第 猿章对调度的基本原理、调度算法、算法性能的评估等作了详细的介绍，此外，还对 哉驛戴的调度算法进行了分析。第 源章介绍了关于进程通信的知识，包括进程互斥与同步的概念、临界资源与临界区的概念、信号量机制、经典同步问题与管程的概念，最后对 哉驛戴系统的信号量机制进行了分析。第 缘章首先介绍了死锁的基本概念，然后分别对产生死锁的必要条件、避免和预防死锁的机制做了介绍。第 远章和第 苑章介绍了操作系统的存储管理。其中第 远章主要介绍了操作系统中有关存储管理的基本概念，几种常用存储管理技术的基本思想、实现算法及硬件支持。第 苑章主要是讨论了虚存管理技术和辅助存储器管理。第 愿章是关于设备管理的，主要介绍了 隅酌设备、中断技术、缓存技术等。第 怨章对文件系统中的文件管理、文件的共享与保护做了介绍，此外，作为实例还对 哉驛戴文件系统的主要结构和算法作了说明。在本书的最后部分对另外两个与操作系统有关的知识做了介绍，即操作系统的安全和设计。第 员园章主要是在介绍计算机信息安全基础知识的基础上，分析了操作系统产生的原因，讨论了操作系统的保护和安全机制，并对 哉驛戴操作系统的安全做了简要的介绍。第 员员章在介绍计算机操作系统结构的基础上介绍了设计操作系统的原理及方法等。

本书由胡志刚教授负责总体框架、制订写作大纲及最后定稿。其中第 员 源章（源员~源圆 源远 源愿 由谭长庚编写，第 圆章及 源缘 源苑由朱艳辉编写，第 猿章由陈强编写，第 缘章由阳慧编写，第 远 苑章由宋虹编写，第 愿章由左贵后编写，第 怨章由陈利平编写，第 员园章由王楚正编写，第 员员章由何啸峰编写，附录部分由李玺编写。

本书可作为高等院校计算机科学与软件工程类专业的教材，也可作为对计算机有一定了解的科技工作者及计算机使用者自学的参考书。在此，向为本书编写给予指导、提出宝贵意见的老师 and 同行们表示衷心的感谢。书中难免有些疏漏谬误之处，恳请读者批评指正。

编摇者

圆园缘年 远月

目 录

第 1 章 绪 论	1
1.1 操作系统的设置目的	1
1.2 操作系统的定义和设计目标	1
1.3 操作系统的作用	1
1.4 操作系统的功能	1
1.5 处理器管理	1
1.6 存储管理	1
1.7 设备管理	1
1.8 信息管理	1
1.9 用户接口	1
1.10 操作系统的基本类型	1
1.10.1 批处理操作系统	1
1.10.2 分时操作系统	1
1.10.3 实时操作系统	1
1.10.4 其他操作系统	1
1.11 操作系统的特征与性能	1
1.11.1 操作系统的特征	1
1.11.2 操作系统的性能	1
1.12 操作系统特征	1
第 2 章 进 程	2
2.1 进程概念	2
2.2 进程的引入	2
2.3 进程的定义与特征	2
2.4 进程描述	2
2.5 进程状态及转换	2
2.5.1 三态模型	2
2.5.2 五态模型	2
2.6 具有挂起功能的进程状态及其转换	2
2.7 进程控制	2
2.7.1 进程的创建	2
2.7.2 进程的撤销	2
2.7.3 进程的阻塞与唤醒	2

进程的挂起与激活	猿
进程切换与模式切换	猿
进程执行模式	猿
进程模式切换	猿
进程切换	猿
线程	猿
线程的概念	猿
线程与进程的比较	猿
线程的实现	猿
线程结构	猿
线程映象	猿
线程状态及变迁	猿

第 猿章 调度

调度策略	猿
作业调度	猿
进程调度	猿
调度算法	猿
调度算法评价	猿
调度客观型调度算法	猿
调度主观型调度算法	猿
调度与进程调度与换进换出	猿
调度进程调度	猿
调度的换进换出	猿

第 源章 进程同步与通信

临界区	猿
互斥与同步	猿
临界资源	猿
临界区	猿
同步机构的设计原则	猿
解决互斥问题的软件方法	猿
利用硬件方法解决进程互斥问题	猿
信号量	猿
信号量概念	猿
信号量的应用	猿
信号量集机制	猿
经典进程同步问题	猿
生产者—消费者问题	猿

源猿猿猿猿读者—写者问题	(猿圆)
源猿猿猿猿哲学家进餐问题	(猿猿)
源猿猿猿猿管程	(猿源)
源猿猿猿猿信号量机制存在的问题	(猿源)
源猿猿猿猿管程概念	(猿源)
源猿猿猿猿管程的实现	(猿缘)
源猿猿猿猿进程通信	(猿缘)
源猿猿猿猿进程通信的类型	(猿圆)
源猿猿猿猿共享存储器系统	(猿圆)
源猿猿猿猿管道通信系统	(猿圆)
源猿猿猿猿消息传递系统	(猿圆)
源猿猿猿猿信号量机构分析	(猿猿)
源猿猿猿猿进程通信机构	(猿猿)
第 缘章 猿死猿锁	(猿缘)
缘猿猿猿猿的概念	(猿缘)
猿猿猿猿猿的定义	(猿缘)
猿猿猿猿猿产生死锁的原因	(猿缘)
猿猿猿猿猿死锁举例	(猿圆)
缘猿猿猿猿资源分配模型	(猿猿)
缘猿猿猿猿死锁条件	(猿猿)
缘猿猿猿猿死锁预防	(猿猿)
猿猿猿猿猿破坏互斥条件	(猿猿)
猿猿猿猿猿破坏占有等待条件	(猿猿)
猿猿猿猿猿破坏非剥夺条件	(猿猿)
猿猿猿猿猿破坏循环等待条件	(猿猿)
缘猿猿猿猿死锁避免	(猿源)
猿猿猿猿猿系统的安全状态	(猿源)
猿猿猿猿猿银行家算法	(猿源)
缘猿猿猿猿死锁检测	(猿圆)
缘猿猿猿猿死锁恢复	(猿猿)
第 远章 猿存储管理	(猿源)
远猿猿猿猿存储管理功能	(猿源)
猿猿猿猿猿主存空间分配和管理	(猿源)
猿猿猿猿猿地址转换和重定位	(猿源)
猿猿猿猿猿存储保护和共享	(猿缘)
猿猿猿猿猿存储扩充	(猿缘)
远猿猿猿猿分区存储管理	(猿圆)

摇摇摇摇固定分区管理	(页码)
摇摇摇摇动态分区管理	(页码)
摇摇摇摇动态重定位分区管理	(页码)
摇摇覆盖与交换技术	(页码)
摇摇摇摇覆盖	(页码)
摇摇摇摇交换技术	(页码)
摇摇分页存储管理	(页码)
摇摇摇摇基本原理	(页码)
摇摇摇摇数据结构	(页码)
摇摇摇摇地址转换机制	(页码)
摇摇摇摇页面分配策略	(页码)
摇摇摇摇分页存储管理的共享与保护	(页码)
摇摇分段存储管理	(页码)
摇摇摇摇基本原理	(页码)
摇摇摇摇数据结构和地址转换机制	(页码)
摇摇摇摇段的共享与保护	(页码)
摇摇摇摇分页存储管理和分段存储管理的区别	(页码)
摇摇段页式存储管理	(页码)
第 苑章 摇摇虚存管理	(页码)
摇摇基本概念	(页码)
摇摇摇摇程序访问局部性原理	(页码)
摇摇摇摇虚存的定义	(页码)
摇摇摇摇虚存的特征	(页码)
摇摇摇摇虚存的实现方法	(页码)
摇摇请求分页存储管理	(页码)
摇摇摇摇数据结构	(页码)
摇摇摇摇地址转换	(页码)
摇摇摇摇页面调度策略	(页码)
摇摇摇摇性能分析	(页码)
摇摇摇摇请求分页管理的共享与保护	(页码)
摇摇请求分段存储管理	(页码)
摇摇摇摇段表机制	(页码)
摇摇摇摇地址转换和硬件支持	(页码)
摇摇摇摇戴存储管理	(页码)
摇摇摇摇交换	(页码)
摇摇摇摇请求调页管理	(页码)

第 愿章 设备管理	(愿苑)
愿愿 设备的组织	(愿苑)
愿愿 设备的设备	(愿苑)
愿愿 设备控制器	(愿苑)
愿愿 系统的结构	(愿苑)
愿愿 数据传送控制方式	(愿苑)
愿愿 程序直接控制方式	(愿苑)
愿愿 程序中断控制方式	(愿苑)
愿愿 异步控制方式	(愿苑)
愿愿 通道控制方式	(愿苑)
愿愿 中断技术	(愿苑)
愿愿 中断的基本概念	(愿苑)
愿愿 中断处理	(愿苑)
愿愿 缓冲技术	(愿苑)
愿愿 缓冲的引入	(愿苑)
愿愿 缓冲区的种类	(愿苑)
愿愿 缓冲池	(愿苑)
愿愿 设备分配	(愿苑)
愿愿 设备分配策略	(愿苑)
愿愿 设备分配程序	(愿苑)
愿愿 软件的层次结构	(愿苑)
愿愿 用户层的软件	(愿苑)
愿愿 与设备无关的软件	(愿苑)
愿愿 设备驱动程序	(愿苑)
愿愿 中断处理程序	(愿苑)
愿愿 磁盘管理	(愿苑)
愿愿 磁盘性能参数	(愿苑)
愿愿 磁盘调度策略	(愿苑)
愿愿 廉价磁盘冗余阵列	(愿苑)
愿愿 块设备管理	(愿苑)
愿愿 概述	(愿苑)
愿愿 块设备管理	(愿苑)
愿愿 块设备驱动程序	(愿苑)
愿愿 块设备读写技术	(愿苑)
愿愿 块设备的程序结构	(愿苑)
第 愿章 文件系统	(愿苑)
愿愿 目录结构	(愿苑)

一级目录结构	(页码)
二级目录结构	(页码)
树形目录结构	(页码)
无环图目录结构	(页码)
通用图目录结构	(页码)
文件结构和存取方法	(页码)
文件结构	(页码)
存取方法	(页码)
文件存储空间管理	(页码)
文件存储空间分配	(页码)
存储空间的管理	(页码)
文件的共享和保护	(页码)
文件的共享	(页码)
文件的保护	(页码)
文件的系统调用	(页码)
文件系统的基本模型	(页码)
文件系统	(页码)
文件系统概述	(页码)
文件系统的物理结构	(页码)
文件系统的数据结构及其关系	(页码)
空闲磁盘空间的管理	(页码)
文件的文件操作	(页码)

第 10 章 安全和保护

操作系统安全概述	(页码)
安全性概述	(页码)
操作系统安全的概念	(页码)
数据保密	(页码)
数据加密技术	(页码)
密码体制的分析	(页码)
保护	(页码)
存储保护	(页码)
访问控制	(页码)
保护	(页码)
安全机制	(页码)
认证机制	(页码)
授权机制	(页码)
审计机制	(页码)
的安全机制	(页码)

第 1 章 操作系统概述	1
1.1 操作系统的概念	1
1.2 操作系统的功能	2
1.3 操作系统的层次结构	3
1.4 操作系统的组成	4
1.5 操作系统的特性	5
1.6 操作系统的历史	6
1.7 操作系统的分类	7
1.8 操作系统的组成	8
1.9 操作系统的组成	9
1.10 操作系统的组成	10
第 2 章 操作系统结构与设计	11
2.1 操作系统结构	11
2.2 操作系统用户接口设计	12
2.3 操作系统命令接口	13
2.4 操作系统程序接口	14
2.5 操作系统图形用户接口	15
2.6 操作系统设计与实现	16
2.7 操作系统性能	17
附录 A 实验	18
实验一 处理机调度	18
实验二 主存储器空间的分配和回收	19
实验三 中断处理	20
实验四 资源分配	21
参考文献	22

第 1 章 绪论

操作系统是控制和管理计算机硬件和软件资源、合理地组织计算机工作流程以及方便用户使用计算机系统的程序集合。随着计算机的发展，计算机系统的硬件和软件资源也越来越丰富。为了提高这些资源的利用率，增强系统的处理能力和改善用户与计算机间的接口，操作系统得到了较大的发展，出现了满足不同要求的操作系统，其功能也越来越强。本章主要介绍操作系统的设置目的、功能、基本类型、基本特征与性能评价，以及操作系统的设置目的。

1.1 操作系统的设置目的

1.1.1 操作系统的定义和设计目标

1.1.1.1 操作系统的定义

操作系统（Operating System，简称 OS）是控制和管理计算机系统硬件和软件资源、合理地组织计算机工作流程以及方便用户使用计算机系统的程序集合。

计算机系统是由硬件系统和软件系统两部分组成的。没有配置软件的机器称为裸机，操作系统是在裸机上配置的第一层系统软件，在操作系统之上还可安装其他系统软件，如语言处理程序、编辑软件等。用户通过操作系统来使用计算机，操作系统在计算机用户和计算机之间起着中介作用。操作系统已成为计算机系统中最基本的、必不可少的系统软件。

为了给用户提供功能更完善和个性化的工作环境，除了安装操作系统外，根据用户的不同需求，还可以安装各种其他系统软件，这些系统软件在操作系统控制下工作。硬件、操作系统、其他系统软件与应用软件的层次关系如图 1-1 所示。

1.1.1.2 操作系统的设计目标

目前存在着多种类型的操作系统，不同类型的操作系统，其目标各有所不同。通常给计算机系统配置的操作系统，期望达到以下几个目标：

（1）方便性。操作系统为用户提供良好的、一致的用户接口，用户按需要输入命令，操作系统按命令去控制程序的执行；用户也可以在程序中调用操作系统的功能模块完成相应服务，而不必了解硬件的物理特性。

（2）有效性。有效地管理和分配硬件、软件资源，合理地组织计算机的工作流程，提高系统工作效率。操作系统扩充硬件的功能，使硬件的功能发挥得更好；操作系统使用户合理共享资源，防止各用户间的相互干扰；操作系统以文件形式管理软件资源，保证信息的安全和快速存取；操作系统合理地组织计算机的工作流程，使用户程序能顺利完成。

（3）可扩充性。为满足计算机硬件与体系结构的发展以及应用不断扩大的要求，操作系统应能方便地扩展新的功能。

1.1 操作系统是计算机系统资源的管理者

在一个计算机系统中，通常都有各种各样的硬件和软件资源，主要有以下四类：处理机、存储器、I/O 设备以及信息（数据和程序）。操作系统作为计算机系统资源的管理者，就是把上述四类资源作为管理对象，操作系统的主要功能也正是针对这四类资源进行有效的管理。操作系统管理的内容包括资源的当前状态、资源的分配、回收和访问操作、相应管理策略等。

1.2 操作系统扩充了机器功能

一台没有安装软件的计算机系统，即裸机，其功能再强，也是难以使用的。如果在裸机上安装了操作系统，用户便可利用它所提供的各种命令来操作计算机，以完成希望完成的任务。此时用户所看到的机器，将是一台比裸机功能更强、使用更方便的机器。操作系统是加在计算机硬件之上的第一层软件，它扩充了机器功能，还可以在其上增加系统软件与应用软件。每增加一层软件，就成为一个功能更强的机器，称这个新的功能更强的机器为虚拟机（Virtual Machine）。操作系统是计算机系统的核心，它管理着计算机系统的各种资源，并为用户提供各种服务。

1.3 操作系统合理组织工作流程

操作系统能合理地进行作业和进程管理，以充分利用计算机系统资源，提高系统运行效率。

1.4 操作系统的功能

操作系统是计算机系统资源的管理者，其主要任务是对系统中的硬件、软件资源实施有效的管理，以提高系统资源的利用率。操作系统的主要功能包括处理机管理、内存管理、设备管理、文件管理以及为方便用户而提供的友好的用户接口。

下面只对这些功能作简单介绍，使读者对这些功能有一个初步的了解，有关详细内容将在后面的章节中详细讨论。

1.4.1 处理机管理

处理机是计算机系统核心资源，对处理机管理的好坏直接关系到系统的整体性能。处理机管理完成对处理机的分配调度与运行管理等功能。在传统的操作系统中，处理机的分配调度是以进程（Process）为单位，因此处理机管理最终归为对进程的管理。现代操作系统中，都引入了线程（Thread），处理机管理还需包含对线程的管理。进程管理主要包括：

1.4.1.1 进程控制

在传统的多道程序环境下，要使作业运行，必须先为它创建一个或几个进程，并为之分配必要的资源。进程运行结束时，立即撤销该进程，以便能及时回收该进程所占用的各类资源。进程控制的主要功能是为作业创建与撤销进程，控制进程在运行过程中的状态转换（如阻塞与唤醒进程，挂起与激活进程）。

在引入线程的操作系统中，进程控制还应具有为一个进程创建若干个线程的功能和撤销已完成任务的线程的功能及其他控制功能。

1.4.1.2 进程同步

进程同步的主要任务是协调多个并发执行的进程（线程）之间的推进步骤。为使多个进程能有条不紊地并发执行，系统中必须设置进程同步机制。

进程同步有两种协调方式：

(员 进程互斥方式。这是指诸进程(线程)在对临界资源进行访问时,应采用互斥方式。

(圆 进程同步方式。指在相互合作完成共同任务的诸进程(线程)间,由同步机构对它们的执行次序加以协调。

为了实现进程同步,系统中必须设置进程同步机制。最简单的用于实现进程互斥的机制是为每一个临界资源配置一把锁,当锁打开时,进程(线程)可以对该临界资源进行访问;而当锁关上时,则禁止进程(线程)访问该临界资源。信号量机制是各操作系统采用的行之有效的同步机制。

猿 进程通信

进程通信的任务是用来实现在相互合作的进程之间的信息交换。通过进程之间的信息交换,来协调合作进程间的推进顺序。

在多线程环境下,为了加速应用程序的运行,应在系统中建立多个进程,并且再为一个进程建立若干个线程,由这些进程(线程)相互合作去完成一个共同的任务。这些进程(线程)之间,往往需要交换信息。例如,有三个相互合作的进程,它们是输入进程、计算进程和打印进程。输入进程负责将所输入的数据传送给计算进程;计算进程利用输入数据进行计算,并把计算结果传送给打印进程;最后,由打印进程把计算结果打印出来。进程通信的任务就是用来实现在相互合作的进程之间的信息交换。

进程通信方式主要有直接通信和间接通信两类。当相互合作的进程(线程)处于同一计算机系统时,通常采用直接通信方式,即由源进程利用发送命令直接将消息(数据)挂到目标进程的消息队列上,以后由目标进程利用接收命令从其消息队列中取出消息。间接通信采用信箱完成信息交换,主要用于处于不同计算机节点进程之间的通信。

源 进程调度

进程调度的任务是:从进程的就绪队列中选出一个进程,把处理机分配给它,并为它设置运行现场,使进程投入执行。

值得提出的是,在多线程系统中,通常是把线程作为独立运行和分配处理机的基本单位。为此,需把就绪线程排成一个队列。每次调度时,从就绪线程队列中选出一个线程,把处理机分配给它。

缘 内存管理

存储管理的主要任务是提高内存利用率,为用户程序提供足够的存储空间,方便进程并发执行,为多线程的运行提供良好的环境,为用户使用存储器提供方便。存储管理的功能包括以下几个方面:

员 内存分配

在实现内存分配时,可采取静态和动态两种方式。

静态分配方式:每个作业的内存空间是在作业装入时确定的;在作业装入后的整个运行期间,不允许该作业再申请新的内存空间,也不允许作业在内存中移动。

动态分配方式:每个作业所要求的基本内存空间,也是在装入时确定的,但允许作业在运行过程中,继续申请新的内存空间,以适应程序和数据的动态增长,也允许作业在内存中移动。

为了实现内存分配，在内存分配的机制中应具有这样的数据结构和功能：

(1) 内存分配数据结构：该结构用于记录内存空间的使用情况，作为内存分配的依据。

(2) 内存分配功能：系统按照一定的内存分配算法，为用户程序分配内存空间。

(3) 内存回收功能：系统对于用户不再需要的内存，通过用户的释放请求，去完成内存的回收功能。

4.1.2 内存保护

内存保护的主要任务是确保每道用户程序都只在自己的内存空间内运行，彼此互不干扰。为了确保每道程序都只在自己的内存区中运行，必须设置内存保护机制。一种简单实用的内存保护方式是通过设置两个界限寄存器，分别用于存放正在执行程序地址的上界和下界，系统对每条指令所要访问的地址进行检查，如果发生越界，便发出越界中断请求，以停止该程序的执行。越界检查由硬件实现，对发生越界后的处理，还须与软件配合来完成。

4.1.3 地址映射

地址映射实现进程逻辑地址到内存物理地址的转换，以便程序能够正确地运行。

一个应用程序（源程序）经编译后，通常会形成若干个目标程序。这些目标程序再经过链接便形成了可装入程序。这些程序的地址都是从 0 开始的，程序中的其他地址都是相对于起始地址计算的，我们把程序中使用的地址称为逻辑地址或相对地址，由这些地址所形成的地址范围称为逻辑地址空间。此外，由内存中的一系列单元所限定的地址范围称为物理地址空间，其中的地址称为物理地址。

在多道程序系统中，多道程序地址空间内的逻辑地址和内存空间中的物理地址往往不一致。为使程序能正确运行，存储器管理必须提供地址映射功能，将地址空间中的逻辑地址转换为内存空间中与之对应的物理地址。该功能应在硬件的支持下完成。

4.1.4 内存扩充

一般来说，内存容量是有限的。在多道程序设计环境下往往内存容量不能满足用户作业的需要。存储器管理中的内存扩充任务，是利用虚拟存储技术，从逻辑上扩充内存容量，使用户所感觉到的内存容量比实际内存容量大得多，而并非是去扩大物理内存的容量。这样，可以满足大作业的需要及增加内存中并发进程数，既改善了系统的性能，又基本上不增加硬件成本。

4.2 设备管理

设备管理的目标是提高 I/O 设备利用率，方便用户使用设备。在计算机系统的硬件资源中，除了 CPU 和内存外，还有许多外部设备。外部设备种类繁多，物理特性相差甚大。因此，操作系统的设备管理往往很复杂。设备管理的主要任务是：完成用户进程提出的 I/O 请求；为用户进程分配其所需的 I/O 设备；提高 CPU 和 I/O 设备的利用率；提高 I/O 速度；实现设备的独立性，方便用户使用 I/O 设备。

为此，设备管理应具有缓冲管理、设备分配与回收、设备处理以及虚拟设备等功能。

4.2.1 缓冲管理

CPU 运行的高速性和 I/O 低速性间的矛盾随着 CPU 速度的迅速提高，而使得此矛盾更为突出，严重降低了 CPU 的利用率。

在 I/O 设备和 CPU 之间引入缓冲，即在内存中开辟相应区域作为 I/O 设备与 CPU 之间

的缓冲区。例如,输入时先将数据输入到内存缓冲区中暂时保存,需要时,系统直接从内存缓冲区中读取,从而达到接近内存的存取速度使用慢速输入设备。这样可有效地缓和进程和设备速度不匹配的矛盾,提高进程的利用率,进而提高系统吞吐量。

设备分配与回收

设备分配与回收的基本任务是:根据用户进程的请求、系统现有资源情况以及按照某种设备分配策略,为之分配其所需的设备;如果在设备和设备之间,还存在着设备控制器和通道时,还须为分配出去的设备分配相应的控制器和通道。在用户进程使用设备完毕后,应立即由系统回收。

在进行设备分配时,应针对不同的设备类型而采用不同的设备分配方式。对于独占设备(临界资源)的分配,还应考虑到该设备被分配出去后,系统是否安全。

设备处理

设备处理程序又称为设备驱动程序。对于未设置通道的计算机系统,其基本任务是用于实现进程和设备控制器之间的通信。即由进程向设备控制器发出命令,要求它完成指定的操作;反之由进程接收从控制器发来的中断请求,并给予迅速的响应和相应的处理。

对于设置了通道的计算机系统,设备处理程序还应能根据用户的请求,自动地生成通道程序。

实现设备独立性和虚拟设备

为方便用户使用设备,提高设备可适应性,应考虑设备独立性(设备无关性)。设备独立性即设备无关性,是指应用程序独立于具体的物理设备,应用程序中使用逻辑设备,从而与物理设备的物理特性无关。设备管理还应该能实现虚拟设备的功能,即通过虚拟技术将低速的独占设备改造为高速的共享设备。

文件系统管理

处理机管理、存储管理和设备管理都属于硬件资源的管理,软件资源的管理称为信息管理。计算机系统中的信息(程序和数据)以文件的形式存储在外存储器中(如磁盘)供用户使用,操作系统由文件管理系统来实现信息管理。文件管理的主要任务是实现软件资源的存储、共享、保密和保护。文件管理包括:文件存储空间管理,目录管理,文件的读写管理和存取控制。

文件存储空间的管理

由文件系统对计算机系统中的文件及文件的存储空间实施统一的管理。其主要任务是为每个文件分配必要的外存空间,提高外存的利用率,并能有助于提高文件系统的运行速度。为此,系统应设置相应的数据结构,用于记录文件存储空间的使用情况,以供分配存储空间时参考;系统还应具有对存储空间进行分配和回收的功能。

目录管理

目录管理的主要任务是:为每个文件建立其目录项,并对众多的目录项加以有效的组织,以实现“按名存取”。为了使用户能方便地在外存上找到自己所需的文件,通常由系统为每个文件建立一个目录项。目录项包括文件名、文件属性、文件在磁盘上的物理位置等。由若干个目录项又可构成一个目录文件。其次,目录管理还应能实现文件共享,这样,只需在

外存上保留一份该共享文件的副本。此外,还应能提供快速的目录查询手段,以提高对文件的检索速度。

文件的读、写管理和保护

文件的读、写管理是文件管理的最基本功能。文件系统根据用户给出的文件名去查找文件目录,从中得到文件在外存中的存储位置,对文件进行读、写操作。

为了防止系统中的文件被非法窃取或破坏,在文件系统中应建立有效的保护机制,防止未经核准的用户存取文件,防止冒名顶替存取文件,防止以不正确的方式使用文件,以保证文件系统的安全性。

用户接口

操作系统的作用之一是提供一个友好的用户访问计算机系统的接口。它屏蔽了计算机硬件的操作细节,使用户或程序员与系统硬件隔离开来。用户通过使用这些接口达到方便使用计算机的目的。操作系统为用户提供了命令接口、程序接口与图形用户接口三种接口。

命令接口

(1) 联机用户接口。这是为联机用户提供的,它由一组键盘操作命令及命令解释程序所组成。当用户在终端或控制台上每键入一条命令后,系统便立即转入命令解释程序,对该命令加以解释并执行该命令。在完成指定功能后,控制又返回到终端或控制台上,等待用户键入下一条命令。

(2) 脱机用户接口。该接口提供给批处理作业用户用于组织和控制自己的作业运行。故也称为批处理用户接口。该接口由一组作业控制语言(如 JCL、RJE 等)组成。批处理作业的用户不能直接与自己的作业交互作用,只能委托系统代替用户对作业进行控制和干预。这里的作业控制语言便是提供给批处理作业用户的、为实现所需功能而委托系统代为控制的一种语言。用户用作业控制语言把需要对作业进行的控制和干预,事先写在作业说明书上,然后将作业连同作业说明书一起提供给系统。当系统调度到该作业运行时,又调用命令解释程序,对作业说明书上的命令,逐条地解释执行。

程序接口

程序接口是为用户程序在执行中访问系统资源而设置的,是用户程序取得操作系统服务的惟一途径。它是由一组系统调用组成,每一个系统调用都是一个能完成特定功能的子程序,每当应用程序要求操作系统提供某种服务时,便调用具有相应功能的系统调用。

图形用户接口

以终端命令和命令语言方式来控制程序的运行固然有效,但用户必须记住各种命令,并从键盘输入这些命令以及所需数据来控制程序的运行。随着大屏幕高分辨率图形显示和多种交互式输入输出设备(如鼠标、光笔、触摸屏等)的出现,使得命令接口图形形式成为可能。图形用户接口采用了图形化的操作界面,用非常容易识别的各种图标(如窗口、菜单等)来表示系统的各项功能、各种应用程序和文件,直观、逼真地表示出来,通过对出现在屏幕上的对象直接进行操作,控制和操纵程序的运行。图形用户接口可以看成是命令接口的另一种形式,但它大大减轻或免除了用户记忆的工作量,极大地方便了用户。

员操作系统的的基本类型

计算机诞生初期(1946年到 1960年代中期,属于第一代计算机),计算机上没有配置操作系统,用户(即程序员)直接使用机器指令设计程序。程序的装入、调试以及控制程序的运行都是通过控制台上的开关来实现的。在这期间,整个计算机都被一个程序员所占有。当程序运行时,通过控制台上的开关和显示灯来管理他的程序。如果发现错误,程序员暂停程序的运行,测试主存和寄存器的内容,然后直接从控制台上排除程序错误。

随着计算机的发展,硬件增加了读卡机、行式打印机和磁带机。程序员将程序与数据通过穿孔方式记录在纸带(对应于程序和数据)或卡片上,然后把记录程序和数据后的卡片(或纸带)装入卡片输入机(或纸带输入机),再启动它们将程序和数据输入计算机,最后,计算机执行程序。当程序运行完毕并取走计算结果后,才让下一个用户上机。在这种工作方式下,输入输出都是在主机控制下进行的,故称为联机输入输出方式。

随着计算机硬件的发展,计算机处理能力大大提高,高速的运算和低速的设备之间速度不匹配的矛盾日益突出。另一方面,人工操作的低效率(执行前,手工装入纸带或卡片;程序运行完成后,手工卸取纸带或卡片),使计算机资源利用率降低,产生了所谓“人机矛盾”。为了缓和这对矛盾,操作系统引入了脱机输入输出方式与批处理技术。

所谓脱机输入输出方式是指输入输出不是在主机控制下完成,而是由一台功能简单、成本较低的专门负责输入输出的计算机控制完成,这种计算机也称外围机或卫星机。脱机输入方式是在外围机控制下,事先将程序和数据从低速输入设备输入到速度快得多的磁带上,需要执行程序时主机从磁带上读入程序与数据。脱机输出方式是主机将程序执行结果写入到磁带上,在外围机控制下,将磁带中的程序结果从低速输出设备输出。以脱机输入输出方式工作时主机不直接与慢速输入输出设备相连,只与磁带机相连,从而大大地缓解了高速的处理机与慢速的输入输出设备之间的矛盾,提高了系统性能。

批处理机技术是指将一批用户作业提交给系统,由系统自动控制作业顺序执行的技术。

20世纪 60年代末至 70年代初,计算机发展进入了第二代。不仅计算速度有了很大提高,而且存储容量大幅度增长,同时计算机软件也得到了较好的发展。先后出现了 FORTRAN、ALGOL 和 COBOL 等程序设计语言及相应的编译程序,方便了程序员编程。由于对计算机的操作仍停留在手工操作方式上,用户要不断地装卸卡片或磁带,等待用户,其利用率低。于是,迫切需要一种能对计算机硬件和软件进行管理与调度的软件,这种软件称为监督程序(管理程序)。当时,在美国的 IBM 系列计算机系统上就配置了这种监督程序软件。

有了监督程序后程序员不必亲自上机操作,而将要执行的程序与数据交给操作员即可。操作员从控制台输入命令,然后由监督程序来识别和执行。这样,不仅提高了操作速度,而且在执行过程中发生错误时,监督程序将输出相应信息向操作员报告。监督程序还管理计算机的部分资源。用户程序总是通过监督程序去启动设备,当设备发生故障时,监督程序将负责处理。监督程序还具有文件管理功能,用户可以实现按文件名而不是物理地址来存取信息,这不仅方便灵活,而且安全可靠。这种监督程序已经具有了操作系统的某些特征与功能。

随着计算机硬件的发展以及越来越广泛的应用需求，人们开始考虑如何进一步提高资源的利用率和系统性能，以及更方便地使用计算机，从而推动了操作系统的形成与发展。

操作系统一般可以分成以下几类：批处理操作系统、分时操作系统、实时操作系统、网络操作系统和分布式操作系统。其中批处理操作系统、分时操作系统、实时操作系统是三类基本的操作系统。下面主要介绍这三类操作系统。

1.1 批处理操作系统

批处理操作系统的基本特征是“批量处理”，它是将作业成批装入计算机，由操作系统将其组织好，按某种调度算法选择一道或几道作业装入内存运行。它的设计目标主要是提高资源利用率与系统的吞吐量（单位时间内完成作业的数量）。批处理操作系统分为单道批处理系统和多道批处理系统两种。

1.1.1 单道批处理系统

单道批处理系统是一种早期的批处理操作系统，作业是按批量顺序进入内存，每次只有一道作业在内存中运行，该作业运行结束或出现异常时，由监督程序控制自动装入下一道作业运行，这样缩短了作业切换时间，实现了一批作业自动连续运行，缓解了越来越突出的“人机矛盾”，提高了系统的资源利用率与系统吞吐量。其特点是：

- （1）单道性。监督程序每次只从磁带上装入一道作业到内存中运行，程序运行结束或出现异常才装入其后的作业运行。
- （2）顺序性。磁带上的一批作业按顺序装入内存执行。
- （3）自动性。磁带上的一批作业在监督程序的管理下自动执行，无须人工干预。

单道批处理系统的执行过程如图 1-1 所示：作业 A 首先装入内存，在 $t_1 \sim t_2$ 时段占用 CPU 运行。在 $t_2 \sim t_3$ 这段时间执行 I/O 操作，CPU 空闲。当作业 A 在 t_3 时刻运行结束后，在 t_3 时刻由监督程序将作业 B 装入内存运行，直到 t_4 时刻结束。

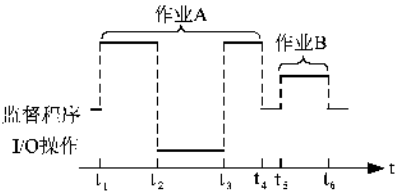


图 1-1 单道批处理系统的执行过程

1.1.2 多道批处理系统

单道批处理系统在一定程度上解决了“人机矛盾”以及高速 CPU 与慢速 I/O 设备间速度不匹配的问题，提高了系统的资源利用率。但是由于单道系统本身的“单道性”特点，无法实现处理机与 I/O 设备的并发执行，难以再进一步提高系统的资源利用率。随着计算机硬件的发展，20 世纪 60 年代中期，中断和通道技术的出现使得输入输出设备与中央处理器可以并行工作。一个作业启动了输入输出设备后信息的传送工作由通道来控制，暂时不使用 CPU 可以让其去执行其他程序。为进一步提高系统的资源利用率（特别是 CPU 的利用率）与系统吞吐量，引入了“多道程序设计”技术，形成了多道批处理操作系统。

所谓多道程序设计是指在内存中装入多道作业，使它们同时运行、共享系统资源。在单处理机系统中，内存中的作业只是在宏观上“同时”运行，即指多道作业都已开始运行，但尚

未完成。在微观上，各作业是交替执行。任意时刻只有一道作业在处理机上运行。

引入了多道程序设计技术的批处理系统称为多道批处理系统（配置是月操作系统，月操作系统早系统）。

多道批处理系统的基本思想：用户提交的作业先在外存排成一个队列（后备队列），由作业调度程序按照一定的策略从中选择若干作业调入内存运行，共享系统中的各种资源。

把用户准备好的一批作业信息（包括程序、数据、作业控制说明书）通过相应的输入设备传送到磁盘上，等待处理。操作系统中的作业调度程序按某种调度原则从磁盘上选择若干作业装入主存储器，主存中的这些作业可交替地占用处理器运行。当某个作业执行结束时，启动打印机，输出计算结果。然后，又可从磁盘上选择作业装入主存储器，让其执行。与此同时，还可把新的作业信息通过输入设备传送到磁盘上等待处理，多道批处理系统的控制如图 员原原所示。

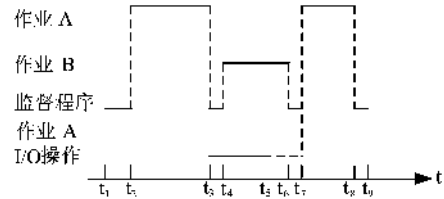


图 员原原 多道批处理系统的控制示意图

从图 员原原中可看出，作业 粤在 赋时刻被调度运行，到 赋时刻进行 赋的操作，由监督程序接管 悦载并在 赋时刻调度 月作业运行。在 赋~ 赋时段，作业 月占用 悦载运行，作业 粤在进行 赋的操作，实现了 悦载与 赋设备的并行操作。从而提高了系统资源的利用率与系统工作效率。具体表现为：

（员 多道作业并行工作，减少了处理机的空闲时间，即提高了处理机的利用率。

（圆 作业调度可以按一定的组合选择装入主存储器的作业，只要搭配合理，例如，把请求使用不同设备的作业组合在一起，计算型作业与 赋型作业合理搭配，则可充分利用计算机系统的资源。

（猿 成批输入、自动选择和控制作业执行，减少了人工操作时间和作业切换时间。作业执行过程中，不再访问低速的设备，而是直接在高速的磁盘上存取信息，缩短了作业执行时间，有利于提高系统的吞吐量。

多道批处理系统虽然大大提高了计算机系统的资源利用率与系统吞吐量，但系统的设计也更为复杂，必须解决好下面的问题：

（员 处理机分配、回收。在内存中的多道程序竞争处理机，必须有相应的方法来解决处理机的分配与回收问题。

（圆 存储保护机制。要保证进入内存的多道程序在各自的内存区顺利运行，互不干扰。

（猿 合理的 赋设备分配机制。多道程序同时运行，共享 赋设备，必须有合理的分配策略，保证各程序有序使用外部设备，避免死锁的产生。

（源 作业管理。根据具体情况，选择合理的作业调度算法调度作业运行。

（缘 文件管理。有完善的文件管理功能，提供有效、快速的信息检索、存储机制；提供文件保护机制。

猿多道批处理系统中的作业执行过程

在批处理系统中，从作业的提交到作业的完成大体上分为以下四个阶段：

（员 提交。用户首先准备好作业，然后提交给系统，此阶段称为作业的提交。作业所处

的状态为提交状态。

④ 后备。一批作业提交后，系统将作业存放到输入井（磁盘上的一块特定区），在后备队列中排队，并等待执行，称此阶段为作业的后备或称作业的收容；作业处于后备状态。

⑤ 执行。作业调度程序从磁盘的输入井的后备队列中挑选作业装入内存，创建进程，然后由进程调度程序调度，使它们在处理机上执行，此阶段称为作业的执行；作业处于运行状态。

⑥ 完成。作业执行结束后，系统收回资源，取出作业执行结果，这一阶段称为作业的完成；作业处于完成状态。

一个作业结束后，系统根据当时资源的使用情况以及处于后备阶段的作业特点，按相应策略重新选择一个或几个作业装入内存并使其在处理机上执行。

一个作业从进入系统到退出系统所经历的状态及其转换过程，如图 1-1 所示。

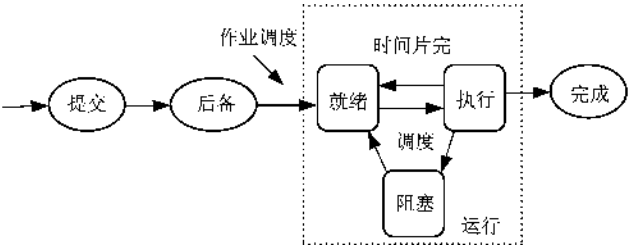


图 1-1 多道批处理系统中的作业状态及其转换

多道批处理系统的主要特征

与单道批处理系统相比，具有以下三个特征：

① 多道性。允许内存中同时装入多个作业并发执行，以充分利用系统资源。多道批处理系统的资源利用效率是基于各作业对系统资源的需求差异得到的。

② 调度性。在外存后备队列中的多个作业，哪些装入内存，由作业调度程序按一定的策略来调度，作业装入内存并创建进程后，再由进程调度程序分配处理机。

③ 无序性。与单道批处理系统不同，在内存中的多个作业的执行次序、完成时间都与进入内存的先后顺序没有直接的对应关系。

多道批处理操作系统大大提高了计算机系统的资源利用率与系统吞吐量，它的出现标志着操作系统的形成。但是批处理系统无交互能力，在作业执行时用户不能直接干预作业的执行。这样，用户就不能及时了解自己的程序运行情况。当程序出现错误时，用户也不能及时加以控制，只能由操作系统输出信息，再由操作员通知用户重新修改程序，然后再次提交给系统重新装入执行，降低了程序开发效率。另一方面，多道程序在内存中共享资源，其互相制约使得作业的平均周转时间（周转时间：作业从进入系统到结束并退出系统所经历的时间）延长。

分时操作系统

分时系统的概念

批处理系统虽然提高了资源利用率与系统吞吐量，但无法实现人机交互，为了解决这一问题，引入了分时系统。

所谓分时系统是指一台主机与多个终端相连，允许多个用户通过终端同时以交互的方式使用计算机系统，共享资源，这种系统使得每个用户感到好像自己独占一台支持自己请求服务的计算机系统。

在分时系统中，一个计算机系统与许多终端设备连接。最简单的终端可以由一个显示器和一个键盘组成。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。系统根据用户的请求完成指定的工作，并把执行情况返回给用户。然后，用户根据上一个请求的执行结果，又向系统提出下一步的请求。重复系统与用户的交互会话过程，直至用户完成自己的全部工作。

在分时系统中，为了使一个计算机系统能同时为多个终端用户服务，系统采用了分时技术，即把系统时间分割成一定大小的时间段，每个时间段称为一个时间片。每个终端用户每次可以使用一个由时间片规定的系统时间。这样，多个终端用户就轮流地使用系统。如果某个用户在规定的的一个时间片内还没有完成它的全部工作，这时也要把系统时间让给其他用户，等待下一轮再使用一个时间片的系统时间，循环轮转，直至结束。

分时系统与多道批处理系统有所不同。多道批处理系统是实现自动控制无需人为干预的系统，而分时系统是实现人机交互的系统。

分时系统的实现

分时系统有不同的实现形式：

(1) 单道分时系统：早期的分时系统，内存中只允许一道作业运行，系统采用调入调出的方式来实现，系统开销太大，系统性能较差。

(2) 前台后台分时系统：这种实现方式是把内存划分成前台区与后台区，前台区存放分时作业，优先执行；后台区存放批处理作业，当前台区没有分时作业时，系统调度后台区的批处理作业运行。

(3) 多道分时系统：多道程序在内存中同时执行。

为了实现分时系统，必须解决好下面两个问题：

(1) 及时接收。即能及时地接收来自终端用户的命令，通过使用多个缓冲端口，设立多路缓冲区来保证及时接收输入。

(2) 及时响应。即能及时地响应用户输入的命令，响应时间要短。所谓响应时间是指从用户输入命令后到系统将处理结果提交给用户（如在屏幕上显示结果）所经历的时间。响应时间可以表示为：

其中 n 为用户数， t 为时间片， k 为系统开销。

为保证合适的响应时间，可以通过限制用户数目、缩短时间片以及提高对换速度来实现。值得注意的是缩短时间片可能引起对换次数增多，系统开销增大，反而会降低系统性能，所以时间片不能划分得太小。

分时系统的特征

与批处理系统相比，分时系统具有如下特征：

(1) 多路性。允许多个终端用户同时使用一个计算机系统，共享系统资源，提高了资源利用率，促进了计算机的普遍应用。

(2) 独立性。用户在各自的终端上请求系统服务，彼此独立，互不干扰。当某用户在输入命令时，操作系统将把 CPU 分配给其他用户使用，而不让 CPU 处于空闲；当用户输入命令结束时又能让他使用 CPU。这样，给用户的感觉就像是每个用户只有自己在单独使用计算机系统，而实际上计算机系统正在被许多用户分享。

(3) 及时性。系统能及时对用户的操作进行响应，显著提高调试和修改程序的效率，缩短了周转时间。

(4) 交互性。采用联机（即人—机对话）的工作方式，用户在终端上可以直接输入、调试和运行自己的程序，能及时修改程序中的错误且直接获得结果。

分时系统为用户在测试、修改程序以及在控制程序执行方面提供了很大的灵活性。

实时操作系统

计算机不但广泛应用于科学计算和数据处理，也广泛用于各种实时自动控制系统中。如工业自动控制系统、导弹发射系统、票证预订管理系统、银行证券系统等方面，通常我们称之为实时系统。所谓“实时”是指对随机发生的外部事件作出及时的响应并能对其进行处理。对不同的系统，其“及时”性也要求不一样。所谓外部事件是指来自与计算机系统相连接的设备所提出的服务要求，而不是由人来启动和直接干预而引起的。上面介绍的批处理操作系统和分时操作系统可以获得令人满意的资源利用率、响应时间以及用户的交互性，但随着计算机应用领域的推广，还无法满足对实时采样数据进行及时（实时）处理的要求，所以引入了实时操作系统。

实时操作系统的主要特点是响应及时和可靠性高。它的设计目标是能对特定的输入作出及时响应，并在规定的时间内完成对该事件的处理。

两种实时系统

实时系统通常包括实时过程控制和实时信息处理两种系统。

(1) 实时过程控制系统。在这类应用中要求计算机系统实时采集被测量系统数据，并对其及时进行加工处理及输出。它主要用于生产过程和军事自动控制领域。

(2) 实时信息处理系统。在这类应用中要求计算机系统能对用户的服务请求及时作出回答，并能及时修改、处理系统中的数据。它主要用于像飞机票的预订、银行证券的财务管理等大量数据处理的实时系统。

实时任务的类型

实时任务可以按执行是否呈现周期性和按截止时间来划分。

(1) 按任务执行是否呈现周期性，可划分为：周期性实时任务与非周期性实时任务。

周期性实时任务：是指外部设备所发出的服务请求信号具有周期性。

非周期性实时任务：是指外部设备所发出的服务请求信号并无明显的周期性。

(2) 按截止时间来划分：为了控制实时性，每一实时任务都有一截止时间（即任务必须在某时间以前开始执行）。

开始截止时间：即最晚开始时间，任务在某时间以前必须开始执行。

完成截止时间：即最晚完成时间，任务在某时间以前必须完成。

根据对截止时间的要求不同，实时任务可分为：硬实时任务（与截止时间严格相关）、软实时任务（与截止时间松散相关）。硬实时任务是指：系统必须满足任务对截止时间的要求，否则可能出现难以预测的结果，严重影响系统的运行。对软实时任务，同样也联系着一个截止时间，但并不严格，若偶尔错过了任务的截止时间，对系统产生的影响也不会太大。

实时系统的特征

（员 多路性：实时信息处理系统允许多个终端用户同时共享使用一个计算机系统，实时控制系统也可以对多个控制对象进行控制或进行多路现场信息采集。

（圆 独立性：实时信息处理系统用户在各自己的终端上请求系统服务，彼此独立，互不干扰，每个用户感觉只有自己在使用计算机系统。实时控制系统对多个控制对象进行的控制或进行多路现场信息采集也是相互独立的。所以实时系统与分时系统一样具有独立性。

（猿 及时性：实时系统对及时性的要求比分时系统更高。特别是实时控制系统，它根据控制对象所能等待时间来确定。

（源 交互性：实时系统具有一定的交互性，特别是实时信息处理系统，允许用户输入数据，提出有限的服务请求，但交互性较分时系统弱。

（缘 可靠性：实时系统要求有非常高的可靠性，否则可能带来灾难性后果。因此实时系统具有一定的容错能力，如数据热备份、关键设备冗余备份（如双机）和故障自动复位等。

实时操作系统虽然也具有分时操作系统的四个特征，但一般来说，它的交互能力较弱，但及时性较强。实时操作系统除了具有一般操作系统的功能外，它在设计时还应考虑：实时时钟管理、快速中断处理能力、系统运行的安全可靠性以及具有较强的系统重组能力等，一般它是以时间（周期任务）和事件驱动任务的。

其他操作系统

多处理机操作系统

多处理机（配置多个处理器）系统是由多个处理机组成的计算机系统，它的出现是为了提高计算机系统性能和可靠性。多处理机系统分为紧密耦合（共享总线型）与松散耦合（星型或环型）。前者各处理机之间通过快速总线或开关阵列相连，共享内存，整体系统由一个统一的资源管理（一个资源核心）；后者的各处理机带有各自的存储器、I/O设备和操作系统，通过通信线路相连，每个处理机上独立运行资源

运行在多处理机系统上的操作系统称为多处理机操作系统（配置多个处理器的资源管理）。多处理机操作系统是在单处理机操作系统基础上发展而来，其最大的差别在处理机的管理与调度上。

网络操作系统

所谓计算机网络是指一组自主的计算机系统，通过通信设施相互连接，完成信息交换、资源共享、互操作和协同工作等功能构成的系统。网络操作系统（简称 NOS，网络操作系统）是在一般操作系统功能的基础上提供网络通信和网络服务功能的操作系统，它是为网上计算机进行方便而有效的网络资源共享，提供网络用户所需各种服务的软件和相关协议的集合。网络功能与操作系统的结合程度是网络操作系统的重要性能指标。网络操作系统除了应有的处理机管理、存储器管理、设备管理、文件管理和提供用户接口的功能之外，还应

具有：提供高效、可靠的网络通信能力；提供多种网络服务功能，如文件传输、邮件服务等。

1.1.2 分布式操作系统

分布式系统是以计算机网络为基础的，它的基本特征是处理上的分布，即功能和任务的分布。分布式操作系统（即网络操作系统）的所有系统任务可在系统中任何处理机上运行，自动实现全系统范围内的任务分配并自动调度各处理机的工作负载。

分布式操作系统与网络操作系统的比较：

（1）耦合程度

分布式系统是紧密耦合系统：分布式操作系统是在各处理机上统一建立的，直接管理存储器和外设；统一进行全系统的管理。网络操作系统通常容许异种操作系统互连，各处理机上各种服务程序需按不同网络协议来实现。

（2）并行性

分布式操作系统可以将一个任务的多个进程分散在各处理机上并行执行；网络操作系统没有任务分派功能，任务只在本地机上运行。

（3）透明性

分布式系统的网络资源调度对用户透明，用户不了解所占有资源的位置；网络操作系统中对网络资源的使用要由用户明确指定。

（4）健壮性

分布式系统的处理和控制功能是分布的，任何节点的故障不会给系统造成太大影响，而且分布式系统具有更强的容错能力，能够实现系统重构，以保证系统正常运行。而网络系统的控制功能都集中在服务器中，使系统具有潜在的不可靠性。

1.1.3 个人计算机操作系统

针对单用户使用的个人计算机进行优化的操作系统。要求使用方便、支持多种硬件和外设（多媒体设备、网络、远程通信）。其应用领域主要是事务处理、个人学习与娱乐等。

现代操作系统，通常具有分时、实时和批处理功能，又称作通用操作系统，可适用于计算、事务处理、实时控制等多种领域，能运行在多种硬件平台上，如微型计算机系统、大型计算机系统、网络系统等。

1.2 操作系统的特征与性能

1.2.1 操作系统的特征

操作系统虽然有各种类型，但以多道程序设计为基础的操作系统都具有以下特征：

（1）并发

从一般意义来说，并发性是指两个或多个事件在同一时间间隔内发生。在多道程序环境下，并发是指在一段时间内，宏观上有多个程序在同时运行。这种允许多道程序并发运行的系统称为并发系统。多道与并发是同一事物的两个方面。多道程序设计的实现，使得多个程序能够并发执行；而程序的并发执行又导致多个程序竞争一台处理机。在单处理机系统中，每一时刻仅能有一道程序执行，故微观上这些程序只能是分时地交替执行。操作系统也是一个并发系统，其本身就是与用户程序一起并发执行的。操作系统要完成各进程间的并发、系

统与应用程序间的并发的管理。

程序的并发执行提高了计算机系统的资源利用率与系统吞吐量，但同时也带来了串行执行所没有的新问题，导致了操作系统对程序管理的复杂化以及操作系统本身的复杂化。

并行性（~~并行性~~）是与并发性既相似又有区别的一个概念。它指两个或多个事件在同一时刻发生。并行性具有并发的含义，但并发事件不一定具有并行性。在多道程序环境下，并发性是指在一段时间内，宏观上有多个程序在同时运行。但在单处理机系统中，每一时刻仅能有一道程序执行，微观上这些程序只能是分时地交替执行，即不能并行执行。而在多处理机系统中，这些可以并发执行的程序便可被分配到多个处理机上，实现并行执行，即多个程序可同时执行。

资源共享（~~资源共享~~）

共享是指系统中的资源可供内存中多个并发执行的进程共同使用。由于资源有限，操作系统必须对系统资源进行合理分配和使用。根据资源属性的不同（分为共享资源与独占资源），进程对资源共享的方式也不同，目前主要有以下两种资源共享方式；

（**互斥共享方式**）。系统中的独占资源，如音频设备、打印机等。虽然它们可以提供给多个进程使用，但在一段时间内资源分配给某进程后到使用完毕释放前，不能被其他进程所用。这类资源只能用互斥方式来共享。

（**同时访问方式**）。系统中还有另一类资源，如磁盘设备、可重入代码等，在同一段时间内可以由多个进程同时对它们进行访问。这里所谓的同时往往是宏观上的，而在微观上，这些进程可能是交替地对该资源进行访问，但交替访问的顺序不会影响执行的结果。如磁盘是典型的共享设备，在一段时间内可以有多个进程进行读写，而在任意时刻只能有一个进程访问磁盘。

资源虚拟（~~资源虚拟~~）

所谓虚拟，是指通过某种技术（分时或分空间）把一个物理实体变为若干个逻辑上的对应物。物理实体是实的，即实际存在的；而后者是虚的，是用户感觉上的东西。用于实现虚拟的技术，称为虚拟技术。虚拟是操作系统管理系统资源的重要手段，能大大提高资源利用率。

虚拟技术在操作系统中的应用主要体现在以下方面：

（**通过多道程序设计技术**，让多道程序并发执行，分时使用一台处理机，将处理机虚拟为多台逻辑上的处理机，让每个用户都感觉自己在使用的是一台处理机，把用户所感觉到的处理机称为虚拟处理机。

（**通过虚拟存储器技术**，如请求式分页管理，来实现虚拟内存管理，从逻辑上“扩充内存容量”，为每个作业建立一个虚拟存储器。

（**在设备管理中，通过 ~~虚拟设备~~ 技术来实现虚拟设备**，可以将独占设备改造为供多个进程共享的逻辑设备。

资源异步性（~~资源异步性~~）

操作系统的异步性，是指进程是以人们不可预知的速度向前推进，也称不确定性。在多道程序环境下，允许多个进程并发执行，但只有进程在获得所需的资源后方能执行。由于资源等因素的限制，使进程的执行通常都不是“一气呵成”，而是以“时走时停”的方式运行。

例如，正在执行的进程提出打印请求，而此时打印机又被其他进程占用，由于打印机属

于独占资源(临界资源),每次只允许一个进程使用,因此请求打印机的进程必须等待,且放弃处理机,直到打印机空闲,并再次把处理机分配给该进程时,它才能继续执行。可见,内存中的每个进程在何时能获得处理机运行,何时又因提出某种资源请求而暂停,以及进程以怎样的速度向前推进,每道程序总共需多少时间才能完成等等,都是不可预知的。可能先进入内存的作业后完成,而后进入内存的作业先完成。

值得注意的是不确定性是指进程的执行顺序和执行时间不确定,系统在某个时刻的状态难以重现,而不是执行结果不确定。只要运行环境相同(如输入参数相同),作业经多次运行,都会获得完全相同的结果。这一点是通过进程互斥和同步手段来保证的。因此,允许进程以异步方式运行是操作系统的一个重要特征。

操作系统的并发、共享、虚拟、异步性四个特征不是相互独立的,它的具有密切的关系:

并发和共享是操作系统的两个最基本的特征,它们又是互为存在的条件。一方面,资源共享是以程序(进程)的并发执行为条件的,若系统不允许程序并发执行,自然不存在资源共享问题;另一方面,若系统不能对资源共享实施有效管理,协调好诸进程对共享资源的访问,也必然影响到程序并发执行的程度,甚至根本无法实现并发执行。

虚拟与异步性是操作系统的两个重要特征。虚拟技术为共享提供了更好的条件,而并发与共享是导致不确定性的根本原因。

1.1 操作系统的性能

操作系统的性能如何,直接影响计算机系统的总体性能。操作系统要能有效地使用系统资源,尽可能快地响应用户请求,方便用户使用计算机。一般从下面几个方面来评价操作系统的性能。

1.1.1 系统效率

操作系统的效率主要体现在:

(1) 资源的利用率要高。处理机空闲时间要少,内存利用率高,外部设备利用均衡。

(2) 对批处理系统来说,作业的平均周转时间要短,吞吐量要大。

(3) 对分时系统来说,其响应时间要尽量短。

1.1.2 系统可靠性与安全性

一个运行可靠、安全的操作系统是大家所期待的。由于操作系统是整个计算机系统的所有资源(包括硬件与软件)的管理者,它的可靠性与安全性,直接影响着整个计算机系统的运行可靠性与安全性。此外,由于系统的使用环境复杂多变,用户的误操作也可能造成系统工作不正常。在系统运行时,能及时检测出错误,并能尽量准确地指出错误发生的原因,采取相应的措施排除错误;对错误产生的损害,能进行修复,使系统恢复正常工作,尽量减少损失,具有较好的健壮性。

1.1.3 系统可维护性

众所周知,在软件的生命期中,其维护成本已占软件成本的 20% 左右。操作系统和普通软件一样应具有较好的可维护性。操作系统往往是一个庞大的软件集合,目前还无法证明软件的正确性,操作系统本身存在错误与不足在所难免,因而对操作系统进行纠错性维护是必须的。另外,随着应用领域的扩大,硬件的扩充,计算机体系结构的变化,新功能的增加,都将要求对操作系统进行修改维护,这就要求操作系统有较好的结构,完备的文档,以便于维护。

操作系统是否具有较好的维护性是决定操作系统生命周期长短的重要因素。

濺方便性

系统提供的各种服务方便用户使用。

员豫哉羣戴操作系统特征

多用户操作系统是美国电报电话公司(贝尔)月实验室的 和 于 年在一台闲置的 原形机器上设计实现的 位操作系统,并在 年被移植到 原形机器上。系统最初的版本使用汇编语言编写的, 年, 用 悦语言重写了 的大部分程序代码。 年, 系统正式发表(第 版), 并且在大学得到使用和好评。到 世纪 年代 以 和 为主分别开发, 有多种版本。 年, 们(系统)发表 系统 是 使 和 在用户界面上统一。进入 世纪 年代, 得到迅速的发展和广泛应用, 系统开发商加强了在网络方面的研究, 不断地推出用于企业网络的 网络 版本。人们普遍认为 是目前惟一能稳定运行在从微型机到大、中型等各种机器上的安全性好、运行稳定的操作系统。

散打系统能取得如此巨大的成功，其原因可归结为该系统具有以下特性：

(员 开放性

人们普遍认为 哉羈戴是目前开放性最好的操作系统。开放性是 哉羈戴系统最本质的特征。所谓开放性是指系统遵循国际标准规范,凡遵循国际标准所开发的硬件和软件,能彼此兼容,可方便地实现互连。开放性已成为 圆世纪 怨年代计算机技术的核心问题,也是一个新推出的系统或软件能否被广泛应用的重要因素。开放性对 哉羈戴的普及和发展起到了很大的促进作用。

(圓) 多用户、多任务环境

分时系统是一个多用户、多任务系统。它既可以同时支持数十个乃至数百个用户通过各自的联机终端同时使用一台计算机，而且还允许每个用户同时执行多个任务。例如，在进行字符图形处理时，用户可建立多个任务，分别用于处理字符的输入、图形的制作和编辑等任务。

③ 功能强大，实现高效

微操作系统在结构上分成内核与外壳两大层。内核程序的功能、使用的算法以及数据结构经过设计者精心设计,不但具有足够强的功能,而且也很精练简洁。由于微操作系统的内核占用空间小,常驻内存,能保证系统高效运行。微操作系统外壳提供了丰富的语言处理程序与实用程序,为用户提供了完备的程序设计环境。微操作系统提供了丰富的系统功能,使用户可方便地、快速地完成许多其他系统所难于实现的功能。微操作系统的内核能为外壳提供强有力的支持,而外壳也能很好地利用内核对其的支持,两者有机地结合,使微操作系统成为了世界上功能最强大、小巧、灵活的操作系统。

此外，它在许多功能的实现上还有其独到之处，并且是高效的。例如，**哉驛載**的目录结构、磁盘空间的管理方式、**障重定向**和管道功能等。**哉驛載**的文件系统采用多级目录，文件卷可以在子目录下动态装卸。采用设备文件的形式（读写，参数控制）简化了设备管理。**哉驛載**系统把设备当作文件处理，用户无须区分文件与设备，把普通文件、文件目录及外部设

备都作为文件统一处理,既简化了系统设计,也便于用户使用。

（缘）提供了丰富的网络功能

哉系统还提供了十分丰富的网络功能。作为 随网络技术基础的 裁协议,便是在 哉系统上开发出来的,并已成为 哉系统不可分割的部分。哉系统 还提供了许多最常用的网络通信协议软件,其中包括网络文件系统 晕软件、客户 服务器协议软件 晕软件、网络 数据库软件等。通过这些产品可以实现在各 哉系统之间,哉与 晕、 以及 杂原 晕、 晕等网络之间的互联和互操作。

（缘）支持多处理器功能

与 宰及 晕等操作系统相比,哉是最早提供支持多处理器功能的 韵。它所能支持的多处理器数目也一直处于领先水平。例如,灵年推出的 晕只能支持 员个处理器,而 宰最多也只支持 员个处理器,然而 哉系统在 怨年代中期,便能支持 员个处理器。

（远）移植性好

哉的外壳程序与大部分的内核程序都是用 悦语言编写的,而 悦语言是一种与硬件无关的编程语言,因而 哉可以方便地移植到不同的硬件平台上。在 哉系统下开发的应用程序也能方便地移植到配有 哉的其他机器上。

本章小结

操作系统是控制和管理计算机硬件和软件资源、合理地组织计算机工作流程以及方便用户使用计算机系统的程序集合。操作系统设计的两个主要目标是方便用户与提高系统资源的利用率。韵是在计算机裸机上扩充的第一层软件,其作用是:从一般用户的观点来看,韵是用户与计算机系统间的接口,用户并不直接与计算机硬件打交道,而是通过 韵提供的程序接口和命令来使用计算机;从资源管理的观点来看,韵是计算机系统资源的管理者,管理的对象是 悦、存储器、 设备与文件,管理的内容是:资源的当前状态、资源的分配、回收和访问操作,相应管理策略等;从虚拟机的观点来看,韵是加在计算机硬件之上的第一层软件,它扩充了裸机的功能,此时用户所看到的机器是一台比裸机功能更强、使用更方便的虚拟机;从任务组织的观点来看,韵是计算机工作流程的组织者,它负责多道作业间处理机切换,并协调它们的推进顺序。

操作系统是计算机系统资源的管理者,其主要任务是对系统中的硬件、软件资源实施有效的管理,以提高系统资源的利用率。操作系统的主要功能包括处理机管理、内存管理、设备管理、文件管理以及为方便用户而提供友好的用户接口。

操作系统可以分成以下几类:批处理操作系统、分时操作系统、实时操作系统、网络操作系统和分布式操作系统。其中批处理操作系统、分时操作系统、实时操作系统是三种基本的操作系统。多道程序设计是操作系统的基础,多道程序并发执行,利用 悦与外设并行工作,提高了计算机系统资源的利用率以及系统运行吞吐量。

以多道程序设计为基础的现代操作系统,都具有并发性、共享性、虚拟性、不确定性等特征。对操作系统的性能主要从有效性、方便性、可靠性、安全性以及可维护性几个方面来评价。

习题

1. 什么是操作系统？它的主要作用和功能是什么？

2. 试述虚拟机的概念。

3. 操作系统与系统中的其他软件以及硬件是什么关系？

4. 解释下列名词：脱机输入输出；吞吐量；实时事件。

5. 什么是多道程序设计？引入它的目的是什么？要解决哪些主要问题？

6. 操作系统有什么特征？它们的关系如何？

7. 比较批处理系统、分时系统和实时系统的特点。

8. 如何评价操作系统的性能？

第 0 章 进程管理

在多道程序环境下，众多的计算机用户都以各种各样的任务，随时随地争夺使用处理机，使程序具有了并行、制约和动态的特征，程序概念已刻画不清系统的这种并行情况，反映不了它们的活动规律和状态变化。为了动态地看待操作系统，需要以进程作为资源分配和独立运行的基本单位，从进程观点来研究操作系统，并且操作系统所具有的所有特征也都是基于进程而形成的。显然，在操作系统中，进程是一个最基本、最重要的概念。

本章主要介绍进程的基本概念，进程状态及转换，进程控制，进程通信，线程的基本概念，最后对 进程结构进行分析。

进程概念

进程的引入

前趋图

程序是完成某个功能的指令集合，一个程序通常由若干个程序段组成，为了描述一个程序的各部分（程序段或语句）间的依赖关系，或者是一个大的计算的各个子任务间的因果关系，我们常常采用前趋图方式。

前趋图是一个有向无循环图，图中的每个结点可以表示一条语句、一个程序段或一个进程，结点间的有向边表示两个结点之间存在的偏序或前趋关系“→”：

前趋→越{ (P₁, P₂) | P₂必须在 P₁开始执行之前完成}

如果 (P₁, P₂) ∈ →，可写成 P₁→P₂，则称 P₁是 P₂的直接前趋，P₂是 P₁的直接后继。若存在一个序列 P₁→P₂→...→P_n，则称 P₁是 P_n的前趋。在前趋图中，没有前趋的结点称为初始结点，没有后继的结点称为终止结点。图 0-1 所示的前趋图，存在下面的前趋关系：

前趋前趋前趋前趋→前趋，前趋→前趋，前趋→前趋，前趋→前趋，前趋→前趋，前趋→前趋，前趋→前趋，前趋→前趋，前趋→前趋，前趋→前趋
或表示为：

前趋前趋前趋前趋越{ (P₁, P₂), (P₁, P₃), (P₂, P₄), (P₃, P₄), (P₄, P₅), (P₄, P₆), (P₅, P₇), (P₆, P₇) }
前趋前趋前趋→越{ (P₁, P₂), (P₁, P₃), (P₂, P₄), (P₃, P₄), (P₄, P₅), (P₄, P₆), (P₅, P₇), (P₆, P₇) }

应当注意，前趋图中必须不存在循环。

程序的顺序执行

程序的顺序执行是指各程序段按照某种先后次序执行，仅当前一个操作执行完后才能执

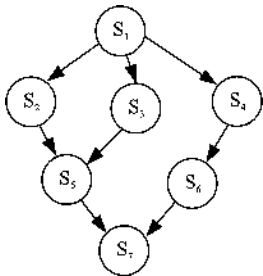


图 0-1 前趋图

行后继操作。例如，在处理一个作业时，总是先输入用户的程序和数据，然后进行计算，最后将所得的结果打印出来。图 4-1 所示为程序在单道批处理系统中顺序执行时的前驱图。其中结点代表程序段的操作，输入、计算和打印分别表示输入、计算和打印操作。

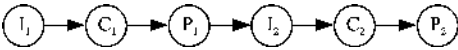


图 4-1 程序顺序执行时的前驱图

程序顺序执行时有如下特征：

(1) 顺序性。处理机的操作严格按照程序所规定的顺序执行，即每一个操作必须在下一个操作开始之前结束。

(2) 封闭性。程序一旦开始运行，其执行结果不受外界因素影响。因为程序在运行时独占系统的各种资源，故这些资源的状态（除初始状态外）只有本程序中才能改变。

(3) 可再现性。只要程序执行时的初始条件和执行环境相同，当程序多次重复执行时，不论它是从头到尾不停顿地执行，还是“停停走走”地执行，都将获得相同的结果（即程序的执行结果与时间无关）。

程序的并发执行

程序的并发执行是指若干个程序（或程序段）同时在系统中运行，这些程序（或程序段）的执行在时间上是重叠的，一个程序（或程序段）的执行尚未结束，另一个程序（或程序段）的执行已经开始。

在图 4-1 中的输入程序、计算程序和打印程序三者之间，存在着输入→计算→打印这样的前趋关系，以致对一个作业的输入、计算和打印三个操作，必须顺序执行，但并不存在打印→输入关系，因而在对一批程序进行处理时，可使它们并发执行。例如，输入程序在输入第一个程序后，在计算程序对该程序进行计算的同时，可由输入程序再输入第二个程序，从而使第一个程序的

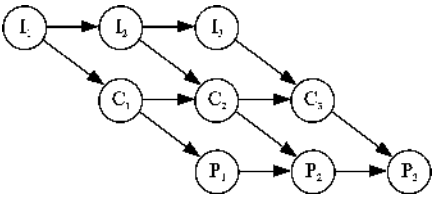


图 4-2 程序并发执行时的前驱图

计算操作与第二个程序的输入操作并发执行。一般说，输入程序在输入第 n 个程序时，计算程序可能正在对第 $n-1$ 个程序进行计算，而打印程序正在打印第 $n-2$ 个程序的计算结果。图 4-2 表示出了输入、计算、打印三个程序对一批作业进行处理的情况。

在该例中存在下述的前趋关系：

输入→计算，计算→打印，输入→计算，计算→打印，输入→计算，计算→打印

而输入和计算及计算和打印是重叠的。亦即输入和计算以及计算和打印可以并发执行。

设执行一个作业需要三分钟，若顺序执行三个作业共需要 9 分钟，而并行执行三个作业只需要 3 分钟，多道程序的并发执行大大提高了系统的处理能力和资源利用率。

程序的并发执行也带来了一些新问题，产生了一些与顺序执行时不同的特征：

(1) 间断性。程序在并发执行时，由于它们共享资源或为完成同一项任务而相互合作，致使并发程序之间形成了相互制约关系。例如在图 4-2 中，若输入未完成则不能进行计算，致使作

作业的打印操作暂停运行,这是由相互合作完成同一项任务而产生的直接制约关系;若输入未完成则不能进行输入,致使作业的输入操作暂停运行,这是由共享资源而产生的间接制约关系。这种相互制约关系将导致并发程序具有“执行—暂停—执行”这种间断性的活动规律。

(3) 失去封闭性。程序在并发执行时,多个程序共享系统中的各种资源,因而这些资源的状态将由多个程序来改变,致使程序的运行失去封闭性。这样一个程序在执行时,必然会受到其他程序的影响。例如,当处理机被某程序占用时,其他程序必须等待。

(4) 不可再现性。程序并发执行时,由于失去了封闭性,也将导致失去其运行结果的可再现性。例如,有两个循环程序 A 和 B,它们共享一个变量 x。程序 A 每执行一次,都要做 x=x+1 操作;程序 B 每执行一次,都要执行 x=x-1 操作。程序 A 和 B 可能以不同的速度运行,这样可能出现下述两种不同情况(假设当前变量 x 的值为 10,而得到不同的结果。

① 程序 A 的 x=x+1 在程序 B 的 x=x-1 之前,此时程序 B 打印 x 的值为 11。

② 程序 A 的 x=x+1 在程序 B 的 x=x-1 之后,此时程序 B 打印 x 的值为 9。

上述情况说明,程序在并发执行时,由于失去了封闭性,其计算结果已与并发程序的执行速度有关,从而使程序失去了可再现性,即程序经过多次执行后,虽然其执行时的环境和初始条件都相同,但得到的结果却各不相同。

可见,在多道程序并发执行环境下,程序具有了并行、制约和动态的特征,程序和计算不再一一对应,程序这个静态概念已经不能如实地反映程序活动的这些特征,例如:两个源程序,由一个编译程序完成,若用程序概念理解,就会以为主存中只有一个编译程序运行(两个源程序只是编译程序加工的数据),而无法说清主存中运行着两个任务。也就是说程序概念刻画不清这种并行情况,反映不了它们的活动规律和状态变化。为此,人们引入一个新的概念——进程。

4.1 进程的定义与特征

4.1.1 进程的定义

进程这个名词最早是 1964 年在麻省理工学院 (MIT) 的 CTSS 系统和 IBM 公司的 CTSS 系统中提出的,直到目前对进程的定义和名称均不统一,不同的系统中采用不同的术语名称,例如 CTSS 称进程 (process),IBM 公司称任务 (task),SOS 公司称活动 (activity)。

有许多人从不同角度对“进程”下过各种定义。其中较能反映进程实质的定义有:

(1) 进程是程序在处理机上的一次执行过程;

(2) 进程是可以和别的计算并行执行的计算;

(3) 进程可定义为一个数据结构及能在其上进行操作的一个程序;

(4) 进程是一个程序及其数据在处理机上顺序执行时所发生的活动;

(5) 进程是程序在一个数据集合上的运行过程,是系统进行资源分配和调度的一个独立单位。

国内学术界较为一致的定义是:

进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动。

4.1.2 进程的特征

进程具有以下基本特征:

(1) 动态性。进程是程序在处理机上的一次执行过程,因而是动态的。动态特性还表现在它由创建而产生,由调度而执行,因得不到资源而暂停执行,最后由撤销而消亡。进程有

一定的生命周期。

④ 并发性。指多个进程实体同时存在于内存中，能在一段时间内同时运行。

⑤ 独立性。进程是一个能独立运行的基本单位，也是系统进行资源分配和调度的基本单位。

⑥ 制约性。进程间由于对资源的争用而相互制约。

⑦ 异步性。进程按各自独立的、不可预知的速度向前推进。

⑧ 结构性。进程包含了数据集合和其上运行的程序。为了描述和记录进程的动态变化过程，还需要配置一个进程控制块，所以每个进程由三要素组成：程序、数据和进程控制块。

进程与程序的关系

进程和程序是两个密切相关但又有所不同的概念，它们在以下几个方面存在区别和联系：

① 进程是动态的，程序是静态的。进程是程序的执行，程序是有序代码的集合。

② 进程是暂时的，程序是永久的。进程的生命是有限的，而程序可以长久保存。

③ 进程与程序的组成不同：进程的组成包括程序、数据和进程控制块。

④ 进程具有并发特征，而程序没有。

⑤ 进程具有创建其他进程的功能，而程序不能创建新的程序。

⑥ 进程与程序是密切相关的。通过多次执行，一个程序可以对应多个进程；通过调用关系，一个进程可以包括多个程序。

进程的类型

系统中同时存在许多进程，它们依性质不同可分为各种不同的类别。

① 系统进程和用户进程

一般来讲，系统进程主要执行操作系统程序，提供系统功能，如：作业调度、各种资源的管理和控制等。用户进程执行用户程序，提供用户功能。

为了防止操作系统及其关键数据（如 缓冲等）受到用户程序有意或无意的破坏，通常将处理机的执行状态分成两种：核心态与用户态。

● 核心态又称管态、系统态，是操作系统管理程序执行时机器所处的状态。它具有较高的特权，能执行一切指令，访问所有的寄存器和存储区。

● 用户态又称目态，是用户程序执行时机器所处的状态。这是具有较低特权的执行状态，它只能执行规定的指令，访问指定的寄存器和存储区。

通常，用户程序运行在用户态，不能执行操作系统程序和访问操作系统区域，这样就防止了用户程序对操作系统的破坏。

② 父进程和子进程

系统或用户首先创建的进程称为父进程；由父进程创建的进程称为子进程。父子进程间存在着某些控制结构和控制关系，由此，我们先定义一个进程图，如图 图原所示。

进程图是有向的，包含一个根节点的树，节点表示进程，记为 P_i 从节点 P_i 到节点 P_j 的一条边表示进程 P_j 是由进程 P_i 创建的。其中，称 P_i 是 P_j 的父进程，而 P_j 则是 P_i 的子进程。进程图表示了进程之间的家族关系。

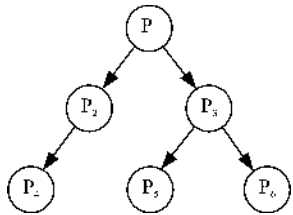


图 图原进程图

父、子进程间的主要关系：

- ① 进程控制。父进程可以控制子进程。
- ② 运行方式。父进程一旦创建子进程后，可以选择父、子进程同时运行的方式；父进程也可以等待子进程的运行，直至全部子进程结束后，才开始重新运行。
- ③ 资源共享。可以选择两种不同的资源共享方式：一是子进程可共享父进程所拥有的全部资源；二是子进程仅能共享父进程的部分资源。

4.1 进程描述

4.1.1 进程的结构

（1）进程映像

当一个程序进入计算机主存运行时就构成了进程，主存中的进程到底是如何组成的呢？进程是程序关于数据的一次执行，因而它包含有一个或多个程序，以及数据。这些程序和数据要有充分的主存来分配给进程，装入它们。程序运行时通常还需要有一个堆栈来保存过程调用现场信息和过程间传递的参数。除此之外，进程描述还应包含一些进程的有关情况的属性信息，例如执行情况、资源使用情况。为了便于操作系统对进程进行控制管理，通常操作系统用一个称为进程控制块（PCB）的数据结构来记录进程的属性信息。所以进程的静态组成包括程序的程序块、数据块、系统用户栈和进程控制块，统称为进程映像。

（2）进程上下文

进程是在操作系统支持下运行的，进程运行时操作系统需要为其设置相应的运行环境。进程物理实体和支持进程运行的环境合称为进程上下文。当系统调度新进程占有处理器时，新老进程随之发生上下文切换。进程上下文分成三个部分：用户级上下文、寄存器级上下文和系统级上下文。

- 用户级上下文：是指进程的用户地址空间，包括用户正文段、用户数据段和用户栈。
- 寄存器级上下文：是指程序计数器、程序状态字寄存器、栈指针、通用寄存器的值等。
- 系统级上下文：是指进程的系统地址空间，包括进程控制块、内存管理信息、进程环境块和系统栈等。

（3）正文段：程序段经过编译后形成的机器指令代码集。

4.1.2 进程控制块

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，操作系统为每个进程定义了一个数据结构——进程控制块（PCB，Process Control Block）。

PCB是标志进程存在的数据结构，包含操作系统所需的、用于描述进程情况及控制进程运行所需的全部信息。

系统根据 PCB 而感知进程的存在，PCB 是进程存在的惟一标志。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。PCB 可以被操作系统中的多个模块读或修改，如被调度程序、资源分配程序、中断处理程序以及监督和分析程序等读或修改。因为 PCB 经常被系统访问，尤其是被运行频率很高的进程调度及分派程序访问，故 PCB 应常驻内存。系统将所

有的进程组织成若干个链表(或队列),存放在操作系统中专门开辟的进程区内。

一般来说,根据操作系统的要求不同,进程所包含的内容多少会有些不同,但通常包括下面所列出的内容:

(一) 进程标识符信息

进程标识符用于唯一地标识一个进程。一个进程通常有以下两种标识符:

1. 外部标识符。由创建者提供,通常是由字母、数字所组成,往往是由用户(进程)在访问该进程时使用。外部标识符便于记忆,如计算进程、打印进程、发送进程、接收进程等。

2. 内部标识符。这是为了方便系统使用而设置的。在所有操作系统中,都为每一个进程赋予一个惟一的整数,作为内部标识符。它通常就是一个进程的序号。为了描述进程的家族关系,还应设置父进程标识符及子进程标识符。此外,还可设置用户标识符,以指示他就是拥有该进程的用户。

(二) 处理机状态信息

处理机状态信息主要是由处理机各种寄存器中的内容组成。处理机在运行时,许多信息都放在寄存器中。当处理机被中断时,所有这些都必须保存在进程中,以便在该进程重新执行时,能从断点继续执行。这组寄存器包括:

1. 通用寄存器。又称为用户可视寄存器,它们是用用户程序可以访问的,用于暂存信息,在大多数处理机中有 8~16 个通用寄存器。在 8086 结构的计算机中,可超过 16 个。

2. 指令计数器。其中存放了要访问的下一条指令的地址。

3. 程序状态字。其中含有状态信息,如条件码、执行方式、中断屏蔽标志等。

4. 用户栈指针。每个用户进程有一个或若干个与之相关的系统栈,用于存放过程和系统调用参数及调用地址。栈指针指向该栈的栈顶。

(三) 进程调度信息

在进程中还存放一些与进程调度和进程对换有关的信息,包括:

1. 进程状态。指明进程的当前状态,作为进程调度和对换时的依据。

2. 进程优先级。是用于描述进程使用处理机的优先级别的一个整数,优先级高的进程应优先获得处理机。

3. 进程调度所需的其他信息。它们与所采用的进程调度算法有关,比如,进程已等待就绪的时间总和、进程已执行的时间总和等。

4. 事件。是指进程由执行状态转变为阻塞状态所等待发生的事件,即阻塞原因。

(四) 进程控制信息

进程控制信息包括:

1. 程序和数据地址。是指进程程序和数据所在的内存或外存地址,以便再调度到该进程执行时,能从进程中找到其程序和数据。

2. 进程同步和通信机制。指实现进程同步和进程通信时必须的机制,如消息队列指针、信号量等,它们可能全部或部分地放在进程中。

3. 资源清单。是一张列出了除就绪以外的、进程所需的全部资源及已经分配到该进程的资源清单。

4. 链接指针。它给出了本进程所在队列中的下一个进程的进程首地址,系统中的进程可能组织成多个队列,如就绪队列、阻塞队列等。

进程控制块的组织形式

在一个系统中，通常存在着许多进程，它们有的处于就绪状态，有的处于阻塞状态，而且阻塞的原因各不相同，为了方便进程的调度和管理，需要将各进程的进程控制块用适当的方法组织起来，统称为 **就绪表**。就绪表是系统最常用的数据结构，它的物理组织方式也直接影响系统的效率。目前常用的组织方式有链接方式和索引方式两种。

(1) 链接方式

将同一状态的进程控制块链接成一个队列，不同状态对应多个不同的队列，如就绪队列、运行队列（多机系统中）和阻塞队列等，如图 4-1 所示。

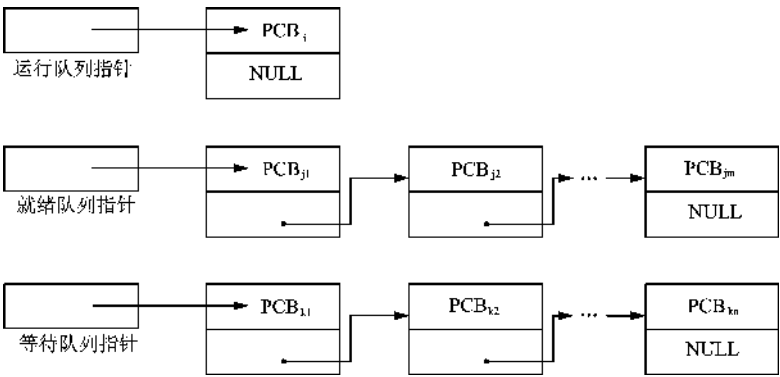


图 4-1 就绪队列链接组织方式示意图

有的系统把所有不同状态进程的 **就绪表** 组织在一个队列中。这种方法最为简单，适用于系统中进程数目不多的类型，如 **批处理系统**。其缺点是调度进程时，往往要查找整个 **就绪表**。

(2) 索引方式

将同一状态的进程组织在一个索引表中，索引表的表项指向相应的进程控制块，不同状态对应不同的索引表，如就绪索引表和阻塞索引表等，如图 4-2 所示。

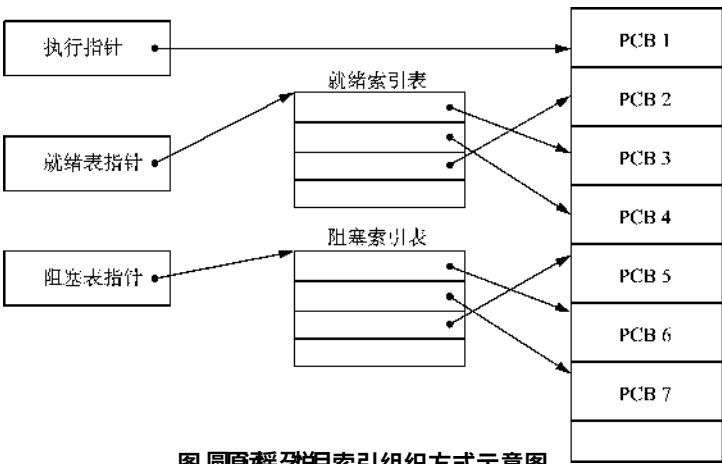


图 4-2 就绪索引组织方式示意图

进程状态及转换

三态模型

在进程的运行过程中，由于系统中多个进程的并发运行及相互制约的结果，使得进程的状态不断发生变化。通常，一个进程至少可划分为三种基本状态：

就绪（Ready）状态。进程已获得了除处理机以外的所有资源，一旦获得处理机就可以立即执行，此时进程所处的状态为就绪态。

运行（Running）状态。当一个进程获得必要的资源并正在处理机上运行时，该进程所处的状态为运行态。

阻塞（Blocked）状态。阻塞态又称等待（Waiting）状态或睡眠（Sleeping）状态。正在运行的进程，由于发生某事件而暂时无法运行下去（如等待输入输出完成），此时进程所处的状态为阻塞态。当进程处于阻塞状态时，即使把处理机分配给该进程，它也无法运行。

进程并非固定处于某一个状态，其状态会随着自身的推进和外界条件的变化而发生变化。通常可以用一个进程状态变化图来说明系统中每个进程可能具备的状态，以及这些状态发生变化的可能原因，如图 4-1 所示。

下面简要阐述进程状态的转换及原因。

就绪—运行。处于就绪态的进程，当进程调度程序为之分配了处理机后，进程便由就绪态转变为运行态。

运行—就绪。正在运行的进程，如因时间片用完而被暂停执行，该进程便由运行态转变为就绪态。进程还有以下可能出现这种状态的改变：

① 如果采用可剥夺调度算法，当有更高优先级的进程就绪时，操作系统调度程序可以将正在运行的进程从运行状态改变为就绪状态，让更高优先级的进程运行。

② 进程自动放弃运行状态，如当进程申请某一系统资源，但系统资源暂时没有，进程中的程序可安排一个循环申请资源，当资源没有得到时产生“放弃处理机”系统调用，这时处理机会切换到其他进程运行，原进程在“放弃处理机”后再申请，获得资源时转出循环。

运行—阻塞。正在运行的进程因等待某事件而无法继续运行时，则由运行态转变为阻塞态。例如，进程请求一个输入输出操作，操作系统已启动外设，但输入输出尚未完成；或进程要与其他进程通信，要接收信件，但对方还未发出时进程都会被阻塞。

阻塞—就绪。处于阻塞态的进程，当其等待的事件已经发生时，则由阻塞态转变为就绪态。如输入输出操作完成。

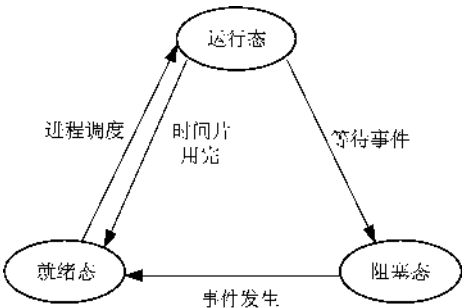


图 4-1 具有三种状态的进程状态图

图 04-15 进程的 5 态模型

在不少系统中，除了上述三种基本状态之外，又增加了两种状态（见图 04-15）：

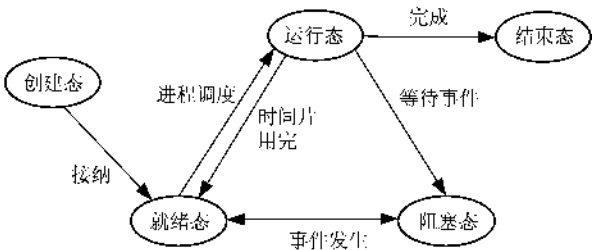


图 04-15 进程的 5 态模型

（1）创建态：进程正在被创建，还没转到就绪状态之前的状态。

（2）结束态：进程已结束运行，释放了除进程控制块之外的其他资源，此时进程所处的状态为结束状态。

在进程管理中，创建态和结束态是非常有用的。引入创建态的原因，是由于操作系统在建立一个新进程时，通常分为两步：第一步是为新创建的进程分配进程控制块结构，设置进程标识等参数，并为它分配资源，此时进程即处于创建态；第二步是把新创建的进程送入就绪队列，一旦进程进入就绪队列，它便由创建状态转变为就绪状态。

类似地，一个已经结束了的进程，其退出系统的过程也分为两步：第一步是将其从就绪队列中移出，使之成为一个不可能再运行的进程，相应的进程处于结束状态。此时系统并不立即撤销它，而是将它暂时留在系统中，以便其他进程去收集该进程的有用信息。例如，由记账进程去了解该进程用了多少时间，使用了哪些类型的资源，以便记账。又如，统计进程可对该进程的类型、运行情况和时间等信息进行收集。然后逐步释放资源，最后释放进程控制块，并将其占用的存储空间返还系统，之后操作系统删除该进程。引起进程状态转换的具体原因如下：

（1）空—创建态：执行一个程序，创建一个子进程。

（2）创建态—就绪态：当进程被创建完成，初始化后，一切就绪，准备运行时变到就绪态。有些操作系统为了使系统资源不过分散到各个进程，它通常限制从创建状态进入就绪状态的进程数，这样做可以使系统内存、内核表格空间等系统资源集中给有限的进程使用。因此可能进程进入创建状态，但很长时间不能进入就绪队列，等操作系统把它选中时才可以分配好所有资源，变为就绪状态。

（3）运行态—结束态：当一个进程到达了自然结束点，或是出现了无法克服的错误，或是被操作系统所终结，或是被其他有终止权的进程所终结，即由运行态变为结束态。

（4）就绪态—结束态：这种状态变化没有在状态变化图中列出，但在有些系统中可以支持父进程终止没有运行的子进程，或特权进程终止任何其他进程。

（5）结束态—空：完成善后操作。

图猿园摇具有挂起功能的进程状态及其转换

在某些系统中,为了更好地管理和调度进程及适应系统的功能目标,引入了挂起状态。引入挂起状态可能基于下述原因:

(员) 系统有时可能出故障或某些功能受到破坏。这时就需要暂时将系统中的进程挂起,以便系统把故障消除后,再把这些进程恢复到原来状态。

(圆) 用户检查自己作业的中间执行情况和中间结果时,因同预期想法不符而产生怀疑,这时用户要求挂起他的进程,以便进行某些检查和改正。

(猿) 系统中有时负荷过重(进程数过多),资源数相对不足,从而造成系统效率下降。此时需要挂起一部分进程以调整系统负荷,等系统中负荷减轻后再将被挂起进程恢复运行。

(源) 一个进程可能是周期性的执行,在等待下一次执行周期时挂起,以减轻系统负荷。

处于挂起状态的进程被交换到外存,不占用任何内存。

在具有挂起和解挂功能的系统进程状态中,新增加了两个状态:挂起就绪(就绪态的挂起态)和挂起阻塞(阻塞态的挂起态)。为了易于区分,将原来的就绪状态和阻塞状态分别称为活动就绪(就绪态)和活动阻塞(阻塞态)。图 猿园给出了具有挂起功能的进程状态图。

从图 猿园可以看出,如果一个进程原来处于运行状态或活动就绪状态,此时可因挂起命令而由原来状态变为挂起就绪状态,处于挂起就绪状态的进程不能参与争夺处理机,即进程调度程序不会把处于挂起就绪状态的进程挑选来运行;当处于挂起就绪状态的进程接到激活命令后,它就由原状态变为活动就绪状态;如果一个进程原来处于活动阻塞状态,它可因挂起命令而变为挂起阻塞状态,直到激活命令才能把它重新变为活动阻塞状态;处于挂起阻塞状态的进程,其所等待的事件发生后,该进程就由原来的挂起阻塞状态变为挂起就绪状态。

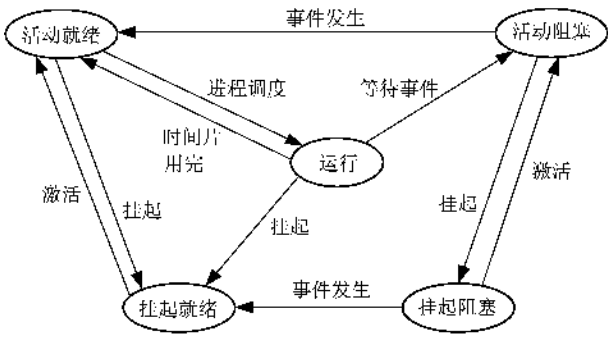


图 猿园摇具有挂起功能的进程状态

图猿员摇进程控制

进程控制的职责是对系统中的所有进程实施有效的管理,其功能包括进程的创建、进程的撤销、进程的阻塞与唤醒、进程的挂起与激活等。

进程控制功能是由操作系统的内核来实现的。内核是操作系统中最重要最核心的部分,通常是将运行频率较高的模块,如进程调度、时钟管理,以及一些与硬件紧密相关的模块,如中断处理程序、设备驱动程序,以及许多模块公用的一些基本操作,都安排在紧靠硬件的层次中,以便提高操作系统的运行效率,并对它实施特殊的保护。因此内核是加到硬件上的第一层软件,常驻内存。

进程控制功能是通过执行各种原语来实现的。原语是机器指令的延伸,是用若干条机器指令构成的,用以完成特定功能的一段程序。为保证操作的正确性,原语在执行期间是不可分割的。这就是说,原语的执行不能被中断,所以原语操作具有原子性。

4.1.2 进程的创建

4.1.2.1 引起创建进程的事件

(1) 用户登录。在分时系统中,用户通过终端进行登录时,系统将为该终端用户建立一个进程,并把它插入就绪队列。

(2) 作业调度。在批处理系统中,当作业调度程序调度到某个作业时,便将该作业装入内存,并立即为它创建进程并插入就绪队列。

(3) 提供服务。当运行中的用户程序提出某种请求后,系统将专门创建一个进程来提供用户所需要的服务。例如,如果用户程序要求进行文件打印,操作系统将为之创建一个打印进程。这样,不仅可使打印进程与该用户进程并发执行,而且还便于计算为完成打印任务所花费的时间。

(4) 应用请求。在上述三种情况下,都是由系统内核为之创建一个新进程。而第四类事件则是基于应用进程的需要,由它自己创建一个新进程,以便使新进程以并发运行方式完成特定任务。例如,某应用程序需要不断地从键盘终端读入输入数据,继而又要对输入数据进行相应的处理,然后,又将处理结果以表格形式在屏幕上显示。该应用进程为使这几个操作能并发执行,以加速任务的完成,可以分别建立键盘输入进程、表格输出进程。

4.1.2.2 进程创建过程

一个进程可以调用创建进程原语来创建一个子进程。该原语的主要工作是为被建立进程建立起一个进程控制块 PCB,并填入相应的初始值。其主要操作过程是:

(1) 先向系统申请一个空闲的 PCB。

(2) 为新进程分配必要的资源,如内存空间。

(3) 初始化进程控制块。

PCB 的初始化工作包括:

- 初始化标识符信息。为新进程分配一个惟一的进程标识符,将系统分配的标识符、父进程标识符等填入新 PCB 中。

- 初始化处理机状态信息。使程序计数器指向程序的入口地址,使栈指针指向栈顶。

- 初始化处理机控制信息。将进程的状态设置为就绪状态或挂起就绪状态;对于优先级,通常是将其设置为最低优先级,除非用户以显式方式提出高优先级要求。

(4) 将新进程插入就绪队列。

如果进程就绪队列能够接纳新进程,便将新进程插入就绪队列。

4.1.3 进程的撤销

4.1.3.1 引起进程撤销的事件

以下几种情况导致进程撤销:

(1) 进程已完成所要求的功能而正常终止。

(2) 由于某种错误导致非正常终止。

在进程运行期间，由于出现某些错误和故障而迫使进程终止。这类异常事件很多，常见的有：

- ① 越界错误。它是指程序所访问的存储区已越出该进程的区域。
 - ② 保护错。进程试图去访问一个未被允许访问的资源或文件；或者以不适当的方式进行访问。例如，试图去写一个只读文件。
 - ③ 特权指令错。用户进程试图去执行一条只允许操作系统执行的特权指令。
 - ④ 非法指令错。进程试图去执行一条不存在的指令。出现该错误的原因，可能是程序错误地转移到数据区，把数据当成了指令。
 - ⑤ 运行超时。进程的执行时间超过了指定的最大值。
 - ⑥ 等待超时。进程等待某事件的时间，超过了规定的最大值。
 - ⑦ 算术运算错。进程试图去执行一个被禁止的计算。例如，被 0 除。
 - ⑧ 随机故障。在随机过程中发生了错误等。
- (猿 外界干预。

外界干预并非指在本进程运行中出现了异常事件，而是进程应外界的请求而终止运行，这些干预有：

● 操作员或操作系统干预。由于某种原因，例如，发生了死锁，由操作员或操作系统终止该进程。

● 父进程请求。由于父进程具有终止自己的任何子孙进程的权力，因而当父进程提出终止子进程请求时，系统将终止该进程。

● 父进程终止。当父进程终止时，操作系统也将它的所有子孙进程终止。

进程撤销的过程

无论哪一种情况导致进程被撤销，进程都必须释放它所占用的各种资源和 数据结构本身，以利于资源的有效利用。撤销原语可采用两种撤销策略：一种策略是只撤销一个具有指定标识符的进程，另一种策略是撤销指定进程及其所有子孙进程。

撤销原语的主要操作过程是：

● 根据被终止进程的标识符，从 进程集合中检索出该进程的 进程，从中读出该进程的状态。

● 若被终止进程正处于执行状态，应立即中止该进程的执行，并设置调度标志为真，用于指示该进程被终止后应重新进行调度，选择一新进程，把处理机分配给它。

● 对后一种撤销策略，若被撤销进程有子孙进程，还应将其所有子孙进程予以终止，以防它们成为不可控的。

● 将该进程所拥有的全部资源，或者归还给其父进程，或者归还给系统。

● 将被终止进程（它的 进程 从所在队列（或链表）中移出，等待其他程序来搜集信息。

进程阻塞与唤醒

引起进程阻塞和唤醒的条件

(员 请求系统服务

当正在执行的进程请求操作系统提供服务时，由于某种原因，操作系统并不立即满足该进程的要求时，该进程只能转变为阻塞状态来等待。例如，一进程请求使用打印机，但系统

已将它分配给其他进程而不能分配给请求进程，故请求者只能被阻塞，仅在其他进程释放出打印机的同时，再由释放者将请求者唤醒。

（四）启动某种操作

当进程启动某种操作后，如果该进程必须在该操作完成之后才能继续执行，则必须先使进程阻塞。例如，进程启动了某个 I/O 设备，如果只有在 I/O 设备完成了指定 I/O 任务后，进程才能继续执行，则进程在启动了 I/O 操作后，便自动进入阻塞状态去等待。在 I/O 操作完成后，由中断处理程序或中断进程将该进程唤醒。

（五）新数据尚未到达

对于相互合作的进程，如果其中一个进程需要先获得另一进程提供的数据后才能运行（对数据进行处理），则只要其所需数据尚未到达，进程只有进入阻塞状态等待。例如，有两个进程，进程 A 用于输入数据，进程 B 对输入数据进行加工，假如进程 A 尚未将数据输入完毕，则进程 B 将因无所需的处理数据而阻塞；一旦进程 A 把数据输入完后，便可唤醒进程 B。

（六）无新工作可做

系统往往设置一些具有某特定功能的系统进程，每当这种进程完成任务后，便把自己阻塞起来等待新任务的到来。例如，系统中的发送进程，其主要工作是发送数据，若已有的数据已全部发送完成而又无新的发送请求，进程将使自己进入阻塞状态；仅当又有进程提出新的发送请求时，才又将发送进程唤醒。

（七）进程阻塞过程

正在执行的进程，当出现上述某个事件时，由于无法继续执行，于是进程便通过调用阻塞原语把自己阻塞起来。可见，进程的阻塞是进程自身的一种主动行为。进入阻塞过程后，由于此时该进程还处于执行状态，所以应先立即停止执行，把进程控制块中的现行状态由“执行”改为“阻塞”，并将它插入到阻塞队列。然后，转调度程序进行重新调度，将处理机分配给另一就绪进程，并进行切换。亦即保留被阻塞进程的处理机状态（在 PCB 中），再按新进程 PCB 中的处理机状态设置就绪环境。

（八）进程唤醒过程

当等待队列中的进程所等待的事件发生时，等待该事件的所有进程都将被唤醒。显然，一个处于阻塞状态的进程不可能自己唤醒自己。唤醒一个进程有两种方法：一种是由系统进程唤醒，另一种是由事件发生进程唤醒。当由系统进程唤醒等待进程时，系统进程统一控制事件的发生并将“事件发生”这一消息通知等待进程，从而使得该进程因等待事件发生而进入就绪队列。等待进程也可由事件发生进程唤醒。由事件发生进程唤醒时，事件发生进程和被唤醒进程之间是合作关系。因此，唤醒原语既可被系统进程调用，也可被事件发生进程调用，称调用唤醒原语的进程为唤醒进程。

唤醒原语的主要操作如下：将被唤醒进程从相应的等待列中移出，将其状态改为就绪并插入相应的就绪队列。在把被唤醒进程送入就绪队列之后，唤醒原语既可以返回原调用程序，也可以转向进程调度。

应当指出，阻塞原语和唤醒原语是一对作用刚好相反的原语。因此，如果在某进程中调用了阻塞原语，则必须在与之相合作的另一进程或其他相关进程中，调用唤醒原语来唤醒阻塞进程，否则，该阻塞进程将会因不能被唤醒而长久地处于阻塞状态，从而再无机会运行。

圆猿猿猿进程的挂起与激活

当需要挂起某个进程时可调用挂起原语。应注意,调用挂起原语的进程只能挂起它自己或它的子孙。挂起的实现方式有多种:

(员) 把发出挂起原语的进程自身挂起。

(圆) 挂起具有指定标识名的进程。

(猿) 把某进程及其全部或部分(例如具有指定优先数的)子孙进程挂起。

现以挂起具有指定标识名的进程为例,说明挂起原语的主要操作过程:以被挂起的进程标识名为索引,到 进程集合 中查找该进程的 进程,得到该进程的内部标识,并检查该进程的状态。若状态为执行,则中断处理机,把 当前状态 保存在 进程中,停止运行该进程并将状态改为挂起就绪;若状态为阻塞,则改为挂起阻塞;若状态为就绪,则改为挂起就绪;若状态为执行,则从就绪队列中按某种算法选择一个进程投入运行。被挂起进程 进程的非常驻部分要交换到磁盘交换区。

激活原语使处于挂起状态的进程变成活动,即把挂起就绪状态变成活动就绪状态,把挂起阻塞状态变成活动阻塞状态。一旦被激活的进程处于活动就绪状态,由于其优先数可能已发生改变,便可能引起处理机的重新调度。同样,激活原语也可以有多种激活方式:如激活一个具有指定标识名的进程,或者激活某进程及其所有的子孙进程。

注意,挂起原语既可由进程自己也可由其他进程调用,但激活原语却只能由其他进程调用。

圆猿猿进程的切换与模式切换

圆猿猿猿执行模式

处理机在执行用户程序和执行操作系统程序时的模式是有区别的,这是因为处理机在运行系统程序时可以获得更多的特权,以便操作系统程序实现更强的功能;而在运行用户程序的时候,仅有有限的权限,用来保障操作系统的安全性。

许多处理机支持至少两种执行模式:特权模式和用户模式。特权模式又称核心模式、系统模式或控制模式,操作系统核心程序在这种模式下运行。某些指令只能在特权模式下执行,如读写处理机状态字 状态字 等控制寄存器以及存储管理相关的一些指令;另外在特权模式下,程序还可以访问更大的地址空间。用户模式就是非特权的模式,用户程序在这个模式下运行。当然,现代操作系统有许多系统功能也由运行在用户模式的程序实现。如 操作系统 的 员号进程,它在用户态运行 操作系统程序,负责所有用户终端进程的创建。

划分两种模式的理由是为了保护操作系统和操作系统的数据表格不被可能出错的用户程序破坏。处理机如何知道它在哪种模式下?执行模式如何变化、何时变化呢?处理机状态字 状态字 中有一位表示处理机执行模式,这一位是可以通过特殊指令修改的。当运行在处理机上的用户程序产生自陷,或被外部事件中断进入操作系统程序运行时,操作系统程序要用置模式指令马上将处理机模式转变为核心态,原程序状态字作为运行现场被保护,新的程序状态字中的模式被置为核心工作状态。这时处理机执行任何特权指令或访问系统的虚空间都不会

报错，待到返回用户态程序时，原保护的程序状态字被恢复，处理机模式又转到用户态下，这以后如果执行了特权指令，处理机就会报错。

从上面可以看出，进程既为运行用户程序而创建，又会在用户自陷或外部中断来时去运行操作系统内核程序。当进程运行系统内核程序时，系统只是保存了用户程序的运行现场，包括所有处理机状态、现场信息，保留原用户程序用数据栈不被核心态程序使用。当核心程序运行时，系统使用为该进程分配的核心栈空间，这样当核心程序转子或被中断时可以利用核心栈保存现场。

4.1.2 用户态与核心态模式切换

处理机在执行每一条机器指令时，都会查看是否有中断发生。如果没有中断，处理机则继续原来的执行流程。如果有中断发生，处理机按设计时的约定，用中断处理程序的入口地址重置程序计数器，同时保存到处理机状态字中原来的 PC 值，这时处理机转向操作系统的程序执行。毫无疑问，操作系统应马上保护中断点的处理机现场，将处理机模式从用户态置成核心态模式。在处理机控制权转到操作系统核心入口后，核心入口程序应做以下工作：

(1) 保护处理机现场，包括保护包含了原程序计数器值的处理机状态字、各种其他寄存器，其中包含有用户栈的指针。

(2) 将处理机模式转换成核心态，以便以后的程序可以包含特权指令。

(3) 根据中断级别设置中断屏蔽。一般情况下，如果发生了某一级中断，则要屏蔽该级及以下级别的中断，转中断处理程序。

用户程序执行自陷指令进入操作系统程序执行的情形与上述情形基本相似，只是无需设置中断屏蔽，然后操作系统调用服务例程运行。

在操作系统处理中断或系统调用过程中，可能会导致一些阻塞的高优先级进程变成就绪态，如一个阻塞的进程在中断处理完成后会使得原来那个等待的阻塞进程变成就绪。在处理时钟中断的过程中可能发现正运行的进程时间片已用完，或发现到了运行进程交换程序的时候，把一些挂起的进程解挂或把一些就绪或阻塞的进程挂起，这都会引起请求进程切换。当遇到上述情形发生时，操作系统程序立即置好请求进程调度标志，马上进行进程调度并切换，或等到中断或自陷处理完成准备返回恢复点时进行进程调度并切换。当然，如果没有发生上述事情，原进程的用户程序现场还会被恢复运行。

4.1.3 用户态与核心态进程切换

进程切换是指处理机从一个进程的运行转到另一个进程上运行。进程切换与处理机模式变化是不同的，模式变化时，处理机逻辑上还在同一进程中运行。而进程切换指处理机转入另一个进程运行。

如果进程因外部中断或自陷切换到核心态运行，在执行完操作后又恢复用户态刚被中断的程序运行，则操作系统只需恢复进程进入内核时所保存的处理机现场，无需改变当前进程空间等环境信息。但如果要切换进程，那么当前运行进程改变了，当前进程空间等环境信息也要改变。进程切换程序会涉及到如下的过程：

(1) 保存处理机的上下文，包括程序计数器、程序状态字、其他寄存器。

(圆 修改当前运行进程的进程控制块内容,包括将进程状态从运行态改成其他状态,将该进程的进程控制块链接到相应新状态的队列中。

(猿 选择另一个进程执行,这是进程调度所涉及到的内容。

(源 修改被调度进程的进程控制块,包括把其状态改变到运行态。

(缘 将当前进程存储管理数据结构修改为新选进程的存储管理数据结构。例如,由于原来在处理机联想存储器(粤)中的页表项都是原进程页表项的缓冲,其地址变换是原进程的虚实映射,所以要作废掉处理机中联想存储器的各项数据,并将当前进程页表指针改为指向选定的进程页表。

(远 恢复被选进程上次切换出处理机时的处理机现场,按原保护的程序计数器值重置程序计数器,运行新选进程。

圆猿 线程

自从 圆世纪 远年代提出进程概念后,在操作系统中一直都是以进程作为能独立运行的基本单位的。直到 圆世纪 愿年代中期,人们又提出了比进程更小的能独立运行的基本单位——线程(栽),试图用它来提高系统内程序并发执行的程度,从而可进一步提高系统的吞吐量。近几年,线程概念已得到广泛应用,不仅在新推出的操作系统中,大多都已引入了线程概念,而且在新推出的数据库管理系统和其他应用软件中,也都纷纷引入了线程,来改善系统的性能。

圆猿 线程的概念

猿线程的引入

如果说,在操作系统中引入进程的目的是为了使多个程序并发执行,以改善资源利用率及提高系统吞吐量;那么,在操作系统中再引入线程,则是为了减少程序并发执行时所付出的时空开销,使操作系统具有更好的并发性。为了说明这一点,让我们回顾进程的两个基本属性:

(员 进程是一个拥有资源的独立单元。

(圆 进程同时又是一个可以独立调度和分配的基本单位。

上述两个属性构成了程序并发执行的基础。然而为了使进程能并发执行,操作系统还必须进行以下的一系列操作:

① 创建进程。系统在创建一个进程时,必须为之分配其所必需的、除处理机以外的所有资源,如内存空间、设备以及建立相应的

② 撤销进程。系统在撤销进程时,又必须先对这些资源进行回收操作,然后再撤销

③ 进程切换。在对进程进行切换时,由于要保留当前进程的 环境和设置新选中进程的 环境,须花费不少处理机时间。

简言之,由于进程是一个资源拥有者,因而在进程的创建、撤销和切换中,系统必须为之付出较大的时空开销。也正因如此,在系统中所设置的进程数目不宜过多,进程切换的频率也不宜过高,但这也限制了并发程度的进一步提高。

如何能使多个程序更好地并发执行,同时又尽量减少系统的开销,已成为近年来设计操

作系统时所追求的重要目标。于是，有不少研究操作系统的学者们想到，可否将进程的上述两个属性分开，由操作系统分开进行处理，即对作为调度和分派的基本单位，不同时作为独立分配资源的单位，以使之轻装运行；而对拥有资源的基本单位，又不频繁地对之进行切换。正是在这种思想的指导下，产生了线程概念。

1. 线程的定义

线程也有多种定义方法，一般操作系统中将线程定义为：线程是进程内一个相对独立的可以并行执行的单元，是进程内的一个可调度实体。

线程具有许多传统进程所具有的特征，所以又称为轻型进程或进程元。而把传统的进程称为重型进程，它相当只有一个线程的任务。在引入了线程机制的操作系统中，进程内至少包含一个线程，也可以包含多个线程。

2. 线程的结构

(1) 传统进程与多线程进程模型

图 4-1 给出了从进程管理角度看传统进程与多线程进程的内部结构。在传统进程模型中，进程控制块记录进程的所有信息，进程拥有一个虚地址空间，一个用户栈用于执行用户程序，一个核心栈用于执行核心程序。在多线程进程模型中，除了进程控制块和进程虚空间外，每个线程拥有一个线程控制块，每个线程都有一个用户栈和核心栈。

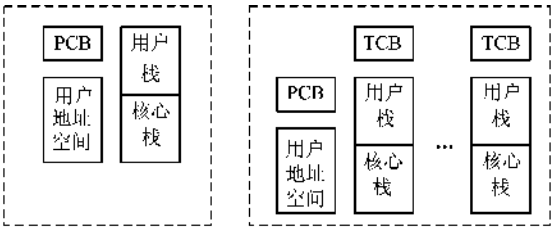


图 4-1 传统进程模型与多线程进程模型

图 4-1 传统进程与多线程进程模型

线程自己基本上不拥有资源，只拥有一点在运行时必不可少的资源（如程序计数器、一组寄存器和栈），一个进程中的所有线程可共享地址空间及系统其他资源的使用。如果一个线程修改了一个数据项，其他线程可以了解和使用此结果数据。一个线程打开并读一个文件时，同一进程中的其他线程也可以同时读此文件。

(2) 线程控制块

线程控制块类似于进程控制块，是标志线程存在的数据结构，其中包含系统对于线程进行管理所需要的全部信息。

不过一般线程控制块中的内容较少，因为有关资源分配等多数信息已经记录于所属进程的进程控制块中。线程控制块中的主要信息如下：

- 线程标识信息。
- 线程运行状态（如运行、就绪等）和调度优先级等调度信息。
- 分别在用户态和核心态下使用的两个栈。用于保存现场信息、子程序局部变量等。
- 一个线程关联的私有存储区。
- 与进程描述信息的连接信息。进程内的每个线程都共享进程描述信息，如地址空间、使用资源等。

3. 线程与进程的比较

下面从几个方面来比较线程与进程。

(1) 调度。在传统的操作系统中，拥有资源和独立调度的基本单位都是进程。而在引入

线程的操作系统中，进程作为资源拥有的基本单位，而线程是调度和分派的基本单位，使传统进程的两个属性分开，线程便能轻装运行，从而可显著提高系统并发执行的程度。在同一进程中，线程的切换不会引起进程切换。在不同进程中进行线程切换，如从一个进程内的线程切换到另一个进程中的线程时，将会引起进程切换。

④ 拥有资源。不论是传统操作系统还是没有线程的操作系统，进程都是拥有资源的基本单位，而线程不拥有系统资源（也有一点必不可少的资源），但线程可以访问其隶属进程拥有的资源。

⑤ 并发性。在引入线程的操作系统中，不仅进程之间可以并发执行，而且同一进程内的多个线程之间也可以并发执行，从而使操作系统具有更好的并发性，大大提高了系统的吞吐量。

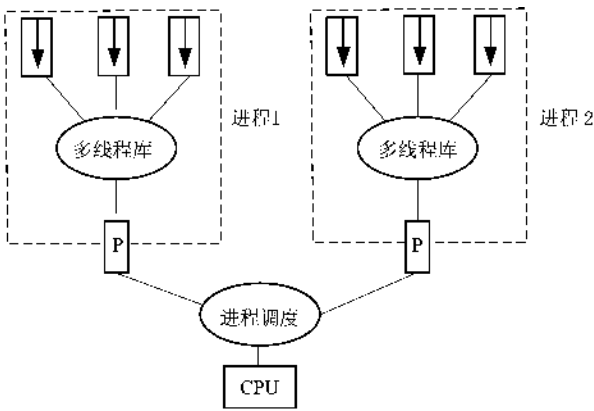
⑥ 系统开销。在创建或撤销进程时，系统要为之分配或回收资源，如内存空间、~~设备~~设备等，操作系统所付出的开销远大于创建或撤销线程时的开销。类似地，在进行进程切换时，涉及到整个当前进程 ~~状态~~环境的保存及新调度到进程的 ~~状态~~环境的设置；而线程切换只需保存和设置少量寄存器内容，因此开销很小。另外，由于同一进程内的多个线程共享进程的地址空间，因此，多线程之间的同步与通信非常容易实现，甚至无须操作系统的干预。

⑦ 线程的实现

线程有两种实现方式：在目态实现的用户级线程和在管态实现的核心级线程，也有一些系统（如 ~~杂~~提供了混合式线程，同时支持两种线程实现方式。

⑧ 用户级线程

没有操作系统内核的支持，完全在用户级提供一个库程序来实现多线程。这些库提供了创建、同步、调度与管理线程的所有功能，无需操作系统特别支持。由于对线程的所有操作都不涉及内核，因此，用户级线程的创建、结束、调度、现场保护与切换开销非常少。图 圆原员给出了用户级线程示意图。与线程相关的控制结构 ~~裁~~保存在目态空间并由运行系统维护，由于线程对操作系统不可见，系统调度仍以进程为单位。



注：→表示线程，下图同

图 圆原员 用户级线程

用户级线程的优点在于：① 线程不依赖于操作系统，可以采用与问题相关的调度策略，灵活性好；② 同一进程中的线程切换不需进入操作系统，因而实现效率较高。用户级线程的缺点在于：① 同一进程中的多个线程不能真正并行；② 由于线程对操作系统不可见，调度在进程级别，若某进程中的一个线程通过系统调用进入操作系统受阻，该进程的其他线程也不能运行。

图 4-1-1 核心级线程

核心级线程是由操作系统支持实现的线程，操作系统维护核心级线程的各种管理表格，负责线程在处理机上的调度和切换，线程是系统调度的基本单位。操作系统提供了一系列系统调用界面，让用户程序请求操作系统进行线程创建、结束、同步等操作，这种核心级线程在许多地方又称为轻量级进程，简称为 LWP (Light Weight Process)。图 4-1-1 给出了核心级线程示意图，图中 LWP 是 LWP 的缩写。

核心级线程的优点是并发性好，在多处理器环境中同一进程中的多个线程可以真正并行执行。缺点是线程控制和状态转换需要进入操作系统完成，系统开销比较大。

图 4-1-2 混合级线程

用户级线程和核心级线程实现方式各有其优缺点，现代操作系统通常都提供上述两种实现方式，这样既利于用户编写并行程序又能够最大限度地发挥多处理机的并行性。

图 4-1-2 给出了混合级线程的示意图，LWP 表示核心级线程，是 LWP 的缩写。

在这种实现系统中，用户利用多线程库提供的并行程序设计界面编写用户并行程序，可以根据用户应用并行度的需要产生足够的用户级线程。核心级线程的操作系统调用界面只提供给多线程库使用，用户可以完全不知多线程库使用了几个核心级线程，也可以通过多线程库的接口申请更多的核心级线程。所有核心级线程都由多线程库管理使用，由多线程库选择用户级线程在哪个核心级线程上运行。

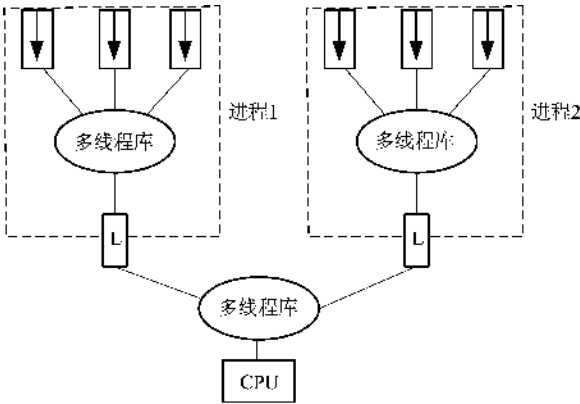


图 4-1-1 核心级线程

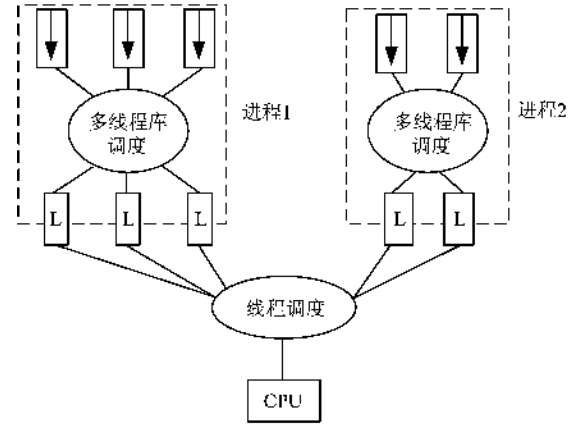


图 4-1-2 混合级线程

图 4-1-3 进程映像

图 4-1-4 进程映像的组成

在操作系统中，进程被描述为映像的执行。而映像 (Image) 则是计算机的执行环境，它包括存储映像、通用寄存器值、打开文件的状态和当前目录等信息。其中存储映像是进程映

像中最为关键的部分，系统 灾中进程的存储映像由下列部分组成：进程控制块（ P_c 区）、正文段、数据段、共享正文段、工作区等几部分组成。

（员 进程控制块 P_c 区：裁为了节省 P_c 区所占主存，将 P_c 区分为两部分：

- 常驻内存的 结构：包含进程调度必须的信息，称为进程基本控制块。
- 不常驻内存的 结构：包含进程运行时才要用到的更多其他信息，它随用户程序和数据部分换进 换出主存，称为进程扩充控制块。 区的字段只能由正在运行的进程存取，即只有占用 的进程才能访问属于当前进程的私有 区。

（圆 正文段：从进程虚地址空间的 单元开始，在执行期间该段受到保护，并且该段的一个单一副本供执行同一程序的所有进程共享。它包括可重入的程序和常数。

（猿 数据段：进程执行时用到的数据，包括外部变量、静态局部变量和常数等。如果进程执行的程序为非共享的，也可以构成数据段的一部分。该段的大小可以通过一个系统调用加以扩充或压缩。

（源 共享正文段：又称纯正文段、不可修改段。是可再入的程序代码。为多个进程共享同一数据区提供了可能。该段为可选择部分，当进程需要与其他进程共享某一内存段时，使用系统调用将其连入本进程，不再使用后可以使其脱离本进程。

（缘 工作区：进程在用户态和核心态下运行的工作区分别称为用户栈和核心栈。用于函数调用时的参数传递，现场保留，返回地址保存，局部变量存储等。

圆进程映像存储器中的分布

存放进程映像的存储器包括两部分：一部分是内存，另一部分是作为内存扩充部分的外存。存放进程映像的这部分外存盘区称为盘交换区（ P_d 区）。进程映像分为常驻内存和非常驻内存两部分。

- 常驻部分：包括进程基本控制块 结构和共享正文段的控制信息块。
- 非常驻部分：包括进程扩充控制块 结构、数据段、共享正文段和工作区。

进程虚地址空间分布如图 圆所示，进程虚空间从虚地址 开始，进程的正文段、数据段和共享正文段安排在 P_c 区，正文段和数据段逻辑上是连续的。数据段与共享正文段之间留有间隙，用户在连入共享正文段时可以指定间隙的大小，缺省值为 个页面，以便数据段可以向高地址方向动态增长。核心栈、 结构和用户栈安排在 P_d 区。核心栈从 地址开始向低地址方向扩张，其长度固定，与 一起共有 字节。用户栈可向低地址方向动态增长。正文段的保护方式为用户方式只读（ R ），数据段和用户栈为用户方式可读写（ RW ），核心栈和 为核心方式可读写（ RW ）。

猿进程映像的数据结构

（员 结构

在 系统 灾中最多容纳 个进程，每个进程对应一个 结构，它是常驻内存部分，组成一个线性表，具体描述如下：

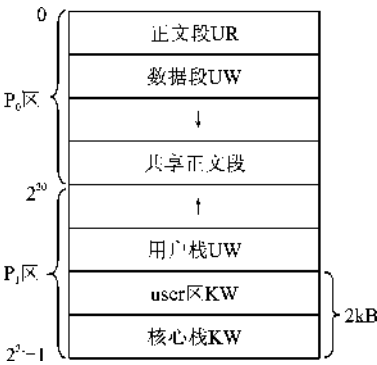


图 圆进程虚地址空间分布

进程控制块 (主要部分)

```
{
    进程控制块

```

进程状态	进程状态 * 进程
进程标志	进程标志 * 进程
进程优先数	进程优先数 * 进程
调度驻留时间	调度驻留时间 * 进程
用于调度的 进程使用情况	用于调度的 进程使用情况 * 进程
计算进程优先数所使用的偏移量	计算进程优先数所使用的偏移量 * 进程
实际用户标识号	实际用户标识号 * 进程
设置 (有效) 用户标识号	设置 (有效) 用户标识号 * 进程
进程组的首进程名	进程组的首进程名 * 进程
惟一的进程标识号	惟一的进程标识号 * 进程
父进程标识号	父进程标识号 * 进程
进程结构的起址	进程结构的起址 * 进程
转换进程页表所用的系统页表项数	转换进程页表所用的系统页表项数 * 进程
进程页表指针	进程页表指针 * 进程
可交换映像的长度 (字节)	可交换映像的长度 (字节) * 进程
交换时的磁盘地址	交换时的磁盘地址 * 进程
已被换出的块 (字节数)	已被换出的块 (字节数) * 进程
正文段长度 (供 进程使用)	正文段长度 (供 进程使用) * 进程
栈段长度 (供 进程使用)	栈段长度 (供 进程使用) * 进程
发向该进程的信号	发向该进程的信号 * 进程

```
};
```

```
{
```

睡眠原因	睡眠原因 * 进程
传递给换出进程的参数	传递给换出进程的参数 * 进程

```
} 进程;
```

```
进程控制块

```

```
进程控制块

```

共享正文段控制块指针	共享正文段控制块指针 * 进程
进程链接指针	进程链接指针 * 进程
发报警时钟信号的时间	发报警时钟信号的时间 * 进程
共享正文段的开始页表项	共享正文段的开始页表项 * 进程
共享正文段的结束页表项	共享正文段的结束页表项 * 进程

```
};
```

(图 4.1 进程结构)

每个进程的 进程结构含有当进程换出内存时不需要访问的所有数据。定义为：

进程控制块 (主要部分)

[illegible]

轡 进程调度时进程硬件 寄存器保存区 * 轡
轡 调度进程的正常保存区 * 轡
轡 中止系统调用的越级返回保存区 * 轡
轡 暂时恢复子进程运行的保存区 * 轡
轡 随机标志
轡 摇摇园: 用户数据空间
轡 摇摇员: 系统空间
轡 摇摇圆: 用户指令空间 * 轡
轡 返回错误代码 * 轡
轡 有效用户标识号 * 轡
轡 有效用户组标识号 * 轡
轡 实际用户标识号 * 轡
轡 实际用户组标识号 * 轡
轡 进程 控制块结构指针, 指向对应于该 进程的进程表项 * 轡
轡 参数表指针 * 轡

轡 一般返回值 员 * 轡
轡 一般返回值 圆 * 轡

轡 返回的返回值 * 轡
轡 返回的时间值 * 轡

轡 随机地址 * 轡
轡 随机字节数 * 轡
轡 随机时在文件中的位移量 * 轡
轡 用于 随机的文件方式 * 轡
轡 随机块中已有字节数 * 轡
轡 随机块中的偏移量 * 轡
轡 随机实际设备号 * 轡
轡 超前读块的地址 * 轡
轡 系统调用出错统计 * 轡
轡 进程当前目录 节点指针 * 轡
轡 进程根目录 节点指针 * 轡
轡 路径名指针 * 轡
轡 存放当前目录项 * 轡
轡 当前父目录指针 * 轡
轡 [系统调用]; 用户打开文件表 * 轡
轡 打开文件标志 * 轡

进程控制块；	进程 当前系统调用的参数 * 进程
进程控制块地址	进程 正文段长度 (字节) * 进程
进程控制块地址	进程 数据段长度 (字节) * 进程
进程控制块地址	进程 堆栈段长度 (字节) * 进程
进程控制块地址；	进程 信号处理表达方式 * 进程
进程控制块地址	进程 进程用户时间 * 进程
进程控制块地址	进程 进程系统时间 * 进程
进程控制块地址	进程 子进程用户时间总和 * 进程
进程控制块地址	进程 子进程系统时间总和 * 进程
进程控制块地址；	进程 保留 0 的用户区地址 * 进程
进程控制块	
进程控制块地址 * 进程控制块	进程 缓冲区基础地址 * 进程
进程控制块地址 * 进程控制块	进程 缓冲区长度 * 进程
进程控制块地址 * 进程控制块	进程 缓冲区移动量 * 进程
进程控制块地址 * 进程控制块	进程 缓冲区比例尺 * 进程
进程控制块地址 * 进程控制块	进程 系统调用参数 * 进程
进程控制块地址；	
进程控制块地址；	进程 进程空间分离标志 * 进程
进程控制块地址；	进程 指向 进程结构中 进程的指针 * 进程
进程控制块地址；	进程 控制 进程设备 * 进程
进程控制块；	

4.1 共享正文段

在一个多用户系统中，不同进程可能调用相同的程序，例如，都调用 编译或 链接装配程序。为了节省空间，相同的程序正文应该允许不同进程之间共享。

进程的正文段由不可重写的程序和参数组成，可被多个进程共享。为了便于对共享正文段进行管理，系统为所有共享正文段设置了一张表，系统实现进程间共享正文段的主要数据结构是 表定义如下：

进程控制块	
{	
进程控制块地址 * 进程控制块	进程 正文段的磁盘地址 (相对于 地址 * 进程
进程控制块地址 * 进程控制块	进程 正文段的块数 * 进程
进程控制块地址 * 进程控制块	进程 指向链接的 进程的指针 * 进程
进程控制块地址 * 进程控制块	进程 指向正文段所在文件的 节点指针 * 进程
进程控制块地址 * 进程控制块	进程 共享正文段的进程数 * 进程
进程控制块地址 * 进程控制块	进程 共享正文段且映像在内存的进程数 * 进程
进程控制块地址 * 进程控制块	进程 标志 * 进程
}	

其中 表项的数目为 取值 也可在系统生成时重新设置。一个共享正文段若正被若干个进程共享，则必驻在盘交换区或内存中。若共享进程中有一个或几个映像在

内存中，则该共享正文段一定在内存中。

进程映像的基本结构图

进程映像、用户以及文本段之间的联系如图 1-10 所示。

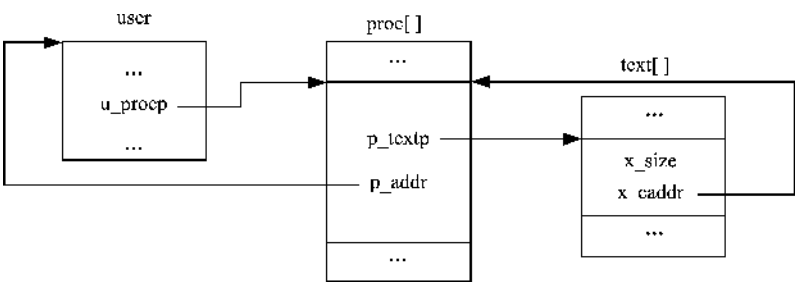


图 1-10 进程映像的基本结构

操作系统的进程状态及变迁

进程状态可以从两个方面理解，一是同一个进程在整个存在周期中不同时刻有不同的状态，另一个含义是同一个时刻不同进程处于不同的状态。操作系统中，进程的完整状态有 9 种，状态转换图如图 1-11 所示。

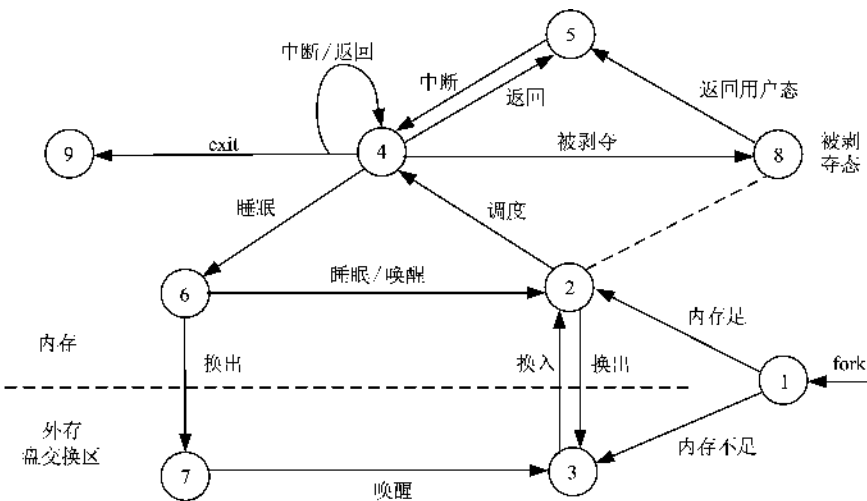


图 1-11 操作系统进程状态转换图

- ① 创建状态：进程刚被创建，处于变迁状态。该进程存在，但还没有就绪，是除了 0 号进程之外所有进程的初始状态。
- ② 内存就绪：进程被执行，但处于就绪状态，只要核心调度它，即可执行。
- ③ 就绪且换出：新创建的就绪进程，因内存紧张而换出。
- ④ 核心态执行：运行在核心模式。

- ⑤ 用户态执行：运行在用户模式。
- ⑥ 内存中睡眠：执行阻塞操作进程，等待。
- ⑦ 睡眠且换出：内存紧张时，睡眠进程首先被换出。
- ⑧ 被剥夺状态：进程被调度程序剥夺了处理机后的状态。
- ⑨ 僵死状态：类似于终止状态。

本章小结

处理机是计算机系统最重要的资源。在现代计算机系统中，为了提高系统的资源利用率，系统将为某一程序独占。通常采用多道程序设计技术，即允许多个程序同时进入计算机系统的内存并行运行。多道程序设计系统充分发挥了处理器与外围设备以及诸外围设备之间的并行工作能力，从而极大地提高处理器和其他各种资源的利用率。

进程概念是操作系统中最基本、最重要的概念。它是在多道程序系统出现后，为了刻画系统内部出现的情况，描述系统内部各作业的活动规律而引进的一个新的概念。

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，进程是系统进行资源分配的一个独立单位。进程是动态产生、动态消亡、可以并发执行的，程序是静态的、顺序执行的；一个程序可以对应多个进程，一个进程也可以对应多个程序。进程的基本特征是：动态性、并发性、独立性、制约性、异步性和结构性。从结构上分析，进程实体由程序段、数据集和进程控制块三部分组成，进程控制块是标识进程存在、刻画进程瞬间特征的数据结构。

进程有三种基本状态：就绪状态、运行状态、等待状态。随着进程的进展，它们在状态之间相互变化。

进程控制是对系统中所有进程从创建到撤销的全过程实现有效的管理和控制，进程控制所采用的是进程控制原语，主要有创建原语、撤销原语、阻塞原语、唤醒原语、挂起原语与激活原语。

进程的模式切换是指进程从运行用户态的程序转入核心态程序执行或反过来从核心态转入用户态。进程切换则是指从一个进程转到另一个进程中运行。

在引入线程概念后，进程成为资源（除进程以外）分配单位，线程则是处理机分配单位。线程控制块类似于进程控制块，是标志线程存在的数据结构，其中包含系统对于线程进行管理所需要的全部信息。线程的实现方法有：在目态实现的用户级线程、在管态实现的核心级线程及混合式线程三种。用内核实现多线程可以发挥多处理机系统对同一进程的并行处理，而用户库程序实现线程则开销少。

在操作系统中，进程被描述为进程映像的执行，其中存储映像是进程映像中最为关键的部分，它由进程控制块、共享正文段、数据段、工作区等几部分组成。操作系统为了节省主存，将进程映像分为两部分：常驻内存的常驻映像和不常驻内存的浮动映像。操作系统中进程的完整状态有 6 种，在一定的条件下，状态之间进行相互转换。

习 题

试述程序的顺序执行和并发执行各有何特点。

什么是进程？操作系统中为什么要引入进程概念？

进程有哪些特征？进程与程序有何区别和联系？

什么是进程控制块？它有什么作用？其中应包括哪些信息？它是如何组织的？

进程至少有哪几种基本状态？它们是如何变化的？

在一个有 n 个处理器的系统中，能够处于就绪、运行、阻塞状态的最大进程数，最少进程数分别为多少？

五态模型的进程中，创建态和结束态的主要作用是什么？

什么是进程的挂起状态？操作系统中引入挂起状态的原因是什么？

什么是原语？它与系统调用有何区别？如何实现原语执行的不可分割性？

创建进程和撤销进程主要应完成哪些工作？

什么是进程切换？叙述进程切换的主要步骤。

什么是模式切换？它与进程切换有何主要差别？

操作系统中为什么要引入线程的概念？

进程与线程的主要区别是什么？

什么是线程？什么是线程控制块？其中主要包括哪些信息？

线程的实现方式有哪几种？

什么是管态？什么是目态？

操作系统中的就绪态是如何构成的？

操作系统中进程有哪几种状态？

第 猿章 摇摇摇摇度

操作系统对处理机的调度是操作系统的资源管理功能中一个重要的问题，其目的是合理、高效地选出作业或进程并分配处理机，它的实现策略决定了操作系统的类型，其调度算法的优劣直接影响整个系统的性能。

调度可分为三级：高级调度、中级调度和低级调度。高级调度即作业调度，决定把外存上处于后备队列中的哪些作业调入内存，以准备执行；低级调度即进程调度，决定就绪队列中的哪一个进程将获得处理机；中级调度决定哪些进程被允许参与竞争处理器资源，其主要目的是为了~~提高~~提高内存的利用率和系统吞吐量。

摇摇摇摇调度策略

处理机是计算机系统中十分重要的资源，在单道批处理时代，作业独立、封闭地占有资源。但多道程序设计技术使资源为多个作业所共享，无法避免地会出现对这一核心资源的争夺。各种不同类型的操作系统为了实现不同的目的，对处理机必须采取相应的有效的管理，因此，各种满足不同的用户需求的调度策略也随之产生。

摇摇调度的意义

决定什么时候、以怎样的方式选择一个作业（进程）运行的这组规则就是调度策略。调度策略有两个最根本的目标：

（~~录~~ 合理性：在多道作业（进程）并发时，各道作业（进程）根据调度策略来分配处理机的使用，调度策略既要保证系统实现特殊功能要求，如实时性，同时也要使各道作业合理地分配~~到~~处理机份额。

（~~圆~~ 有效性：是指通过系统的调度，使处理机和 ~~设备~~ 设备得到合理有效的分配，从而使系统资源得到充分的利用。

摇摇调度模式

处理器的调度模式一般分为三级：高级调度、中级调度和低级调度。

（~~录~~ 高级调度：又称为作业调度。其主要功能是按照某种原则从批处理作业队列或交互作业中选取某一作业进入主存，并为作业做好运行前的准备工作和作业完成后的后期处理。作业从用户提交到最后完成，只需要一次高级调度。

（~~圆~~ 低级调度：又称为处理机调度，其主要功能是按照调度策略将处理机分配给就绪进程或线程。完成此功能的程序称之为“进程（或线程）调度程序”，是操作系统内核的主要部分。

（~~猿~~ 中级调度：又称中程调度，对换调度。引入中级调度的主要目的是为了~~提高~~提高内存的利用率和系统的吞吐量，它决定哪些进程被允许参与竞争处理机资源，哪些进程调至外存上去等待，在合适的情况下，再由中级调度决定将外存上的哪些进程重新调入内存，并将其挂

在就绪队列上，以恢复对处理机资源的竞争。

一个具体操作系统是否有作业和作业调度的概念，是否设置中级调度，因系统而异。但任何一个操作系统都有进程调度。

根据自身的功能特征，以满足系统的需求为目标，各级调度均有其自身的调度策略。

选择调度策略的依据

调度策略具体体现为调度算法。根据系统的不同性质，在选择调度策略时一般要考虑到以下几个方面的问题：

(员 系统设计目标

系统设计目标是决定调度策略的主要依据。批处理系统以充分发挥和提高计算机的效率为目的；实时系统更重要的问题是保证不要丢失实时信息并使之得到及时的处理；而分时系统则应该优先保证用户的请求能够及时给予响应。

(圆 资源利用率

资源利用率是系统性能的一个重要指标，因此，在选择调度策略时，应该考虑在实现设计目标的前提下，尽可能地发挥各种资源的效能。

(猿 均衡系统与用户的要求

系统对作业或进程的调度，需要在系统与用户的冲突中进行适当的协调。一方面，不能因为用户的要求而完全不顾及系统的性能，另一方面，也不应该使一个作业的运行被无限期地推迟。

源 作业与进程的关系

所谓作业，就是用户在一次上机活动中要求计算机系统所做的一系列工作的集合，有的系统也将其称之为任务。一道作业由三部分组成：源程序（或程序）、数据及加工步骤。在支持多道用户程序运行的系统中，不同用户的作业彼此相互独立，也可以认为作业是用户一次提交给计算机系统的一件具有独立性的工作。

计算机系统一旦接收到一道作业，则根据给定的步骤逐步完成对作业的加工活动。从用户提交给到完成后离开系统，作业的整个活动过程常被划分为四个阶段，通常称之为四种状态：提交状态、后备状态、运行状态和完成状态。

(员 提交状态：当用户将作业提交给机房或通过终端将其键入计算机，作业进入提交状态。

(圆 后备状态：通过输入设备的输入，操作系统将其存放到磁盘，作业进入后备状态，等待运行。

(猿 运行状态：作业调度程序依照调度规则将后备状态中的某个作业调度进主存并为之建立进程，投入运行，作业便进入运行状态。

(源 完成状态：当作业全部运行完毕后，作业调度程序为其完成后期处理，作业进入完成状态。

根据系统对作业的不同处理方式，常将作业分为批处理作业与终端交互型作业两大类，这两类作业和进程的关系存在着一些差异。

在批处理作业形式下，作业的数据和程序与如何控制作业自动执行的作业说明书一起提交后，用户不再与系统打交道，系统自动控制作业的全部运行。在这种形式下，作业调度程序每当选择一道后备作业运行时，首先为该作业创建一个“进程”（称为该作业的“根进程”），

由该“进程”完成这一项作业。在作业的运行过程中,“根进程”可以动态地创建一个或多个“子进程”来具体地执行该作业中的某些功能,“子进程”也可以根据作业的需要创建“子孙进程”去完成指定的功能。因此一个作业就动态地转换成了一组运行实体——进程组。当作业全部结束时,“根进程”将作业状态改为完成状态后请求删除进程。

终端交互型作业是用户通过分时终端动态地向系统提供作业控制意图,动态提供作业的步骤命令(称之为作业步),系统以交互的方式处理作业。当用户接通终端设备时系统便为之建立一个进程,称之为终端进程。终端进程执行命令解释程序,解释执行用户键入的每一条命令。对于每一条终端命令可以创建一个子进程去具体执行,各子进程又可以在运行过程中根据需要创建“子孙进程”以完成相应的功能。当终端命令所对应的进程结束后,命令的功能也相应处理完毕。

由上看到,进程是作业的执行状态,一个作业实际上是由一组相应的进程来完成的,当作业所对应的进程完成时,作业便进入了完成状态,整个作业也就完成了。

4.2 作业调度

作业调度为高级调度,又称之为长程调度(进程调度),主要是选择后备队列中的某些作业,完成其从后备状态到运行状态的转变。

4.2.1 作业调度的功能

当作业调度从后备队列中选取相应的作业并将其转变到运行状态时,主要完成以下几方面的工作:

(1) 记录系统中各作业的状况

系统为了对作业实施有效的管理,每一个作业均有一个作业控制表(JCB),系统在作业进入后备状态时完成该作业的 JCB 建立。每一个作业在各阶段所要求和分配的资源以及该作业的状态都记录在它的 JCB 中,根据 JCB 中的有关信息,作业调度程序对作业进行调度和管理。

(2) 为作业做好执行前的准备工作

作业调度程序为作业建立相应的进程,并为这些进程分配它们所需要的系统资源,如内存、外存、外设等。

在批处理系统中,作业进入系统后,是先驻留在外存上的,因此需要有作业调度,以将它们分批装入内存。然而,在分时系统中,为了能及时响应,用户通过键盘输入的命令或数据等,都是直接送入内存,因而无需配置作业调度。类似地,在实时系统中,通常也不需要作业调度。

4.2.2 作业调度的依据

当运行状态的某一道作业被撤除或者是作业后备队列中增加新的后备作业时,作业调度便会引发。作业调度依据的原则有两条:

(1) 接纳多少个作业

作业调度每次要接纳多少个作业进入内存,取决于多道程序度(即同时运行的作业数),即允许有多少个作业同时在内存中运行。当内存中可以同时运行的作业太多时,可能会影响到系统的服务质量,如使周转时间太长,资源冲突严重。但是,如果内存中同时运

行的作业太少时,又会导致系统资源利用率和系统吞吐量太低,因此,多道程序度的确定应根据系统的规模和运行速度等参数,作适当折中。

④ 接纳哪些作业

应将哪些作业从外存调入内存,将取决于所采用的调度算法。因为系统要求的差异,各个系统可能采取的调度算法也各有其特点。

⑤ 作业调度算法的两种类型

考虑作业调度算法时,必须综合考虑用户的方便性和资源的利用率两个方面。在多道程序设计系统中,调度算法通常可以分为两类:

(1) 基于作业优先级。为了尽可能合理地调度作业,通常根据作业的某些属性为作业规定一个调度参数——优先级(或称作业运行优先级),作业调度程序则根据优先级的高低决定它们的调度顺序。决定优先级的因素很多,如用户任务的紧迫程度、上机所付费用、作业在后备队列中等待的时间、作业所需的运行时间、作业所需的内存量或者作业所需的 I/O 量等,都可以按一定的规则反映在优先级这一参数上,以尽可能合理地满足各种应用的需求。

优先级的确定方法分静态优先数法和动态优先数法两种。所谓静态优先数法是指作业在建立时即为它规定了一个优先数,并在其作业的生存周期中保持不变。动态优先数法是指作业的优先数动态确定,并随着作业活动的过程不断变化,其变化规律的不同体现着不同的调度目标。

(2) 基于作业对资源需求量。作业优先级法较好地满足了用户对系统的要求,但没有顾及系统资源的使用状况。有些系统采用资源需求量法,从有效利用资源为出发点,把对资源需求量有着互补性的作业搭配在一起进行调度。比如:将短作业和长作业搭配,能减少作业之间的相互等待时间;将内存需求少的作业和内存需求大的作业搭配,能提高内存的利用率,使内存能够尽可能经常保持多道作业;将 I/O 型作业与 CPU 型作业搭配,可以提高 CPU 与 I/O 设备之间的并行性,等等。

⑥ 进程调度

进程调度是低级调度(即进程调度),又称之为短程调度(即进程调度),它决定就绪队列中的哪个进程将获得处理机,然后由分派程序(即进程调度)执行把处理机分配给该进程的操作。

引发进程调度的原因可能是处理机已进入空闲状态,而就绪队列中还有进程在等待处理,通过调度选择另一个进程开始运行。另一种可能的原因是系统的进程调度策略允许新的进程有条件地来抢占处理机。

⑦ 进程调度发生的条件

通常有以下几种可能的情况:

(1) 进程完成;

(2) 分配给进程的时间片用完;

(3) 进程自身的原因如请求 I/O 服务,暂时放弃 CPU;

(4) 在抢占式调度中,新建进程的优先级比运行进程的优先级高;

(5) 某些原语操作。

进程调度的频率很高,在分时系统中,通常是几十毫秒就要运行一次。进程调度是最基本的一种调度,不管是批处理,还是分时或实时操作系统中,都必须配置这级调度。

进程调度的功能

进程调度有以下几方面的功能:

(1) 记录进程的相关信息

每一个进程都有一个就绪表,用来记录进程的执行情况和状态特征,是进程调度的重要依据。进程调度也利用就绪表来记录相关的信息,以便对进程进行有效的调度。

(2) 选择进程以分配处理机

进程调度的主要目的是要按照一定的调度策略从就绪队列中选择进程,为其分配处理机,使之投入运行。进程的调度策略是操作系统中的一个重要内容,它决定着系统的性能特征。

(3) 进行进程的上下文切换

一个进程的上下文(Context)包括进程的状态、有关变量和数据结构的值、机器寄存器的值和就绪以及有关程序、数据等。一个进程的执行是在进程的上下文中进行。当正在执行的进程由于种种原因要让出处理机时,系统要做进程上下文切换,以使另一个进程得以执行。

进程上下文切换

在进程因某种原因而切换时,包括进程的状态、有关变量和数据结构的值、机器寄存器的值和就绪以及有关程序、数据等的上下文需要切换。切换包括以下四个步骤。

(1) 决定是否做上下文切换以及是否允许做上下文切换。在有些情况下,如系统正在执行某个不允许中断的原语时,上下文切换是不允许的。

(2) 保存当前正在执行的进程的上下文。在发生进程切换而由此引发进程的上下文切换时,系统要对正在执行的进程的上下文实施保存操作,确保它不会遭到破坏。

(3) 按照进程调度策略选择适当的就绪进程。

(4) 装配所选进程的上下文,将处理机的控制权交给新选进程。

进程调度有两种基本方式

(1) 非抢占方式(Non-preemptive Scheduling)

非抢占方式,也称非剥夺调度。采用这种调度方式时,一旦把处理机分配给某进程,便让该进程一直执行,直至该进程完成或因其自身的原因被阻塞,才把处理机分配给其他进程,在此之前,决不允许其他进程抢占处理机。

这是一种较为简单的调度方式,因为只有在进程自动放弃处理机时,才进行调度,因此系统开销小。在这种调度方式下,程序设计也相应简化,因为采用非抢占方式时,程序员可以在某种程度上预知进程的运行轨迹,可以预知进程将在何时、何处被暂停运行,从而给软件设计带来很大方便。

非抢占方式的缺点是损失了系统的并发性,使系统不能根据内部的并发事件及时实施进程调度,在要求比较严格的实时系统中,如果采用这种调度方式,难以实现调度目的,而一些小型操作系统或批处理系统中可以采用这种调度方式。

(2) 抢占方式(Preemptive Scheduling)

抢占方式,也称剥夺调度。这种调度方式,允许调度程序根据一定的原则,去停止某个正在执行的进程,将已分配给该进程的处理机,重新分配给另一进程。

抢占的原则有三种：

(景 时间片原则：各进程按时间片运行，当一个时间片用完后，便停止该进程的执行而重新进行调度。这种原则在分时系统、大多数实时系统，以及要求较高的批处理系统中得到了应用。

(圆 优先权原则：通常是对一些重要的和紧急的作业赋予较高的优先权，当这种作业到达时，如果其优先权比正在执行的进程的优先权高，便停止正在执行的进程，将处理机分配给优先权高的进程，使之执行。

(猿 短进程优先原则：当新到达的进程比正在执行的进程明显地短时，将剥夺长进程的执行，将处理机分配给短进程，使之优先执行。

相对于非抢占调度方式而言，抢占方式的控制比较复杂，但在该方式下，系统并发性强，能够满足系统的多种要求，因此适应面很广。

进程调度算法

综合上述两种基本调度方式以及系统的不同需求，进程调度常用的算法有如下 远种：

- 先来先服务算法
- 短进程优先算法
- 时间片轮转算法
- 优先权调度算法
- 多队列调度算法
- 多级反馈队列调度算法

进程调度算法

调度算法是根据系统的资源分配策略所规定的资源分配算法。

调度算法必须实现几个相互冲突的目标：进程响应的时间尽可能快，后台作业的吞吐量尽可能高，进程的饥饿现象尽可能避免，低优先级和高优先级进程的需要尽可能调和，等等。

目前存在着如下几种调度算法，适用于不同的系统，以实现不同资源管理目标。

先来先服务调度算法 (FIFO)

在作业调度中，采用该算法调度时，每次从后备作业队列中选择一个或多个最先进入该队列的作业，将它们调入内存，为它们分配资源、创建进程，然后放入就绪队列。

在进程调度中，采用该算法调度时，每次调度是从就绪队列中，选择一个最先进入该队列的进程，把处理机分配给它，使之投入运行。该调度方式属于非抢占方式，获得处理机的进程将一直运行，直到该进程完成或因进程本身发生某事件而阻塞后，才放弃处理机。

先来先服务调度算法是一种非抢占方式调度算法，它能公平地使每个作业（进程）获得处理机。

短作业（进程）优先调度算法 (SJF)

短作业优先调度算法 (SJF) 是一个非抢占方式的算法，是指对短作业优先调度，即从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存。

短进程优先调度算法 (SPF)，是指对短进程优先调度，即从就绪队列中选择一个或若干个估计运行时间最短的进程，为它们分配处理机，使之投入运行。

时间片轮转调度算法

(员 基本原理

为了保证人机交互的及时性,系统让每个进程依次按时间片轮流的方式执行,这种调度方法称之为时间片轮转法,这种算法多用于进程调度,以提高进程的并发性,缩短每一个作业的响应时间,从而提高系统的资源利用率。

在通常的轮转算法中,系统将所有就绪进程按“先来先服务”原则,排成一个队列,每次调度时把时间片分配给队首进程,并令其执行一个时间片。时间片的大小约几到几百毫秒。当执行的时间片用完时,由一个计时器发出时钟中断,调度程序便据此信号来中断该进程的执行,并将它送就绪队列的末尾,等待下一次调度;然后,把处理机分配给就绪队列中的新的队首进程,同时也让它执行一个时间片。这样就可以保证就绪队列中的所有进程,在一给定的时间,如用户所能接受的等待时间内,均能获得一时间片的执行。

(圆 时间片大小的确定

在时间片轮转算法中,时间片的大小对调度的性能有很大影响。如果时间片太大,大到每个进程都能在一个时间片内执行完毕,则时间片轮转算法的调度效果与先来先服务算法相同,可能导致用户在期待的时间内得不到及时的响应。当时间片过小时,用户键入的常用命令就需要花费多个时间片才能处理完毕,这同样难于保证用户对响应时间的要求。因此,时间片的大小应综合考虑以下几个因素来确定:

第一,系统对响应时间的要求:作为分时系统首先就必须满足系统对响应时间的要求。由于响应时间与用户(进程)数目成正比,即与时间片成正比(不考虑系统开销),因此在用户(进程)数一定时,时间片的长短将正比于系统所要求的响应时间。例如,如果要求响应时间为 1 秒,而系统中有 100 个用户,则时间片应为 10 毫秒。

第二,就绪队列中进程的数目:在分时系统中,就绪队列上的进程数,是随着在终端上机的用户数目而改变的,但系统应保证,当所有终端都有用户同时上机时,仍能获得较好的响应时间。因此,时间片的大小应反比于分时系统所配置的终端数目。

第三,系统的处理能力:对应于该系统的处理能力,必须保证用户键入的常用命令能在一个时间片内处理完毕,否则将无法保证得到满意的响应时间,而且会使平均周转时间及带权周转时间都很长。

优先级调度算法

(员 优先级调度算法的基本原理

为了满足紧迫作业对进入系统后的响应时间的要求,引入了最高优先级调度算法,它常被用于批处理系统中作为作业调度算法,也作为多种操作系统中的进程调度算法,更可用于实时系统中,以实现对实时作业的实时响应。当将该算法用于作业调度时,系统将从后备队列中选择若干个优先级最高的作业调入内存。当用于进程调度时,是把处理机分配给就绪队列中优先级最高的进程。优先级在调度过程中所起的作用与系统对进程调度采用非抢占式调度策略还是抢占式调度策略有关。

在非抢占式优先级策略下,系统一旦把处理机分配某一高优先级的进程后,该进程便一直执行下去,直到完成,或因发生某事件而进程自身阻塞,使该进程放弃处理机。该进程对处理机的占有不会受新进程的高优先级的影响,而高优先级的新进程只能等待正在运行的进程自动放弃处理机后,才能得到调度。

而在抢占式优先权调度策略下,系统同样是把处理机分配给优先权最高的进程,使之执行。但一旦出现了另一个优先权更高的进程时,进程调度程序就停止原最高优先权进程的执行,而将处理机分配给新出现的优先权最高的进程。因此,在采用这种调度算法时,每当出现一新的就绪进程 P_n 就将其优先权 P_n 与正在执行的进程 P_k 的优先权 P_k 进行比较,如果 $P_n \leq P_k$,原进程 P_k 继续执行;如果 $P_n > P_k$,则立即停止 P_k 的执行,进行进程的切换,使 P_n 投入执行。显然,使用这种方式的优先权调度算法,系统对进程的实时响应能力强,能更好地满足紧迫作业的要求。

4.1 优先权的类型

对于最高优先权算法,其关键在于优先权如何确定。根据确定优先权的方法的不同,优先权分静态优先权和动态优先权两种。

4.1.1 静态优先权

静态优先权是在创建进程时确定的,且规定它在进程的整个运行期间保持不变。一般地,优先权是利用某一范围内的一个整数来表示。例如,用 $0 \sim 9$ 或 $0 \sim 15$ 中的某一整数,又称该整数为优先数。具体用法不同的系统有所不同:有的系统用“0”表示优先权最高,数值愈大其优先权愈低;而有的系统恰恰相反。

进程优先权的确定依赖于以下几个方面:

- ① 进程类型。通常,系统进程(如接收进程、对换进程、磁盘 I/O 进程)的优先权高于一般用户进程的优先权。
- ② 进程对资源的需求。如进程的估计执行时间短及内存需要量少的进程,应赋予较高的优先权。
- ③ 根据用户要求。这是根据用户进程的紧迫程度及用户所付费用的多少来确定进程的优先权。

静态优先权法简单易行、系统开销小,但不够精确,很有可能出现优先权低的作业(进程)得不到合适的调度的情况。因此,仅在要求不太高的系统中,才使用静态优先权法。

4.1.2 动态优先权

动态优先权是指在创建进程时所赋予的优先权,可以随进程的推进而改变,以便获得更好的调度性能。

动态优先权较之静态优先权能更好地反映系统运行的状态,以根据系统的实际状况合理地调整进程的调度。例如,我们可以规定,在就绪队列中的进程,随其等待时间的增长,其优先权以速率 α 增加。此时,若所有的进程都具有相同的优先权初值,则显然是最先进入就绪队列的进程,将最先获得处理机执行,此即 $FCFS$ 算法;若所有的就绪进程具有各不相同的优先权初值,或者优先权增长的速率 α 不同,对于优先权初值低的进程,在等待足够的时间后,其优先权便可升为最高,从而可以获得处理机执行,因而避免了长作业的长等待现象。又如,当采用抢占式优先权调度算法时,如果再规定正在执行的进程,其优先权以速率 β 下降,于是便可防止一个长作业长期地垄断处理机。

进程本身的特性同样是决定进程优先级的重要依据:

- ① I/O 型进程:赋予较高的优先权,以便能及时响应需要 I/O 交互的进程。通常执行一个小的时间片,在该时间片内,要求能处理完一次 I/O 请求的数据,然后转入阻塞队列。
- ② 计算型进程:每次都执行完时间片,降低其优先权。

③ 进程的优先级不多而主要是根据进程的进程：在进程完成后，放回进程的请求时离开的优先级队列，以免每次都回到最高优先级队列后再逐次下降。

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

优先级响应比优先调度算法

在批处理系统中，短作业优先算法是一个简单而有效的调度算法，此时的主要缺点是长作业的运行得不到保证。为此，我们为每个作业引入前面所述的动态优先权机制，使之以速率 α 增加，如下式所示：

优先权 $\propto$ (等待时间 + 要求服务时间) / 要求服务时间

由此，长作业在等待一定的时间后，优先权数按给定速率增长，必然有机会分配给处理机。

该优先权的分子是等待时间加上要求服务时间，即是系统对该作业的响应时间，而响应时间与服务时间的比值称之为响应比，故该优先权又相当于响应比。

由上可以知：

(1) 如果作业的等待时间相同，则要求服务的时间愈短，其优先权愈高，因而该算法有利于短作业，保持了短作业优先的特点；

(2) 当要求服务的时间相同时，作业的优先权决定于其等待时间，因而是实现了先来先服务，满足了用户的分时要求；

(3) 对于长作业，当其等待时间足够长时，其优先权便可升到很高，从而也可获得处理机，避免了长作业长时间得不到服务的现象。

简言之，该算法既照顾了短作业，又考虑了作业到达的先后次序，也不会使长作业长期得不到服务。因此，该算法实现了一种较好的折中。当然，在利用该算法时，每次进行调度之前，都需先进行响应比的计算，这会增加系统的开销。

多级队列调度算法

目前在计算机系统中，配置的操作系统既具备批处理功能，又具有分时系统的功能以适应批处理作业和交互型作业的不同需要。通常，为批处理作业所建立的进程是排入后台的就绪队列；而为交互型作业所建立的进程，则是排入前台的就绪队列。前台采用时间片轮转调度算法，以使用户能与自己的交互型作业进行交互；而后台可采用优先权高者优先的调度算法或短作业优先的调度算法。因此，这种情形下的调度是对各个队列分级施行的。

一般地说，多级队列调度下，就绪队列被分为若干个独立子队列，例如，在分时操作系统中，就存在着五个独立的就绪队列：系统进程队列、交互式进程队列、交互编辑进程队列、批处理进程队列和学生进程队列。作业和队列的关系则是根据其性质或类型来决定，各个作业固定地分属于一个队列，采用不同的调度算法。

队列之间的关系可以采取多种方式来处理，应用较多的一种方式规定好每个队列的优先权，优先权高的队列优先获得调度。如在具有前、后台队列的系统中，是优先调度前台队列中的进程执行；仅当前台队列中已无可运行的进程时，方才调度后台队列中的进程运行。如在分时操作系统中规定，系统进程具有最高优先权，然后是交互式进程，学生进程优先权最低。仅当前台四个队列中都已无可运行的进程时，方才调度学生队列中的进程运行。另一种处理各队列间关系的方式是为各队列分配一定的占用 CPU 的时间比例。比如，可为前台队列分配 80% 的 CPU 时间，给后台队列 20% 的 CPU 时间。

多级反馈队列调度算法

前面所介绍的各种可用作进程调度的算法，都存在一定的局限性。如短进程优先调度算法，仅照顾了短进程而忽略了长进程，而且如果未指明各进程的长度，则短进程优先和基于进程长度的抢占式调度算法，都将无法使用。而多级反馈队列调度算法基于时间片轮转调度算法和优先级算法发展而来，却又不必估计运行时间的长短，因而是目前公认的较好的一种进程调度算法。

本算法基于以下情况来考虑：

- ① 为提高系统吞吐量和降低进程平均等待时间而照顾短进程。
- ② 为得到较好的输入输出设备利用率和交互用户的及时响应而照顾输入输出型进程。
- ③ 在进程运行过程中，按进程运行情况来动态地考虑进程的性质，进行相应的调度。

在采用多级反馈队列调度算法的系统中，调度算法的实施过程如下：

首先，应设置多个就绪队列，并为各个队列赋予不同的优先权。第一个队列的优先权最高，第二个队列次之，其余队列的优先权逐个降低。

其次，赋予各个队列中进程执行时间片大小也各不相同，在优先权愈高的队列中，每个进程的执行时间片就规定得愈小，而优先权低的队列会分配成倍于高一级优先级的队列的时间片，即第 m 级(优先级)队列的时间片比第 $m-1$ 级(优先级)队列的时间片长一倍。

第三，当一个新进程进入内存后，首先将它放入第一队列的末尾，按先来先服务原则排队等待调度。当轮到该进程执行时，如能在该时间片内完成，便可准备撤离系统；如果它在一个时间片结束时尚未完成，调度程序便将该进程转入第二队列的末尾，再同样地按照先来先服务原则等待调度执行；如果它在第二队列中运行一个时间片仍未完成，再依法将它输入第三队列……依此规律，直至进程完成。若进程在进入最后一个队列后仍未执行完毕，则在该队列采用时间片轮转法，直至进程执行完毕。

第四，仅当第一队列空闲时，调度程序才调度第二队列中的进程运行；仅当第 $m-1$ 级队列为空时，才调度第 m 级队列的进程运行。如果处理机正在运行第 m 级队列的进程，而此时又有新的进程进入优先权较高的任一队列(第 $m-1$ 级或第 $m-2$ 级队列)，则此时新的进程将抢占处理机，即调度程序将处理机重新分配给新进入第 $m-1$ 级或第 $m-2$ 级队列的进程，并把被抢占的进程投入原队列的末尾，如图 9.10 所示。

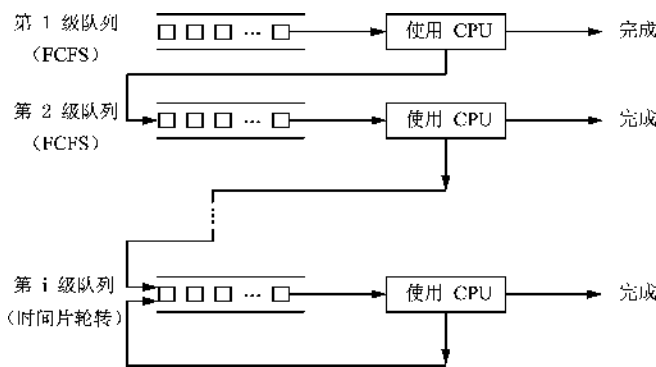


图 9.10 多级反馈队列示意图

猿.猿 摇调摆算法评价

调度算法的性能是衡量操作系统设计的一个重要指标。调度性能的衡量可以分为定性和定量两种。

在定量评价调度算法时，主要有以下几个方面：

- ① 悦裁利用率；
- ② 单位时间内 悦裁完成作业的数量；
- ③ 从作业提交到作业完成的时间间隔。

因此，我们常用以下几个指标来定性地衡量调度策略：

- ① 周转时间：将一个作业提交给计算机系统后到该作业完成并将结果返回给用户所需要的时间。周转时间包括作业在外存后备队列上等待作业调度的时间；进程在就绪队列上等待进程调度的时间；进程在 悦裁上执行的时间；等待 豫酌操作完成的时间。
- ② 带权周转时间：作业的周转时间与系统为其服务时间之比。
- ③ 吞吐率：在给定时间内，一个计算机系统所完成的总工作量。
- ④ 响应时间：指从用户向计算机发出一个命令到计算机把相应的执行结果返回给用户所需要的时间。

⑤ 设备利用率：指输入输出设备的使用情况。在一些对 豫酌处理能力要求高的系统中，高的设备利用率也是一个衡量调度策略好坏的重要指标。

前一节我们介绍了 苑种不同的调度算法，从这些算法的本质上来看，可以分为两大类型：一类是以作业（进程）的客观属性为主要调度依据来进行调度，称之为客观型调度算法；另一类是以人为赋予的某一参数为主要调度依据，称之为主观型调度算法。以下分别对这两类调度算法的性能进行定性的分析。

猿.猿.猿 客观型调度算法

① 先来先服务调度算法（云云云）

先来先服务调度算法只是简单地按照进程或作业到达的先后次序来调度。

假定 粤 月 悦 阅四个作业分别到达系统的时间、要求服务的时间、开始执行时间以及各自的完成时间，并计算出各自的周转时间和带权周转时间如下表。

进程名	到达时间	服务时间	开始执行时间	完成时间	周转时间	带权周转时间
粤	园	员	园	员	员	员
月	员	员	员	员	员	员
悦	圆	员	员	员	员	员
阅	猿	员	员	圆	员	员

从上表可以看出：优先到达的作业最先得到调度，有着最优的带权响应时间。但其后到的长作业 阅也有较优的带权周转时间，仅为 员 带权周转时间最长的是后到的短作业 悦高达 员 这是用户非常难以容忍的。

续上表

短作业(猿猿猿)	完成时间	猿猿	猿圆	猿愿	猿	猿苑	
	周转时间	猿猿	猿员	猿远	远	猿猿	猿猿愿
	带权周转时间	猿猿猿	猿猿苑	猿猿圆	猿	猿猿猿	猿猿猿
短作业(猿猿猿)	完成时间	源	苑	愿	猿	苑	
	周转时间	源	远	愿	猿	猿	猿愿
	带权周转时间	员	圆	猿圆	缘	猿猿猿	圆猿愿

从上表中看到：

(猿) 长作业在 猿猿猿 中有着较优的参数，在三种调度方式中最优。

(圆) 猿猿猿 下平均周转时间和带权平均周转时间均较优，因此系统效率较优，将有效地降低作业的平均等待时间，提高系统的吞吐量。而短作业进程 猿的带权周转时间从 猿猿猿 下降到 猿猿，长作业(进程 猿的带权周转时间从 猿猿猿 算法下的 猿圆增加到 猿猿) 也就是说，短作业的优势非常明显，长作业有可能得不到合适的调度。

(猿) 猿猿猿 方式下，表中数据显示，在该方式下的各个作业的带权周转时间比较均衡，先后来后到的差别被消除，充分体现了分时系统的需要。时间片的大小对效率参数起着非常大的影响作用，从表中看到，对于某一组作业(进程)，当时间片足够大时，它与 猿猿猿 调度算法有着完全相同的调度效果。

猿猿猿 主观型调度算法

猿猿 优先权调度算法

前面三种算法比较尊重作业(进程)的客观情况，在这些算法下，系统的操控能力较低，对于实时性强的作业(进程)无法实现优先调度，而在 猿猿猿 和 猿猿猿 两种算法中，长、短作业得不到兼顾。

优先权调度法加入了一个优先权参数，该参数的主观可调性为操作者控制系统的调度性能提供了方便，对于紧急作业(进程)，可以人为赋予高优先权，用抢占式优先调度保证其实时性；而动态优先权法能够在运行过程中根据系统的状况及时调整调度，因而，长短作业(进程)得到了很好地协调，既能避免短作业(进程)的长时间等待，又能使长作业(进程)得到适时的调度，兼顾了 猿猿猿 和 猿猿猿 的优点。

猿猿 高响应比优先调度算法

高响应比优先调度算法也是一种优先权调度算法。但这一算法是在短作业优先的基础上，对等待时间长的作业动态地赋予它不断增长的优先级。这种算法保留了短作业优先算法的优点，并通过优先级参数的动态修改，避免了长作业的长等待。

猿猿 多级队列调度算法

多级队列调度对进程施行分级管理，同时也对队列和队列内的进程进行分级调度。这种分而治之的调度策略使得不同类型的进程能够得到更有效的管理，使得系统对进程的管理更为合理，采用此算法的系统具有更强的适应能力。

猿猿 多级反馈队列调度算法

多级反馈队列调度算法是动态优先权调度的一个典型的示例。

对于终端型作业用户所提交的交互型作业,作业通常较短小。通过合理地安排时间片,系统能使这些作业(进程)在第一队列所规定的时间片内完成,便可使终端型作业用户都感到满意。

而对于短批处理作业用户,开始时像终端型作业一样,如果仅在第一队列中执行一个时间片就完成,便可获得与终端型作业一样的响应时间。对于稍长的作业,通常也只需在第二队列和第三队列中各执行一个时间片即可完成,其周转时间仍然较短。

而批处理作业用户的长作业,它将依次在第 1 圆...直到第 灶个队列中运行,然后再按轮转方式运行,因而长作业也不会长期得不到处理。

主观型调度算法的优劣

总的来说,主观型调度算法因为加入了人为的调整参数,能更好地表达系统的调度目的,因而,较之客观型调度算法均有着较优的调度效果,更符合实际情况的需要。

然而,主观型调度算法较之客观型调度算法在算法上来得复杂一些。例如,采用优先级抢占式调度时,优先级高低的比较,以及高优先级的作业(进程)在抢占处理机时所引发的进程的切换,都需要损耗系统的性能;又如,在多级反馈队列调度算法中,作业(进程)的优先权数随着进程的执行而计算并动态地调整,也需要损耗系统的性能。这一切,会对系统的性能带来一些负面的影响。

进程调度与换进换出

操作系统的调度由 调度程序 过程实现。此外,操作系统还设置了中级调度(交换),交换由 换入换出 过程完成,主要负责将在外存盘交换区上处于就绪的进程换入内存。操作系统的调度与交换都属于进程管理。

分时系统中的进程调度

分时系统中,系统核心给每一个进程一段 时间片。当这段时间片过去时,核心抢占该进程并调度另一个进程。调度函数使用运行的相对时间作为下次进程调度的参数。每个进程都有一个调度优先级。

分时调度策略

操作系统的进程调度采用多级反馈轮转调度法(即 时间片轮转调度法)。即系统分给进程一个时间片,当时间片结束时,动态计算该进程的优先级,若有优先级高于当前进程的内存就绪态进程时,在由核心态转换到用户态前由 调度程序 过程调度优先级高的进程执行,并把被抢占的进程反馈到相应的优先级队列中。一个进程在它结束之前,可能要经过多次反馈循环。

操作的进程调度按时间片计算优先级,并按优先级的高低来调度进程,抢占处理机。因此,操作的进程调度涉及的主要问题是:

- (1) 调度时机。
- (2) 优先数的计算。
- (3) 调度的实现。

调度时机

一种调度是进程本身无法预知的。在 操作系统中,在系统程序执行期间不会发生调

度。调度程序 `schedule()` 是系统程序，它一般在处理机执行完核心程序之后向用户态转换之前的时间间隙里，检查各就绪进程的优先级并进行调度。

另一种调度在以下几种情况下发生：

(员) 当进程因申请系统资源而未得到满足，从而调用过程 `wait()` 放弃处理机。

(圆) 进程为了与其他进程保持同步而调用 `wait()` 或 `waitx()` 过程主动放弃处理。

(猿) 由于执行了 `kill()` 调用，终止了当前进程。

在这几种情况下，由于发生调度的时机是进程本身可以预见的，因此，在这些程序执行结束之前，都自动调用 `schedule()` 过程调度其他优先级高的进程执行。

进程优先数的计算

操作系统采用动态优先数调度策略，而且设定为优先数越大，则优先级越低。因此，调度策略是从内存就绪队列中选取优先数最小的进程投入运行。

优先数 `priority` 的计算由下式完成：

$$priority = \frac{1}{(1 + \frac{1}{2})^{(t - t_0) / \Delta t}} \times (1 + \frac{1}{2})^{(t - t_0) / \Delta t}$$

式中几个参数的意义如下：

(员) `priority`：每个进程最近使用 `wait()` 的时间，正占有 `wait()` 的进程每一个时钟周期该项值增加 1，最多为 10。同时系统又按每秒钟的时间间隔对 `priority` 执行除以 2 的衰减操作。

(圆) `base`：为系统允许用户设置的一个计算进程优先数的偏置值，但用户仅有权通过 `wait()` 过程将其设置为 0 到 10 中的一个数。一旦设置后，再作改变，仅能使其增加。特权用户有权减少 `base` 的值。

(猿) `1` 和 `10` 是两个常数，分别为 1 和 10。

由上式及各参数的运算规律可知，当一个进程新创建时，由于其 `wait()` 占有时间为零，因此，它拥有较高的优先级而得到调度，对于一个时间片内能够完成的短进程，系统给予了相当的优先，但如果该进程在一个时间片内没有能够完成，则其优先数按每使用一个时钟周期增加 1 的速度递增，优先级下降，高优先级的进程将抢占处理机。然而，随着等待时间的增加，该项值又按一定的速率减少，优先级随之升高，最终重新获得机会抢占处理机。

进程调度的实现

进程调度程序 `schedule()` 作为一个函数供核心栈上的执行程序在需要时调用，它的主要任务是完成一个进程上下文切换的过程，由以下几个步骤来完成：

(员) 检查并决定是否做上下文切换和系统是否允许上下文切换，如果允许上下文切换，则保存当前进程的上下文。在系统内部数据更新完成、各种队列已正确链接、各种数据结构的加锁解锁完成以前，系统不允许进行上下文切换，这种方法可以保证内部数据结构的完整性和一致性。

(圆) 使用如上所述的调度算法，从内存就绪队列中寻找一个优先级最高的进程，选取该进程分配处理机。

(猿) 被选中的进程已成为当前进程，为其恢复有关寄存器内容和栈指针。

进程的换进换出

为了能使更多的进程同时参与系统资源的竞争，系统采用盘交换区作为内存的逻辑扩充，并提供两个基本操作：换进与换出。

进程园也称为交换进程,该进程只在核心态下运行,是惟一的将进程从交换设备上换入内存的进程。在系统初始化结束时,交换进程就进入无限循环,它的任务就是将进程换入或换出,先试图换入进程,内存不够时才换出进程。如果没有进程要换入或者没有进程要换出,交换进程本身进入睡眠状态,核心通过时钟中断定期地唤醒交换进程。交换进程一旦被唤醒,就寻找所有处在“就绪且换出”状态的进程,从中选取换出时间最长者,如果有足够空闲内存空间就将它换入。换入步骤主要包括:分配内存空间,将进程从交换区读入,释放交换空间。换入一个进程后又继续寻找满足上列条件的进程,重复换入工作,一直下去,最终会发生下列情况之一:

- 交换区里没有“就绪”的进程可换入,进程园睡眠,直到有一个进程被唤醒或者换出一个“就绪”进程。

- 有可换入的进程但无空间,转而换出另外一些进程,换出成功后再寻找换入进程。

换出时要查看内存的每个进程,只换出正在内存睡眠的进程,并且要有时间条件:一个进程被换出前必须至少在内存驻留 10 秒钟,换入的进程必须至少在交换区内停留 10 秒钟。

基于停留时间和优先数决定换入换出的交换办法以求得内存空间的扩充,此外也有不少问题:

(1) 为了要换入而需要换出以腾出内存空间,可能换出一个仍不能满足空间要求。

(2) 交换进程醒来后要重新寻找换入进程,由于睡眠期间情况有变,可能找到的是另一个要换入的进程。

(3) 换入内存后还没有运行就可能因为别的进程要换入而被换出。

(4) 可能会遇到死锁:要换出一个进程但没有交换空间又加上其他一些情况。

解决这些麻烦的办法,是采用请求页式调度算法,近年来已在 8086 上实现。换进换出的具体实现由函数 `swapin()` 完成。图 10.10 是 `swapin()` 的工作流程。

说明:

(1) 关于 `swapin` 和 `swapout` 标志

`swapin` 标志表示外存交换区有就绪进程等待换入内存,但内存中没有符合条件要换出的进程,进程园在 `swapin` 上等待。以后某个时刻内存有符合换出条件的进程时,如进程进入低优先级睡眠又无待处理的信号,则唤醒在 `swapin` 上睡眠的进程园。`swapout` 标志表示外存交换区没有要换入的进程。进程园在 `swapout` 上等待。以后某时刻外存有要换入的进程时,唤醒在 `swapout` 上睡眠的进程园。

(2) 换入算法

对系统中的所有进程,只要其系统为 `swapin`,标志 `swapin` 为 1。根据在外存交换区驻留时间的长短,逐个将它们换入内存,直到全部换入或内存中无足够空间又没有符合换出条件的进程为止。

(3) 换出算法

首先,进程标志 `swapin` 中包含 `swapin` 和 `swapout` 标志的进程不应换出。对其他进程,首先换出那些已部分换出的进程,其次选睡眠或者暂停状态的进程中优先数最大的,然后选可运行的进程中运行时间最长者。另外,换入进程在外存时间要大于 10 秒且运行进程在内存时间要大于 10 秒。

(4) 换入换出操作

换入换出操作分别是由函数 `swapin()` 和 `swapout()` 完成。进程的存储器映像中正文段和共

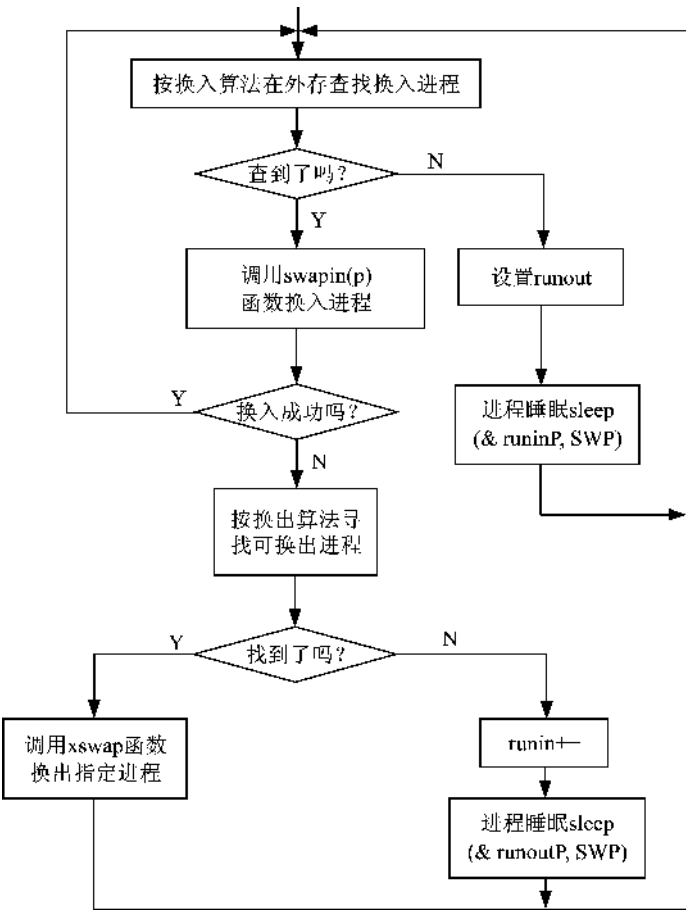


图 猿猿 猿调猿函数 猿调猿流程图

享正文段不随进程换进换出。进程的数据段和栈段随进程换进换出。为了管理上的方便，数据段和栈段在外存对换区集中存放，起始位置存入 猿猿猿猿猿猿猿猿中，换入时首先为进程分配页表页面，为数据段和栈段分配内存页面。然后连入正文段，读入数据段和栈段，并把数据段靠近正文段，空档留在数据段和栈段之间。最后重新附上共享正文段（若有）。换出时首先申请分配外存交换区，然后释放共享正文段，将数据段和栈段集中在一起换到交换区，最后释放数据段和栈段所占内存页面，释放进程页表页面。

本章小结

“调度”是指应用适当的调度策略，合理、有效地将进程或作业投入运行，其根本的目的是在满足功能要求的前提下，使系统资源得到充分的利用。

调度有高级调度、低级调度、中级调度三种。高级调度选择适当的作业进行除处理机以外的资源分配；低级调度决定将处理机分配给哪一个就绪进程；中级调度以实现内存的合理分配和使用为目的。

调度算法考虑两方面的因素：

用户方：用户方的要求体现在两个方面，一是用户总是希望自己的作业或进程在尽可能短的时间之内完成；二是特殊类型的作业能得到合理的优先安排。

资源方：应该对有限的系统资源尽可能均衡、充分地利用起来，达到最高的吞吐率。

以下三种算法从不同的方面考虑了用户在速度方面的要求：

先来先服务算法：系统依次服务进程或作业，不考虑任何特例，后到的进程或作业需要等待。

短作业优先算法：优先照顾短作业，系统资源得到了很好的利用，但长作业可能长时间等待。

时间片轮转算法：系统轮流服务每一个进程一个时间片，后到的进程一定程度上得到了照顾，能在一个时间片内完成的短作业得到了很好的响应。

优先权是一个人为赋予的数值，既可以用来满足用户不同类型的进程的特殊要求，也可以兼顾用户在速度方面的需要。优先权分两种：一种是静态优先权，在进程投入运行时，由进程本身的某些固有属性来决定；另一种是动态优先权，在进程投入运行后，根据进程在系统中所处状态，动态地计算。以优先权为调度依据的常用算法有以下几种：

优先权调度算法：根据作业或进程的不同性质，赋予作业或进程不同的优先权值，使系统能根据优先权值的高低合理调度作业或进程。

高响应比优先调度：在实现对作业的调度时，使优先权随着作业的等待时间的增加而升高，得以优先调度，以避免作业的长时间等待。

多级队列调度：对不同类型的进程赋予静态的优先权，并在同优先权的进程采用时间片轮转来服务。

多级反馈队列调度：以优先权作为调度的依据，用时间片轮转作为基本的调度方法来实现调度。优先权采用动态优先权，综合体现了进程到达的顺序、进程对处理器占用的时间和进程的长短。

进程的调度以优先权为调度依据，优先权包含了静态优先权和动态优先权：

系统设置值：系统根据睡眠原因，设置相应的数值。

作业的长短：对于一个时间片内能够完成的短进程，系统给予了相当的优先。

进程占用处理器的时间：当一个进程新创建时，由于其处理器占有时间为零，因此，它拥有较高的优先级而得到调度。但如果该进程在一个时间片内没有能够完成，优先级按一定速度下降，高优先级的进程将抢占处理机。

等待时间：随着等待时间的增加，优先级随之升高，最终重新获得机会抢占处理机。

习题题

为什么要进行进程调度？详细说明引起进程调度的原因。

区分高级调度、低级调度和中级调度。

下述问题应该由哪一级调度负责：

(1) 在可获得处理器时，应将它分给哪个就绪进程？

(2) 在短期繁重负载下，应将哪个进程暂时挂起？

源 作业和进程有何区别和联系？
缘 作业调度存在于哪种类型的操作系统中，为什么？
远 在操作系统中，引起进程调度的主要因素有哪些？
苑 试述抢占式调度与非抢占式调度之间的区别，并分析各自的特点。
愿 下表是四个作业 粤 月 悦 阅 的到达时间和所需服务时间，计算在
 葬 先来先服务
 遭 短作业优先
 糟 时间片为 缘 时
 凿 时间片为 员 圆 时
三种调度算法下，各个作业的完成时间、周转时间和带权周转时间。

进程名	到达时间	服务时间
粤	园	员 圆
月	员	缘 圆
悦	圆	圆 圆
阅	猿	员 圆

怨 什么是抢占式调度？试比较抢占式调度与非抢占式调度的优先级调度法的要点及各自的优缺点。
员 愿 文中所述的几种调度算法都包含有参数，如时间片轮转法算法中的时间片、多级反馈队列调度算法中的队列个数、优先级和各队列的调度算法以及进程在队列之间移动的规则等。实际上，这些算法通过选取某些变量的不同值可能演变成另一种算法，或者两种算法之间有一种包含关系。试述下列各进程调度算法之间的关系。
 (员 时间片轮转调度算法，先来先服务调度算法；
 (圆 优先权调度算法，短作业 (进程) 优先调度算法；
 (猿 多级反馈队列调度算法，先来先服务调度算法；
 (源 优先权调度算法，先来先服务调度算法；
 (缘 多级队列调度算法，多级反馈队列调度算法。

第 9 章 进程同步与通信

在多道程序环境下，计算机系统存在着多个进程，有些进程在运行时它们彼此之间往往存在一定的联系。一方面，进程间由于互相竞争系统中的有限资源；另一方面，有些进程之间必须相互协作，共同完成某一任务。这种进程间的既竞争又合作的关系表现为进程的互斥与同步关系。本章将主要讨论进程互斥与同步的概念、临界资源与临界区的概念、信号量机制、经典同步问题与管程的概念、进程通信机制。最后对屏蔽系统的信号量机制进行分析。

源域临界区

源環環互斥与同步

多道并发系统中，由于进程的运行往往不是孤立的，进程间的相互联系及资源共享，使进程间存在某种制约关系，因而使进程间要在执行次序上协调，以保证并发程序执行的可再现性。并发执行的进程间可能存在两种形式的制约关系。

员援同步关系

为了完成一个共同任务，相互协作的几个进程需要在某些确定点上协调他们的工作，等待来自其他进程的信息，以调整他们的推进速度，方可顺利执行完毕。我们把进程间的这种制约关系称为直接制约或进程同步（**同步关系**）。

例如，有一个输入进程，一个计算进程，共同完成将数据输入后再进行计算的任务。两个进程共享一个缓冲区，缓冲区中每次只能存放一个数据，输入进程首先输入数据到缓冲区中，然后，计算进程再从缓冲区中取出数据进行处理，输入进程再输入数据到缓冲区中，开始下一个数据的处理，如图 源原员所示。由于缓冲区中每次只能存放一个数据，当输入进程将一个数据输入到缓冲区后，并不能马上输入下一个数据，它必须等待计算进程取走缓冲区中的数据之后才能开始下一个数据的输入。同样，计算进程在输入进程没有向缓冲区中输入数据时它也不能继续执行，而必须等待输入进程输入数据后方可继续向前推进。进程间的这种相互协作、调整各自的运行速度的行为就是我们上面所讨论的同步问题。

圆暖互斥关系

根据资源本身的属性，系统中的资源可以分为两类：一类是共享资源，如内存、磁盘等，各进程可以在一段时间内共同使

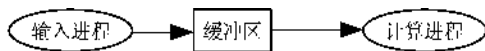


图 源员瑶进程同步示意图

用(但在任意时刻只能有一个进程使用)。另一类称为独占资源,如打印机,进程必须独占使用,一个进程一旦占有打印机就必须让其使用完,释放后其他进程才能使用。试想若有两个进程 P_1 和 P_2 分别完成输出学生成绩与教师工资的任务, P_1 和 P_2 如交替使用打印机,则输出结

引起的。多道系统中,像这种两个或多个进程并发地读写一个共享数据,共享数据的值取决于最后访问它的进程的执行结果的情况称为竞态条件(或称临界区)。为防止竞态条件,保证进程正确地并发执行,必须互斥使用临界资源。

由此可见,对系统中的任意进程,其是否能正确执行,不但取决于程序本身的正确性,而且还与进程执行过程中能否与相关进程实施正确的互斥或同步有关。因此,解决好进程的同步与互斥问题是十分重要的。

临界区

临界区是指在进程中访问临界资源的那段代码。

为保证对临界资源的互斥使用,访问同一临界资源的进程必须互斥地进入各自的临界区。其实现方法是在进入临界区之前,写一段代码以检查可否进入临界区,如果可以进入临界区,通常设置相应“正在访问临界区”标志(或发送相应信息),以阻止其他进程再进入临界区,通常把这段代码称为进入区,把这段代码称为退出区。

进程在进入临界区之后,执行临界区代码,完成对临界资源的访问,并退出临界区。在退出临界区后,为了让其他等待访问临界资源的进程进入临界区,必须有一段代码来清除“正在访问临界区”标志,或发出本进程已经退出临界区的消息,把这段代码称为退出区。

例如,上面示例中的共享变量是一个临界资源,访问它的代码段就是临界区。

为保证对临界资源的互斥使用,在使用临界资源之前应先申请、判断。为此,可把一个访问临界资源的进程描述如下:

```
临界区(员  
  摇 {  
    摇摇 申请进入区 * 转 进入区 * 转  
    摇摇 临界区 * 转 临界区 * 转  
    摇摇 退出区 * 转 退出区 * 转  
    摇摇 剩余区 * 转 剩余区 * 转  
  }  
)
```

同步机构的设计原则

在多道程序系统中,为了保证进程互斥进入临界区,系统需设置专门的同步机构,来协调各进程间的运行。所谓同步机构是指能实现进程同步的机制,该机制能把其他进程需要的信息发送出去,也能测试自己需要的信息是否到达。同步机构应遵循如下原则:

(员) 空闲让进。若其他进程均不处于临界区,则允许申请进入临界区的进程进入。

(圆) 忙则等待。若已有进程处于其临界区,其他要进入临界区(访问相同临界资源)的进程必须等待。

(猿) 有限等待。已进入临界区的进程必须在有限时间内退出,使其他等待进入临界区的进程能在等待有限时间后能进入临界区。

(源) 让权等待。等待进入临界区的进程,应释放处理机,要消除忙等现象。

信号量解决互斥问题的软件方法

为了使访问临界资源的各进程能互斥地进入临界区,下面介绍几种实现互斥的软件算法。

算法 1: 有两个进程 P_1 、 P_2 可以并发执行。设一整型变量 $mutex$ 用 $mutex$ 来标识何进程可进入临界区。初值设为 1,表示 P_1 可以进入临界区,使用临界资源,否则 P_1 必须等待。

在进入区循环检查是否允许本进程进入: $mutex$ 为 1 时,进程 P_1 可进入;在退出区修改允许进入进程标识:进程 P_1 退出时,改 $mutex$ 为进程 P_2 的标识 0。

P_1 结构可以描述为:

过程

repeat { P_1 进入临界区; P_1 退出临界区; }

until P_1 退出临界区;

repeat P_1 进入临界区;

until P_1 退出临界区;

repeat P_1 进入临界区;

until P_1 退出临界区;

这一算法使用单一标志,实现简单。其缺点是:强制各进程轮流进入临界区,没有考虑进程的实际需要,容易造成资源利用不充分:在 P_1 退出临界区之后, P_2 使用临界区之前, P_1 不可能再次使用临界区。

算法 2: 定义标志数组 $flag$ 。元素 $flag[i]$ 用来标识第 i 个进程 P_i 是否在临界区中, $flag[i]$ 初值为 0 表示 P_i 不在临界区中,未使用临界资源。 $flag[i]$ 为 1 表示 P_i 处于临界区中,正在使用临界资源。

P_1 结构可以描述为: P_2 结构可以描述为:

过程

{ P_1 进入临界区; P_1 退出临界区; }

repeat P_1 进入临界区;

until P_1 退出临界区;

repeat P_1 进入临界区;

until P_1 退出临界区;

} P_1 结构;

过程

{ P_2 进入临界区; P_2 退出临界区; }

repeat P_2 进入临界区;

until P_2 退出临界区;

repeat P_2 进入临界区;

until P_2 退出临界区;

} P_2 结构;

这一算法设置双标志,采用在进入区先检查另一个进程是否在临界区,不在时修改本进程在临界区的标志为 1 在退出区将本进程的标志设置为 0。

优点:与算法 1 比较,各进程不用交替进入临界区,进程可连续多次使用临界资源。

缺点: P_1 和 P_2 可能同时进入临界区。例如,按下面顺序执行时,会同时进入: P_1 原 1, P_2 原 1, P_1 原 1, P_2 原 1。即在检查对方 $flag$ 之后和切换自己 $flag$ 之前有一段时间,结果都检查通过,从而导致同时进入临界区。这是由于对标志检查和修改操作是分开进行而造成的。

算法 3: 与算法 2 类似,其区别在于将第 1 行与第 2 行顺序交换,即先修改标志后再检查是否可以进入。这一算法可防止算法 2 存在的多个进程同时进入临界区的问题。

P_1 结构可以描述为 (P_2 的描述略):

过程

repeat { P_1 进入临界区;

圆: 憎嫌(越来越);

猿：糟鹽糖造辣鹽燥土；

源: 蓬萊[蜚越園;

缘：则集字卷之六

摇摇} 憎嫌(员；

这一算法 孕和 孕可能都进入不了临界区。按下面序列执行时,会都进不了临界区:孕原
员 孕原圆 孕原圆 孕原员;即在切换自己 变量之后和检查对方 变量之前有一段时间,结果都切
换 变量,使 孕与 孕都因不能结束第 圆行的测试而无法进入临界区。

算法源: (李永乐讲数学) 算法, 结合算法员和算法猜 得到一个正确的算法。

孕蚤结构可以描述为：

聾

摇摇 { 趑趄[蚤 越员; 踟蹰[越 躁

憎[𢆶]躁(**𢆶**[躁]越越员 驭又𢆶𢆶越越躁；

糟鹽糖造辣薯燥土;

蔓草[蚤 越园;

則集賢館坐鎮則疎舊觀土;

摇摇} 憎翟藁(员；

在进入区先修改后检查，并检查并发修改的先后：检查对方变量，如果不在临界区则自己进入（空闲让进），否则再检查地址。若保存的是较晚的一次赋值，则较晚的进程等待，较早的进程进入（先到先得，后到等待）。

源爱题瑶利用硬件方法解决进程互斥问题

利用上面介绍的软件方法，可以在一定程度上解决临界区互斥问题，但有很大局限性(如不适应于多个进程)，现在已很少采用。

为了解决临界区互斥问题,可以采用锁操作法,即:为每个临界区设置一把锁,锁有打开与关闭两种状态。设锁变量为 `lock`,其开、关锁状态分别用 `0` 和 `1` 来表示。设置关锁原语 `lock_lock` 与开锁原语 `lock_unlock` 来实现关、开锁操作。

进行关锁操作时，若锁处于开状态（`开锁成功`），代表该锁变量所表示的临界资源为空闲，进程将锁变为关状态，即 `进锁成功`，直接进入临界区；否则，若锁已处于关锁状态（`进锁失败`），代表其他进程正在使用临界资源，因而执行关锁操作的进程等待。

进行开锁操作时，将锁状态置为开（开锁），以便其他进程能使用临界资源。

实现进程互斥的锁操作法的关锁与开锁原语可以描述如下：

关锁原语 进餐哲学家

進修藥劑(蛋殼膜藥)

摇摇 { 憎翟藁 (造藁越越员 ;

遵憲起兵；

摇摇}

开锁原语 急进锁G锁：

急燥燥(蛋城燥燥)

```
摇摇 { 进程进入;
```

```
摇摇}
```

利用关锁原语与开锁原语实现临界区互斥时, 只需按如下的形式来实现:

```
摇摇 进程进入(进程); 转 进入区, 关锁 * 转
```

```
摇摇 临界区;
```

```
摇摇 转退出区(进程); 转 退出区, 开锁 * 转
```

由于开锁关锁操作非常简单, 很容易用一条硬件指令来实现。其读写操作由一条指令完成, 也保证了读操作与写操作不被中断。下面介绍两种锁操作的硬件指令实现方式。

1. 检测和设置指令 `test` (读-测试-写)

检测和设置指令描述为:

```
进程(进程) * 进程
```

```
摇摇 { 进程
```

```
    读 * 进程
```

```
    * 进程退出;
```

```
    写 * 进程
```

```
摇摇}
```

`进程`表示资源的两种状态: `员`表示正被占用, `园`表示空闲。利用 `test` 指令实现进程互斥。每个临界资源设置一个公共变量 `进程` 初值为 `园` 在进入区利用 `test` 指令进行检查: 有进程在临界区时, 重复测试, 直到其他进程退出临界区时, 检查通过, 进入临界区。利用 `test` 指令实现进程互斥的方法是:

```
进程(进程) * 园;
```

```
进程
```

```
摇摇 { 进程 (test(进程) != 园; 转 进入区 * 转
```

```
    临界区 * 转
```

```
    进程 * 园; 转 退出区 * 转
```

```
    转其他部分 * 转
```

```
摇摇} 进程 * 园;
```

2. 用 `swap` 指令实现互斥

用 `swap` 指令交换两个字 (字节) 的内容, `swap` 指令的功能描述如下:

```
swap(进程) * 转 进程 * 转
```

```
摇摇 { 进程
```

```
    读 * 转
```

```
    * 转 * 转
```

```
    * 转 * 转
```

```
摇摇}
```

利用 `swap` 实现进程互斥: 每个临界资源设置一个公共变量 `进程` 初值为 `园` 表示资源空闲。每个进程设置一个私有变量 `进程` 初值为 `员` 利用 `swap` 指令实现进程互斥的方法是:

```
进程
```

```
摇摇 { swap(进程, 进程);
```

摇摇 进入区 * 锁
 摇摇 临界区 * 锁
 摇摇 退出区 * 锁
 摇摇 临界区 * 锁

硬件指令方法实现进程互斥的优点：适用于任意数目的进程；在单处理器或多处理器上实现简单，容易验证其正确性；可以支持进程内存在多个临界区，只需为每个临界区设立一把锁（一个共享变量）。

硬件指令方法实现进程互斥的缺点：这种方法在进入区循环检测进程是否可以进入临界区，存在忙等现象，未能实现“让权等待”。另外，这种方式从等待进程中随机选择一个进入临界区，有的进程可能一直选不上，从而可能产生“饥饿”现象。

源信号量

上面介绍的方法在一定程度上可以解决互斥问题，但都存在忙等现象。信号量（ semaphore ）机制就是操作系统提供的解决互斥问题的有效手段。同时，它也能用于解决同步问题。它用“让权等待”方式克服了忙等待的低效。

源信号量概念

整型信号量

1965 年，由荷兰学者 Dijkstra 提出了解决同步问题的更一般的方法，即信号量机制。最初 Dijkstra 把信号量定义为一个整型变量，除初始化外，仅能通过两条同步原语（wait 和 signal）来访问。wait 是荷兰语 等待 的意思和 灾难 的意思的首字母，现在常称为 P 操作和 V 操作。在本书以后的讨论中都是使用 P 操作和 V 操作这一表述。设 S 为信号量，则：

P 操作可描述为：

wait(S)：S--；
 若 S < 0，则进程等待；
 否则，继续执行；

V 操作可描述为：

signal(S)：S++；

信号量机制是一种卓有成效的进程同步机制，在操作系统中被广泛运用，并发展为记录型、信号量集等多种形式。

记录型信号量

在整型信号量机制中的 P 操作，只要是信号量 S > 0 就会不断地测试，因此，该机制并未遵循“让权等待”的准则，而是使进程处于“忙等”的状态，从而浪费了处理机时间。记录型信号量机制，则是一种不存在“忙等”现象的进程同步机制。在采用了“让权等待”的策略后，又会出现多个进程等待访问同一临界资源的情况。为此，在信号量机制中，除了需要一个用于代表资源数目的整型变量 S 外，还应增加一个进程链表 L 用于链接上述的所有等待进程。记录型信号量是由于它采用了记录型（结构体类型）的数据结构而得名的。它所包含的上述两个数据项可描述为如下的结构体类型：

从而保证各进程互斥地进入相关临界区。值得注意的是，对同一信号量操作的 ~~惜与~~ 泽造操作在进程中必须成对出现，否则可能引起死锁。利用信号量实现进程互斥的进程的基本结构可描述如下：

```

泽造信号量 惜与；
责 泽造
摇摇 { 惜与
摇摇 { 惜与； 轱 进入区 * 轱
      糟 泽造； 轱 临界区 * 轱
      泽造 惜与； 轱 退出区 * 轱
      则 泽造； 轱 进程中的剩余部分 * 轱
摇摇 } 惜与；

摇摇}
责 泽造
摇摇 { 惜与
摇摇摇摇 惜与；
摇摇摇摇 糟 泽造
摇摇摇摇 泽造 惜与；
摇摇摇摇 则 泽造
摇摇摇摇 惜与；
摇摇}
摇摇}
    
```

下面通过一个例子来具体说明使用信号量实现进程互斥的方法。有一飞机票售票系统，有皂个售票处，各售票处通过计算机与该系统的票务中心联网，中心数据区中用 砸代表某天某次航班的余票数。进程 责为第 皂个售票处为旅客服务的进程，进程功能如下：

```

责
摇摇 { 接受旅客订票要求；
摇摇摇摇 载越砸； 轱 将票务中心该次航班的余票取出送该进程工作单元 载 * 轱
摇摇摇摇 皂(载跃越员 { 载越载原员；
                        砸越载；
                        输出一张机票；
                    }
摇摇摇摇 泽造输出机票已售完；
摇摇}
    
```

由于皂个售票处的旅客的售票时间、购买机票的日期与航班都是随机的，故存在几个旅客几乎同时在不同的售票处要求订购同天同一次航班的可能。那么，几个进程都要访问票务中心的共享变量 砸，即某次航班的余票。由于各售票处的进程不是协作进程，并发进程执行的相对速度相互无法了解与控制，就可能出现当第 皂个售票处的售票进程 责刚刚取出 砸，执行 载越砸后，第 躁个售票处的售票进程 责马上执行 载越载原员；砸越载；然后 责再执行 载越载原员；砸越载；这样，虽然 责、责都售出了一张机票，但票务中心记录机票余额的变量 砸却

只减了 1。这样 `ticket` 就不能真正反映航班的余票数。这是由于 `ticket` 是共享变量，访问该变量的代码段为临界区，而 `P1`、`P2` 进程没有互斥进入临界区引起的。如果这种情况发生在 `ticket` 减 1 时，将会导致一张票出售给两个不同的旅客。为了避免这种情况的发生，可以设置一个互斥信号量 `mutex`，其初值为 1。在进入临界区之前执行 `wait(mutex)`，退出临界区之后，执行 `signal(mutex)`。进程代码改写如下：

```

wait(mutex);
mutex++;
P1:
while { 接受旅客订票要求；
while wait(mutex);
while wait(ticket); 将票务中心该次航班的余票 ticket 取出送该进程工作单元 ticket；
while wait(mutex) { ticket--;
                    ticket--;
                    mutex--;
                    输出一张机票；
                    }
while mutex {
    mutex--;
    输出机票已售完；
}
P2:
值得注意的是 wait 语句中的两个分支都要加 wait(mutex)；此外，若写成如下形式：
while wait(mutex) { ticket--;
                    ticket--;
                    输出一张机票；
                    mutex--;
                    }
P3:
while {
while 输出机票已售完；
while mutex--;
while

```

虽然系统可以互斥并发执行，但由于“输出一张机票”不是临界区的内容，会降低系统的并发性。

9.3.2 利用信号量实现同步

协作进程为了完成共同的任务，必须进行同步。下面是一个同步模型：假设有两个协作的并发进程 `P1`、`P2`，`S1` 是 `P1` 中的一段代码，`S2` 是 `P2` 中的一段代码，只有 `P1` 中的 `S1` 执行完后 `P2` 中的 `S2` 才能开始执行。为了使 `P1`、`P2` 能顺利执行，在 `P1` 中的 `S1` 执行后要向 `P2` 发送消息，`P2` 接收到 `S1` 已执行完的消息后方可开始执行，否则，`P2` 必须等待。利用信号量的同步原语 `wait` 可以实现 `P1` 向 `P2` 发送消息，用 `signal` 来测试 `P2` 的消息是否到达。设 `S` 是用来实现同步

的信号量，初值为 0 则两个进程的同步可以描述如下：

进程 P1: 摇摇摇摇摇摇进程 P2:

摇摇... 摇摇...

摇摇粤 摇摇粤

摇摇粤; 摇摇粤

摇摇粤; 摇摇粤

摇摇... 摇摇...

在现实生活中，存在许多同步实例，如：公共汽车上司机与售票员之间的活动。司机负责开车，售票员负责开、关车门及售票，其活动如图 9.1.1 所示。虽然各负其责，但必须互相配合、协调他们的活动，他们之间存在如下同步关系：汽车到站，司机停车后售票员方可打开车门让乘客上下车；只有售票员关好车门后，司机才能启动汽车。故司机启动汽车的动作必须与售票员关门动作同步；售票员开门的动作也必须与司机的停车的动作同步。这一同步问题很容易用信号量来实现。

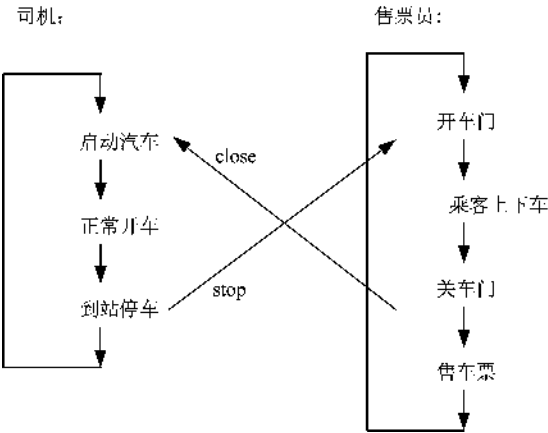


图 9.1.1 司机和售票员同步示意图

设置信号量 S_1 表示车门是否关好，初值为 0 表示门未关好，不允许司机启动汽车；设置信号量 S_2 表示汽车是否停稳，初值为 0 表示未停稳，售票员不能开车门。

S_1 与 S_2 的初值均为 0， S_1 与 S_2 的初值均为 0。

司机 { 启动汽车
正常开车
到站停车
停车了，向售票员发送信息（可以开门了） * S_2

售票员 {

开车门

关车门；

关门后向司机发送门已关的同步信息 * S_1

售车票；

开门前先测试是否停车（是否有来自司机的信息） * S_2

开车门；

乘客上下车；

售票员 }

利用记录型信号量解决生产者—消费者问题

首先，我们分析一下生产者与消费者问题的进程间的同步与互斥关系。

同步关系：如果缓冲池已满，生产者不能再将产品送入缓冲区中，生产者必须等待消费者消费了产品才可继续将产品送入缓冲区；若缓冲区已空，消费者无产品可取，必须等待生产者生产产品后方可继续消费。

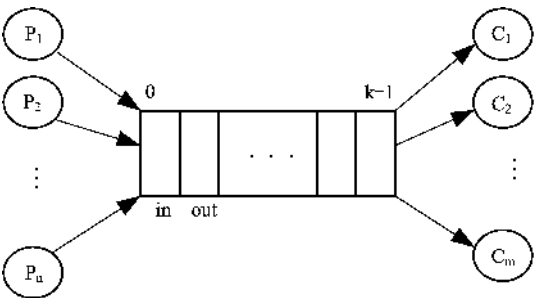


图 源原生产者—消费者问题模型示意图

互斥关系：对所有进程（包括生产者与消费者进程）必须互斥地访问缓冲池。

现假定在生产者和消费者之间的公用缓冲池中，具有 k 个缓冲区，根据上面分析的同步与互斥关系，设置如下信号量：

信号量 $mutex$ ：互斥信号量，实现诸进程对缓冲池的互斥使用，其初值为 1；

信号量 $empty$ ：同步信号量，此信号量表示缓冲池中空闲缓冲区数量，初值为 k ；

信号量 $full$ ：同步信号量，此信号量表示满缓冲区的数量，即缓冲池中生产者投放的产品数量，初值为 0。

设在缓冲池中有 k 个缓冲区，每个缓冲区可存放一个生产者生产的产品， in 为空缓冲区的头指针， out 为满缓冲区的头指针。缓冲区用数组 $buffer$ 描述，数组元素类型 $datatype$ 是生产的产品类型（如数据的数据类型）。

只要缓冲池未满，生产者便可将其生产的产品送入缓冲池；只要缓冲池未空，消费者便可从缓冲池中取走一个产品。

生产者—消费者问题可描述如下：

进程 P_i 的伪代码描述如下：

while (true) {

 wait($empty$);

 wait($mutex$);

 // 生产产品

 // 放入缓冲池

 // 更新 in 指针

 // 通知消费者

 // 等待消费者取走产品

 // 通知生产者

 // 等待生产者生产产品

 // 通知生产者

 // 更新 out 指针

 // 通知生产者

 // 等待生产者生产产品

 // 通知生产者

 // 等待生产者生产产品

 // 通知生产者

 // 等待生产者生产产品

 // 通知生产者

 // 等待生产者生产产品

 // 通知生产者

摇摇}

进程 P1)

摇摇 { 临界区

摇摇摇摇 { 临界区;

临界区;

临界区或临界区;

临界区(临界区 豫 噪

临界区;

临界区;

临界区(临界区 豫 噪

摇摇摇摇}

摇摇}

在生产者—消费者问题中应注意：首先，在每个进程中用于实现互斥的临界区和临界区操作必须成对地出现。其次，对资源信号量 R 和 $count$ 用于实现同步，对它们的 P 和 V 操作，同样需要成对地出现，但它们分别处于不同的进程中。例如， $P(R)$ 在生产者进程中，而 $V(R)$ 则在消费者进程中，生产者进程若因执行 $P(R)$ 而阻塞，则以后将由消费者进程将它唤醒。最后，在每个进程中的多个 P 操作顺序不能颠倒。应先执行对资源信号量的 P 操作，然后再执行对互斥信号量 M 的 P 操作，否则可能引起进程死锁。例如：在生产者进程中，如改为 $P(M)$ ； $P(R)$ ；当缓冲池已满，且无进程访问缓冲池，即 $M=0$ ， $R=0$ 时，若生产者执行 $P(M)$ ，使 $M=0$ ；再执行 $P(R)$ ；此时，由于 $R=0$ 而将该进程阻塞，必须等待消费者取走一个消息，才能唤醒生产者进程。但是由于消费者进程在执行 $P(M)$ 会被阻塞，等待生产者唤醒，无法完成消费任务，从而造成了生产者与消费者互相等待的这样一个僵局，即死锁。

图 9.1 利用信号量解决生产者—消费者问题

在上面的解决方法中，进程中的两个 P 操作的位置不能颠倒，否则，可能导致死锁，使用信号量机制来实现，将对信号量 R 与 M 的两个 P 操作用一个同时 P 操作 $P(R, M)$ 来实现，就可以避免死锁的产生。实现如下：

进程 P1: $P(R, M)$ ； $P(R)$ ；

进程 P2: $V(R)$ ；

进程 P3: $V(M)$ ；

进程 P4)

摇摇 { 临界区

...

临界区(临界区 豫 噪

...

临界区(临界区 豫 噪


```

读进程 Pi
{
    while (true)
    {
        while (读信号量 == 0) ;
        读信号量--;
        读临界区;
        读信号量++;
        ...
        读临界区;
        ...
        读信号量--;
        读临界区;
        读信号量++;
        读临界区;
    }
}

写进程 Wj
{
    while (true)
    {
        while (写信号量 == 0) ;
        写信号量--;
        写临界区;
        写信号量++;
    }
}

```

考虑：能否将读进程直接写成如下形式：

```

读进程 Pi
{
    while (true)
    {
        while (读信号量 == 0) ;
        读信号量--;
        读临界区;
        读信号量++;
        ...
    }
}

```

若写成上面的形式，将导致每次只能有一个读进程进行读操作，这与问题的要求不一致。

考虑：

(1) 若最多允许 n 个读者同时读，应如何实现？

(2) 若规定写者的优先权高于读者，即在有写者等待写的情况下，未开始读的读者必须等待，又如何实现？请读者考虑。

若一个写者已进入临界区执行，并且有写者和几个读者正在等待，则只有一个读者阻塞在写信号量上，其余读者都阻塞在读者信号量上。此外，当写者退出临界区，执行了 `读者信号量++` 后，唤醒写者进程还是读者进程则取决于进程调度策略。

图源利用信号量集机制解决读者—写者问题

使用信号量集机制来解决这一问题将更加简洁。设置两个信号量：读表示读者数信号量，初值为 0，表示最多允许同时有 0 个读者；写为互斥信号量，初值为 1 实现读者与写者、多个写者之间的互斥。读者—写者问题的并发算法可描述如下：

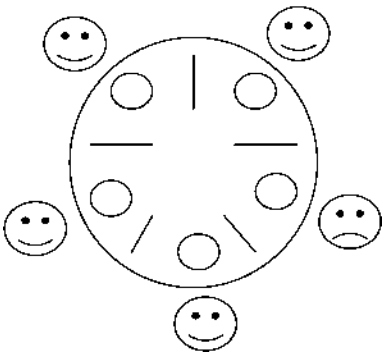
```
semaphore 读 = 0, 写 = 1;
读者进程()
{
    读++;
    写--;
    ...
    读--;
    写++;
}
写者进程()
{
    写--;
    读--;
    ...
    读++;
    写++;
}
```

图源哲学家进餐问题

哲学家进餐问题（就餐问题）由图源首先提出并予以解决。问题描述为：5 个哲学家围绕一张圆桌而坐，桌子上放着 5 支筷子，每两个哲学家之间放一支；哲学家的动作包括思考和进餐，进餐时需要同时拿起他左边和右边的两支筷子，进餐完毕则同时将两支筷子放回原处，如图源所示。如何保证哲学家们的动作有序进行？如：不出现相邻者同时进餐；不出现有人永远拿不到筷子。

经分析可知，放在桌子上的筷子是临界资源，在一段时间内只允许一位哲学家使用。为了实现对筷子的互斥使用，可以用一个互斥信号量表示一只筷子，由这五个信号量构成信号量数组。第 i 个哲学家的活动描述如下：

```
哲学家进程(i)
{
    思考();
    进餐();
}
```



图源哲学家进餐问题示意图

第 i 个哲学家的活动描述如下：

管程由三部分组成：① 局部于管程的共享变量说明；② 对该数据结构进行操作的一组过程，即操作原语；③ 对局部于管程的数据设置初始值的语句。此外，还须为管程赋予一个名字，如图 9-1 所示。

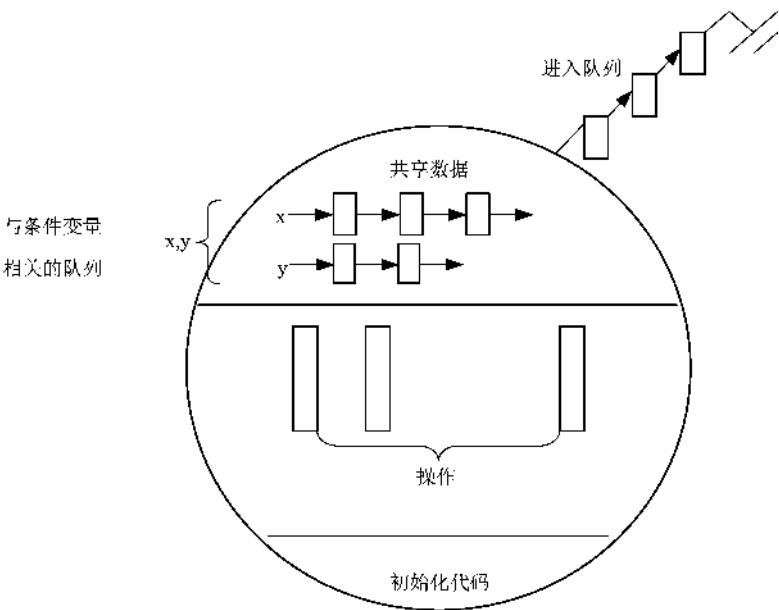


图 9-1 管程示意图

管程的结构如下：

```
monitor 名称 {
    共享变量说明;
    操作原语过程名 (...);
    { ... };
    操作原语过程名 (...);
    { ... };
    ...
    操作原语过程名 (...);
    { ... };
    { 初始化代码 };
}
```

说明

管程有点像面向对象程序设计中的“类”，局部于管程的共享变量只能由局部于管程内的过程（或函数）所访问，管程外的过程不能对其进行操作。而管程内的过程也只能访问管程内的数据结构。管程中对控制变量和临界资源进行操作的一组原语过程（程序代码），是访问该管程的惟一途径。这些原语本身是互斥的，任一时只允许一个进程去调用，一旦有进程在访问管程，其余需要访问临界资源的进程就必须等待。

圆暖条件变量

在实现进程同步过程中,由于进程等待的原因很多,为了区分进程的等待原因,引入若干局限于管程且仅能从管程内进行访问的条件变量(精世器出器器器,以及对条件变量进行同步操作的原语(类似于信号量操作原语 惜器器 泽器器。每个条件变量表示一种等待原因,它并不取具体数值,而相当于每个原因对应一个队列。

管程中对每个条件变量,都须予以说明,其形式为:精世器器器 赠

曾 赠是两个条件变量。对条件变量只能使用 惜器器与 泽器器进行操作,同步操作原语形式变为:条件变量援器器条件变量援器器即条件变量置于 惜器器和 泽器器之前。针对条件变量曾 惜器器与 泽器器原语可分别表示为 曾器器与 曾器器。

原语 曾器器将调用该原语的进程自己阻塞在条件变量 曾相应的队列中,直到有另外一个进程调用了 曾器器为止。

原语 曾器器的作用是将阻塞在条件变量 曾队列中的一个进程唤醒,但如果没有被阻塞的进程,则 曾器器操作什么也不干,这与信号量机制中的 泽器器操作有所不同。因为,后者总是要执行 泽器器操作,因而总会改变信号量的状态。

设有进程 孕 匹,如果进程 匹处于阻塞状态,当进程 孕执行了 曾器器操作后,进程 孕唤醒进程 匹,那么如何决定由哪个进程执行,哪个等待?可采用下面的两种执行方式之一进行处理:

(员 孕等待,直至 匹离开管程或等待另一条件。

(圆 匹等待,直至 孕离开管程或等待另一条件。

源源摇摇管程的实现

在利用管程方法来解决生产者—消费者问题时,首先便是为它们建立一个管程,并命名为 孕悦 问题仍然设定有 噪个缓冲区的缓冲池,每个缓冲区可存放一个生产者生产的产品(数据项),蚤是空缓冲区的头指针,燥是满缓冲区的头指针,并用整型变量 精器器来表示当前在缓冲池中已有的产品数量(满缓冲区的数量)。在生产者 原消费者问题中,若没有空缓冲区,生产者不能继续向缓冲池投放其生产的产品,生产者进程必须等待,为此,用 灶器器作为没有空缓冲区的条件变量。同样,没有满缓冲区,消费者进程不能从缓冲池中取产品消费,为此,用 晕器器作为没有满缓冲区(无产品在缓冲池中)的条件变量。管程中,生产者和消费者调用的管程原语分别是:孕悦器器,孕悦器器。

(员 器器原语。生产者利用该原语将自己生产的产品投放到缓冲池中,当 精器器运行时,表示缓冲池已满,生产者须等待,通过 灶器器实现。

(圆 器器原语。消费者利用该原语从缓冲池中取出一个产品,当 精器器园时,表示缓冲池中已无可取用的产品,消费者应等待,通过 灶器器实现。

管程定义如下:

器器器器器

摇摇 { 蚤器器 燥器器 精器器

摇摇摇摇 蚤器器再器器器器; 精 蚤器器再器器器器为产品 蚤器器的数据类型 * 精

摇摇摇摇 精器器器器器器器 灶器器

摇摇摇摇 器器器

```

// 生产者
while (true) {
    // 生产产品
    product++;
    // 如果缓冲区满了，就等待
    while (buffer == 0)
        continue;
    // 放入缓冲区
    buffer--;
    // 通知消费者
    pthread_mutex_t * mutex = &mutex;
    pthread_mutex_lock(mutex);
    // 从缓冲区取产品
    consumer++;
    // 通知生产者
    pthread_mutex_t * mutex = &mutex;
    pthread_mutex_unlock(mutex);
}

// 消费者
while (true) {
    // 从缓冲区取产品
    consumer--;
    // 如果缓冲区空了，就等待
    while (buffer == 0)
        continue;
    // 取出产品
    buffer++;
    // 通知生产者
    pthread_mutex_t * mutex = &mutex;
    pthread_mutex_lock(mutex);
    // 初始化变量 * 变量
    pthread_mutex_t * mutex = &mutex;
    pthread_mutex_unlock(mutex);
}

```

在利用管程解决生产者-消费者问题时，其中的生产者和消费者可描述为：

```

// 生产者
while (true) {
    // 生产产品
    product++;
    // 如果缓冲区满了，就等待
    while (buffer == 0)
        continue;
    // 放入缓冲区
    buffer--;
    // 通知消费者
    pthread_mutex_t * mutex = &mutex;
    pthread_mutex_lock(mutex);
    // 从缓冲区取产品
    consumer++;
    // 通知生产者
    pthread_mutex_t * mutex = &mutex;
    pthread_mutex_unlock(mutex);
}

// 消费者
while (true) {
    // 从缓冲区取产品
    consumer--;
    // 如果缓冲区空了，就等待
    while (buffer == 0)
        continue;
    // 取出产品
    buffer++;
    // 通知生产者
    pthread_mutex_t * mutex = &mutex;
    pthread_mutex_lock(mutex);
    // 初始化变量 * 变量
    pthread_mutex_t * mutex = &mutex;
    pthread_mutex_unlock(mutex);
}

```

进程通信

进程通信是指进程之间的信息交换。前面介绍的进程互斥与同步就是一种进程间的通信方式，只不过交换的信息量较少且效率较低，用户使用也不方便，因此称这种通信方式为低级进程通信方式，称 P、V 原语为两条低级进程通信原语。

本节介绍进程间的高级通信方式，高级进程通信方式是指进程之间以较高的效率传送大量数据的通信方式。

高级进程通信的类型

目前高级进程通信方式可分为三大类：共享存储器系统（共享存储系统）、管道通信系统（管道系统）以及消息传递系统（消息传递系统）。

共享存储器系统

在存储器中划出一块共享存储区，多个进程可以通过对共享存储区进行读或写来实现通信。进程在通信前，向系统申请建立一个共享存储区，并指定该共享存储区的关键字；若该共享存储区已经建立，则将该共享存储区的描述符返回给申请者。然后，申请者把获得的共享存储区接到进程上。这样，进程便可以像读写普通存储器一样地读写共享存储区。

这种通信模式需要解决两个问题：一个是怎样提供共享内存，另一个是共享内存的读写互斥问题。操作系统一般只提供要共享的内存空间，处理进程间的互斥关系则是程序开发人员的责任。

管道通信系统

管道通信由 **Unix** 首创，作为 **Unix** 的一大特色立即引起了人们的兴趣。由于其有效性，一些系统继 **Unix** 之后相继引入了管道技术。管道通信是一种重要的通信方式。

管道通信以文件系统为基础。所谓管道，就是连接两个进程之间的一个打开的共享文件，专用于进程之间进行数据通信。发送进程可以源源不断地从管道一端写入数据流，每次写入的信息长度是可变的；接收进程在需要时可以从管道的另一端读出数据，读出单位长度也是可变的。

在对管道文件进行读写操作过程中，发送进程和接收进程要实施正确的同步和互斥，以确保通信的正确性。管道通信机制中的同步与互斥都由系统自动进行，对用户是透明的。管道通信的实质是利用外存来进行数据通信，故具有传送数据量大的优点，但通信速度较慢。

消息传递系统

在消息传递系统中，进程间的数据交换以消息为单位，系统提供发送消息与接收消息两个操作，进程间则通过使用这两个操作进行通信，无需共享任何变量。这样操作系统隐藏了通信的实现细节，从而大大简化了通信程序编制的复杂性，因而获得了广泛的应用。消息传递系统因其实现方式不同可以分为两种方式：直接通信和间接通信。

直接通信

发送进程直接把消息发送给接收进程，并将它挂在接收进程的消息缓冲队列上，接收进程从消息缓冲队列中取得消息。这种通信方式又称为消息缓冲通信。

消息缓冲通信技术是由 **Unix** 首先提出的，其基本思想是：根据“生产者—消费者”原理，利用内存中公用消息缓冲区实现进程之间的信息交换。

内存中开辟了若干消息缓冲区，组成消息队列，用以存放消息。每当一个进程（发送进程）向另一个进程（接收进程）发送消息时，便向系统申请一个消息缓冲区，并把已准备好的

消息送到缓冲区，然后把该消息缓冲区插入到接收进程的消息队列中，最后通知接收进程。接收进程收到发送进程发来的通知后，从本进程的消息队列中摘下一消息缓冲区，取出所需的信息，然后把消息缓冲区还给系统。系统负责管理公用消息缓冲区以及消息的传递。一个进程可以给若干个进程发送消息，同样，一个进程可以接收不同进程发来的消息。显然，进程中关于消息队列的操作是临界区。当发送进程正往接收进程的消息队列添加一条消息时，接收进程不能同时从消息队列取消息，反之也一样。

消息缓冲通信机制包含下列内容：

(1) 消息缓冲区，这是一个数据结构，由以下几项组成

- 指向发送进程的指针
- 指向下一个消息缓冲区的指针
- 消息长度
- 消息正文

(2) 消息队列头指针，一般保存在 PCB 中。

(3) 互斥信号量，初值为 1，用于互斥访问消息队列，在 PCB 中设置。

(4) 同步信号量，初值为 0，用于消息计数，在 PCB 中设置。

发送进程在发送消息前，先在自己的内存空间设置一个发送区，把欲发送的消息填入其中，然后再用发送原语将其发送出去。接收进程则在接收消息之前，在自己的内存空间内设置相应的接收区，然后用接收原语接收消息。两个进程进行通信的过程如图 9-1 所示。

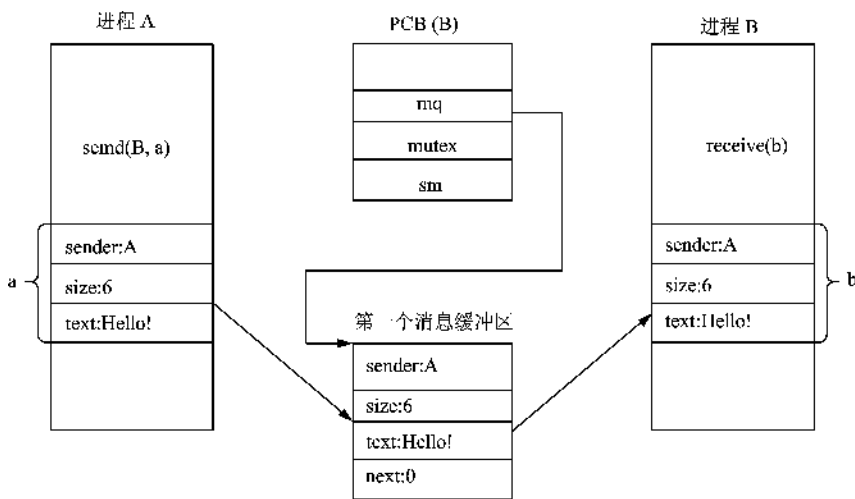


图 9-1 消息缓冲通信

发送原语的功能是把欲发送的消息从发送区复制到消息缓冲区，并将它挂在接收进程的消息队列末尾。如果接收进程因等待消息而处于阻塞状态，则将其唤醒。其工作流程如下：

```
发送原语(进程标识符, 接收者标识符, 发送区首址) * 发送区首址
{
```

```

摇摇向系统申请一个消息缓冲区；
摇摇将发送区消息送入新申请的消息缓冲区；
摇摇将消息放入缓冲区；
摇摇把消息缓冲区挂入接收进程的消息队列；
摇摇将接收区消息放入缓冲区；
摇摇将接收区消息放入缓冲区；
}

```

接收原语的功能是把消息从消息缓冲区复制到接收区，然后将消息缓冲区从消息队列中移出，如果没有消息可读取，则进入阻塞状态。其工作流程如下：

```

摇摇为发送者标识号，遭为接收区首址 * 辕
{
摇摇将接收区消息放入缓冲区；
摇摇将接收区消息放入缓冲区；
摇摇从消息队列中找到要接收的消息；
摇摇从消息队列中摘下此消息；
摇摇将接收区消息放入缓冲区；
摇摇将消息复制到接收区；
摇摇释放消息缓冲区；
}

```

圆间接通信

发送进程把消息发送到某个中间实体中，接收进程从中取得消息。这种中间实体一般称为信箱，这种通信方式又称为信箱通信方式。

信箱是一种数据结构，其中存放信件。当一个进程（发送进程）要与另一个进程（接收进程）通信时，可由发送进程创建一个链接两进程的信箱，通信时发送进程只需把它的信件投入信箱，接收进程就可以在任何时候取走信件。

信箱逻辑上分成信箱头和信箱体两部分。信箱头中存放有关信箱的描述，信箱体由若干格子组成，每格存放一个信件，格子的数目和大小在创建信箱时确定。信件的传递可以是单向的，也可以是双向的。

在单向信箱通信方式中，只要信箱中有空格，发送进程便可以向信箱中投递信件，若所有格子都已装满，则发送进程等待或者继续执行，待有空格子时再发送。类似地，只要格子中装有信件，接收进程便能取出一个信件。若信箱为空，接收进程等待或者继续执行。

在双向通信方式中，信箱中既有发送进程发出的信件，也有接收进程的回答信件。由于发送进程和接收进程是以各自独立的速度向前推进的，当发送进程发送信件的速度超过接收进程的接收速度时，会产生上溢（信箱满）。反之，会产生下溢（即接收进程向空信箱索取信件）。这就需要在两个进程之间进行同步控制，当信箱满时发送进程应等待，直至信箱有空格子时再发送；对接收进程，当信箱空时，它也应等待，直至信箱中有信件时再接收。

信箱通信方式中也使用原语操作，如创建信箱原语、撤销信箱原语、发送与接收原语等。另外，在许多时候，存在着多个发送进程和多个接收进程共享信箱的情况。

消息通信中的同步问题

进程间的消息通信隐含着某种同步关系：只有当一个进程发送出消息之后，接收进程才能接收消息。对于一个发送进程来说，它在执行发送原语后（即发送完消息后），有两种可能选择：

（1）发送进程阻塞自己，直到收到接收进程回答消息后才继续向前执行，这种发送称为阻塞发送。

（2）发送进程不阻塞，不等回答继续执行，这种发送称为非阻塞发送。

同样，对于一个接收进程来说，在执行接收原语后，也有两种可能选择：

（1）如果一个消息在接收原语执行之前已经被发送，则该消息被接收进程接收，接收进程继续执行。

（2）如果没有正在等待的消息，则它或者阻塞自己等待一个消息到达，或者干脆放弃接收的努力，继续执行。前者称为阻塞接收，后者称为非阻塞接收。

因此，发送进程和接收进程都可以阻塞或不阻塞。根据发送进程和接收进程采取方式的不同，通常有三种组合方式，但任何一个特定系统只会实现其中的一种或两种。

（1）非阻塞发送、阻塞接收。这是最常用、最自然的模式。这种非阻塞发送的方式便于发送进程尽快地向多个进程发送一个或多个消息，同时这种不阻塞发送也适合客户进程在提出输出请求后，继续向前执行，不需阻塞等待打印请求。这种阻塞接收的方式也特别适用于那些不等待消息到来就无法进行后续工作的进程，如等待服务请求到来的服务器进程的工作情况和等待资源（硬资源和软资源）的进程。但不阻塞发送的方式也存在隐患。它可能导致有意或由于错误而造成发送进程反复不断地发送消息，造成大量资源（时间和缓冲区空间）浪费。

（2）非阻塞发送、非阻塞接收。这是分布式系统常见的通信方式，因为采用阻塞接收方法时，如果发送来的消息丢失（这在分布式系统中常发生），或者接收进程所期待的消息在未发送之前，发送进程就发生问题了，那么阻塞接收方式将导致接收进程无限期被阻塞。而改进的办法就是使用非阻塞接收方式，即接收进程在接收消息时，若有消息就处理消息，若没有消息就继续执行，放弃接收的努力。

（3）阻塞发送，阻塞接收。发送进程在发送完消息后，阻塞自己等待接收进程发送回答消息后才能继续向前执行。接收进程在接收到消息前，也阻塞等待，直到接到消息后再向发送进程发送一个回答信息。

信号量机构分析

在操作系统中采用了信号量集机制，并作了较大扩充。传统的信号量机制是对信号量施加增量和减量操作，而在操作系统中是利用系统调用来对指定信号量进行操作。它允许对一组信号量（即由一组信号量构成的信号量集）进行相同或不同操作，使信号量机制具有更大的灵活性。

信号量集数据结构

（1）信号量与信号量表

信号量的数据结构定义为：

```

信号量 {
    int value; /* 信号量值 */
    int pid; /* 最近一次对信号量进行操作的进程号 */
    int count; /* 等待信号量值增加的进程数 */
    int limit; /* 等待信号量值为 0 的进程数 */
};
    
```

信号量表是由一个或多个信号量构成的结构体数组，或称为一个信号量集。

（四）信号量集表的数据结构

系统中每个信号量集用一个对应的信号量索引结构来描述，数据结构定义如下：

```

信号量索引 {
    int flags; /* 信号量访问权限 */
    int *p; /* 指向信号量组中的第一个的指针 */
    int n; /* 一个信号量集合中的信号量个数 */
    int time; /* 最后一次调用 sem_wait() 对信号量进行操作的时间 */
    int modtime; /* 最后一次调用 sem_wait() 对信号量状态的修改时间 */
};
    
```

信号量集表是每个信号量集索引（信号量索引）组成的一个结构体数组，每个表目（信号量集索引数组元素）对应一个信号量集。

（五）信号量集的建立

进程在使用信号量之前，需通过系统调用 `sem_init` 来创建新信号量集或打开现有信号量集。其原型如下：

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

其中 `sem` 是用户进程指定的信号量集名；`value` 定义了信号量集中信号量的数目；而 `pshared` 为访问标志；函数的返回值是信号量集的描述符（标识号 `sem_id`）。

进程调用 `sem_init`，核心将完成以下工作：

（1）先检查用户的请求是否合法，即用户是否具有建立信号量集的权限，所要求建立的信号量的数目是否合法。此外，还须检查系统当前是否有足够的资源。如能通过上述检查，便可为用户建立信号量集。

（2）根据系统调用中给定的参数 `value` 分配用于存放信号量集的数据结构，并对它们进行初始化，最后返回信号量集的描述符。

（3）若用户使用系统调用的目的是为了获得信号量集的描述符，则当 `value` 不为 0 且也不大于该信号量集中的信号量数目时，内核才返回信号量集的描述符。

（六）信号量集的操作

一旦信号量集被创建成功，便可利用 `sem_wait` 对这组信号量进行操作。`sem_wait` 的原型为：

```
int sem_wait(sem_t *sem, int *pshared);
```

其中，`sem` 是 `sem_init` 返回的描述符；`pshared` 是指向信号量集结构数组的指针；`size` 是该数组的大小。

`sem_wait` 描述如下：

```
void sigwaiter_t {
```

```
    int sig; // 信号量序号 * 信号量
```

```
    int op; // 要执行的操作：等待或阻塞 * 信号量
```

```
    int flag; // 访问标志 * 信号量
```

```
};
```

其中，`sig` 是信号量的编号，即在信号量集中的序号，它标识要操作的信号量数组的元素；`op` 为所要进行的操作，即 `wait` 或 `block`；`flag` 为标志，如 `sigwaiter_t` 标志表示立即返回等。

内核根据 `sig` 的值来改变信号量的值，可分下面三种情况：

(1) `sig` 的值为正，则将其值加到信号量的值上，并唤醒所有等待此信号量增加的进程，它相当于通常的 `wait` 操作。

(2) `sig` 的值为负，它相当于通常的 `block` 操作。若信号量的值大于操作值 (`sig` 的值的绝对值)，则内核将一个负整数加到信号量上；否则，内核让进程睡眠在等待信号量值增加到达事件上。但是，如果 `sig` 中设置了 `sigwaiter_t` 标志，则立即返回一个错误码。

(3) `sig` 为 0，则是进程等待，直至信号量值为 0，但该操作不改变信号量值。如果 `sig` 中设置了 `sigwaiter_t` 标志且信号量值不为 0，则系统调用立即返回一个错误码。

9.3 进程通信机构 进程

UNIX 提供有名管道和无名管道两种数据通信方式，无名管道用于密切相关的父子进程或兄弟进程之间的通信，而有名管道用于一般的无家族关系的进程间通信。我们这里介绍无名管道。

无名管道为建立管道的进程及其子孙提供一条以比特流方式传送消息的通信管道。该管道在逻辑上被看作管道文件，在物理上则由文件系统的高速缓冲区构成而很少启动外设。发送进程利用文件系统的系统调用 `write`，把 `write` 中的长度为 `size` 字符的消息送入管道入口 `fd`，接收进程则使用系统调用 `read`，从管道出口 `fd` 读出 `size` 字符的消息置入 `read` 中。这里，管道按 `FIFO` (先进先出) 方式传送消息，且只能单向传送消息，如图 9-1 所示。

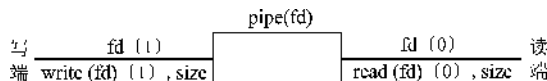


图 9-1 管道通信

利用系统调用 `pipe` 可建立一条同步通信管道。其格式为：

```
int pipe(int fd[2]);
```

```
int pipe2(int fd[2], int flags);
```

这里，`fd[0]` 为写入端，`fd[1]` 为读出端。

例：用 语言编写一个程序，建立一个 同时父进程生成一个子进程，向 中写入一字符串，父进程从 中读出该字符串。

```

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <semaphore.h>

#define PIPE_SIZE 1024
#define SEM_NAME "/mysem"

int main()
{
    sem_t sem;
    int fd;
    char buf[PIPE_SIZE];

    if (sem_init(&sem, 0, 1) != 0) {
        perror("sem_init");
        return 1;
    }

    fd = pipe();
    if (fd[0] == -1 || fd[1] == -1) {
        perror("pipe");
        return 1;
    }

    if (fork() == 0) {
        // Child process
        close(fd[0]);
        write(fd[1], "Hello, World!\n", 15);
        close(fd[1]);
        _exit(0);
    }

    // Parent process
    close(fd[1]);
    read(fd[0], buf, PIPE_SIZE);
    printf("Received: %s\n", buf);
    close(fd[0]);

    wait(NULL);
    sem_destroy(&sem);
    return 0;
}
    
```

本章小结

在多道程序系统中进程是并发执行的，由于共享资源与协作，这些进程之间存在着不同的相互制约关系。其一是直接制约关系，即同步关系。所谓进程同步指的是多个相互合作的进程为了完成共同的任务需要在某些确定点协调它们的工作而相互等待，相互交换信息产生的制约。其二是间接制约关系，即互斥关系，指的是进程之间因相互竞争独占资源而产生的制约。进程互斥是进程同步的一种特殊情况。

在系统中虽然多个进程可以共享各种资源，但许多资源每次只允许一个进程使用。把这种一次仅允许一个进程访问的资源称为临界资源，而每个进程中访问临界资源的那一段代码称为临界区。进程必须互斥访问临界资源或互斥进入临界区。

为了实现进程同步与保证进程互斥访问临界资源，必须采用相应的同步机构。同步机构要能遵循“空闲让进、忙则等待、有限等待”三大准则。信号量机制是一种有效的同步机制，在操作系统中被广泛使用。信号量是指除了被初始化外只能由两个操作原语 P 与 V 才能对其进行操作的整型或记录型变量。由最初的整型信号量发展为记录型、 semaphore 型与一般信号量集多种类型。用信号量机制可以实现进程互斥、进程同步及描述前趋关系。在计算机系统中存在许多的同步问题，生产者—消费者问题、读者—写者问题、哲学家问题是三个典型

的同步问题模型,了解这些问题的解决方法有利于解决计算机系统的同步问题。

由于信号量机制自身的问题,使用时会带来不方便,管程是另一类同步机制。管程定义了一个数据结构和能为并发程序所执行的一组操作,这组操作能同步进程和改变管程中的数据。管程由三部分组成:① 局部于管程的共享变量说明;② 对该数据结构进行操作的一组过程,即操作原语;③ 对局部于管程的数据设置初始值的语句。管程简化了复杂同步问题的实现方法,是一种方便、有效、安全的同步工具。

进程通信机制由一组通信原语组成,进程之间通过通信原语来传递大量信息,其中消息传递是最基本的通信原语。进程高级通信机构包括共享内存,消息传递和通道通信。

读者-写者问题中采用了信号量集机制,并作了较大扩充。与传统的信号量机制不同的是它利用 $\text{wait}()$ 系统调用来对指定信号量进行操作,而不是使用 wait 和 signal 操作。它允许对一组信号量(即由一组信号量构成的信号量集)进行相同或不同操作,使信号量机制具有更大的灵活性。

习 题

1. 在多道系统中,进程间存在哪两种制约关系?

2. 什么是临界资源、临界区?请详述临界区的使用原则。

3. 简述信号量的物理意义,其初值如何确定?

4. 进程之间的互斥是如何通过信号量实现的?

5. 两个并发进程的程序如下:

```
进程 P1:

```

```
wait(mutex);

```

```
critical_section;

```

```
wait(mutex);

```

```
critical_section;

```

```
signal(mutex);

```

```
critical_section;

```

```
critical_section;

```

```
wait(mutex);

```

```
critical_section;

```

```
signal(mutex);

```

```
critical_section;

```

请回答:

(1) 指出这两个并发进程的临界区。

(2) 指出它们并发执行时可能出现的与时间有关的错误。

(3) 用信号量机制进行管理,写出使它们能正确并发执行的程序。

理发师睡觉问题:理发店由等待间(若干个座位)和理发间(一个理发椅)构成。无顾客时,理发师睡觉。当顾客进入理发店,发现没有空座位,顾客离开;发现理发师睡觉时,则唤醒理发师并理发;否则,顾客找个座位坐下等待。理发师在给一个顾客理完发,若发现有顾

客等待，继续给下个顾客理发，否则睡觉。请用信号量机制写出模拟理发师和顾客行为的算法。

题假设三个并发进程 P_1 、 P_2 、 P_3 其中 P_1 负责从输入设备上读入信息并传给 P_2 ， P_2 将信息加工后传给 P_3 ， P_3 则负责将信息打印输出。设： P_1 、 P_2 共享一个缓冲区， P_2 、 P_3 共享另一个缓冲区，一个缓冲区可保存一个数据信息，请用信号量机制写出 P_1 、 P_2 、 P_3 的并发算法。

题有一水果盘，每次只能装一个水果。哥哥有苹果、梨两种水果，若盘子为空，每次放一个水果在盘子里；弟弟取盘子中的梨，妹妹取盘子中的苹果。使用信号量机制来描述哥哥、弟弟、妹妹的同步关系。

题某车站售票厅，任何时刻最多可容纳 N 名购票者进入，当售票厅中少于 N 名购票者时，则厅外的购票者可立即进入，否则需在外面等待。若把一个购票者看作一个进程，请回答下列问题：

(1) 用信号量机制操作管理这些并发进程时，应怎样定义信号量，写出信号量的初值以及信号量各种取值的含义。

(2) 若欲购票者最多为 M 个人，写出信号量可能的变化范围（最大值和最小值）。

(3) 为保证进程能够正确地并发执行，用信号量机制来描述进程的同步关系。

题有某单向行车道路段 AB ，为保证行车的安全需设计一个自动管理系统，管理原则如下：

当 AB 段之间无车行驶时，可让到达 A 点或 B 点的车进入 AB 段行驶；当 AB 段有车行驶时，同向车可以通过，反向车等待。把每一辆需经过 AB 段的车辆看作是一个进程，用信号量机制描述车辆的管理过程。

题什么是管程？使用管程机制解决哲学家进餐问题。

题什么是进程通信？有几种类型？

题消息通信中要设置哪些基本通信原语，如何实现？

题戴的管道通信是如何实现的？

第 9 章 死锁

早期的操作系统对申请某种资源的进程，若该资源尚未分配时，立即将资源分配给这个进程。后来发现，对资源分配不加限制可能导致进程间由于竞争资源而相互制约，以致无法继续运行的局面，人们把这种局面称为死锁（Deadlock）。死锁问题直到 1965 年，才被 Dijkstra 首先认识。1965 年，Dijkstra 在评论 Multics 操作系统时说：“原先多任务的概念是让多个任务不加限制地竞争资源……但是随着系统的发展，很多任务被锁在系统中了。”1968 年，Dijkstra 说：“1965 年，我们设计 Multics II 系统时并没有认识到死锁问题，系统也没有任何防范措施，结果现在一些程序已被锁在系统中。”那么，什么是死锁？死锁在系统中是怎样产生的？人们用什么方法来解决死锁？这些正是本章要讨论的问题。

死锁的概念

死锁的定义

在多道程序运行的系统中，由于多个进程的并发执行，改善了系统资源的利用率并提高了系统的处理能力。然而，也带来了新的问题，也就是可能会出现这样一种危险：当某一进程提出资源的使用要求后，使得系统中一些进程处于无休止的阻塞状态，在无外力的作用下，这些进程永远也不能继续前进，我们称这种现象为死锁。那么，在一个进程集合中，若每个进程都在等待某些事件（指释放事件）的发生，而这些事件又必须由这个进程中的某些进程来产生，就称该进程集合处于死锁状态。

在早期的系统中，由于系统规模小、结构简单以及资源分配采用静态分配法，使得操作系统尚未暴露出死锁问题的严重性。但时至今日，软件系统变得庞大复杂，系统资源的种类日益增多，特别是系统进程、用户进程并发执行，相互通信以及资源的动态申请等，使得系统出现死锁现象的可能性和频度大大增加。死锁的出现，给系统带来了极大的危害，使系统无法正常运行。为此，关于死锁问题的研究已成为操作系统理论的重要课题之一。它是设计、分析任何一个操作系统时必须认真考虑的一个问题。

一般地，可以把死锁描述为：有并发进程 $P_1, P_2, \dots, P_n$ ，它们共享资源 $R_1, R_2, \dots, R_m$ （灶具、锅、碗、筷等）。其中，每个 P_i 拥有资源 R_i （锅、碗、筷），直到不再有剩余资源。同时，各 P_i 又在释放 R_i 的前提下要求得到 R_i （锅、碗、筷），从而造成资源的相互占有和相互等待。在没有外力驱动的情况下，该组并发进程停止往前推进，陷入永久等待状态。

产生死锁的原因

产生死锁的原因可归结为两点：

（1）竞争资源。当系统中供多个进程所共享的资源，不足以同时满足它们的需要时，引

起它们对资源的竞争而产生死锁；

(圆 进程推进顺序非法。进程在运行过程中，请求和释放资源的顺序不当，导致了进程死锁。

员援竞争资源引起的死锁

(员 可剥夺和非可剥夺性资源

系统中有些资源是可剥夺的。例如，优先权高的进程可以剥夺低优先权进程的处理机。又如，内存区可由存储器管理程序把一个进程从一个存储区移至另外一个存储区，即剥夺了该进程原来占有的存储区；甚至可将一个进程从内存调出到外存上。可见，**快**和内存属于可剥夺性资源。另一类资源是不可剥夺的，如磁带机、打印机等，当系统把这类资源分配给某个进程后，再不能强行收回，只能在进程用完后自动释放。

(圆 竞争非剥夺性资源

若系统中只有一台打印机 **印** 和一台磁带机 **磁**，可供进程 **孕** 和 **孕** 共享。假定 **孕** 已占用了打印机 **印**，**孕** 占用了磁带机 **磁**。此时，若 **孕** 继续要求打印机，**孕** 将阻塞；**孕** 若又要求磁带机，**孕** 也将阻塞。于是，**孕** 与 **孕** 之间便形成了僵局，两个进程都在等待对方释放出自己所需要的资源；但它们又都因不能获得所需的资源而不能继续推进，从而也不能释放出自己已占有的资源，以致进入死锁状态。

(猿 竞争临时性资源

上述的打印机资源属于可顺序重复使用的资源，称为永久性资源。还有一种是所谓的临时性资源，这是指由一个进程产生，被另一个进程使用一短暂时间后便无用的资源，故也称之为消耗性资源，它同样可能引起死锁。例如，在进程之间通信时由于对信号量的竞争而形成的死锁。

圆缘进程推进顺序不当引起的死锁

由于进程具有异步性，这就可能使进程按下述两种顺序向前推进。

(员 进程推进顺序合法

在进程 **孕** 和 **孕** 并发执行时，如果按照下述顺序推进：**孕** **取** **印** **取** **磁** **放** **印** **放** **磁**；**孕** **取** **印** **取** **磁** **放** **印** **放** **磁**；**孕** **取** **印** **取** **磁** **放** **印** **放** **磁**；**孕** **取** **印** **取** **磁** **放** **印** **放** **磁**；**孕** **取** **印** **取** **磁** **放** **印** **放** **磁**；**孕** **取** **印** **取** **磁** **放** **印** **放** **磁**；**孕** **取** **印** **取** **磁** **放** **印** **放** **磁**；**孕** **取** **印** **取** **磁** **放** **印** **放** **磁**；两个进程可顺利完成。图 缘原中的曲线①示出了这一情况。类似地，若按曲线②和③所示的顺序推进，两个进程也顺利完成。我们称这种不会引起进程死锁的推进顺序是合法的。

(圆 进程推进顺序非法

若并发进程 **孕** 和 **孕** 按曲线④所示的顺序推进，它们将进入不安全区 阅内。此时 **孕** 保持了资源 **印**，**孕** 保持了资源 **磁**，系统处于不安全状态。因为，这时两进程再向前推进，便可能发生死锁。例如，当 **孕** 运行到 **孕** **取** **印** **取** **磁** **放** **印** 时，将因 **印** 已被 **孕** 占用而阻塞；当 **孕** 运行到 **孕** **取** **印** **取** **磁** **放** **印** 时，也将因 **印** 已被 **孕** 占用而阻塞，于是发生了进程死锁。

缘援进程死锁举例

在这里我们给几个死锁的典型例子，以便加深对死锁的理解。

例 员援**快**操作引起的死锁。

快操作是实现进程间通信的原语，它能有效地实现进程间的同步和互斥，但由于 **快**操作使用不当，也会导致系统死锁。例如，在用 **快**操作实现进程间同

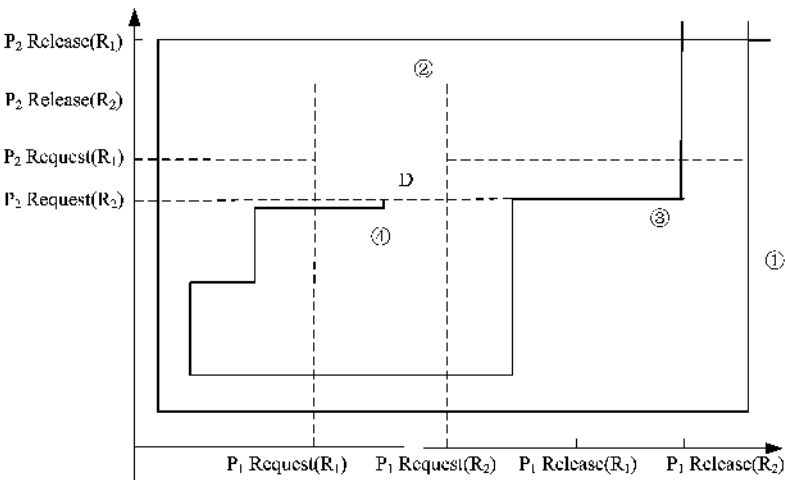


图 4-1 进程推进顺序对死锁的影响

步的例子中，设 R_1 、 R_2 是初值为 2 的两个信号量，如果进程 P1、P2 的程序安排是：

进程 P1:	进程 P2:
启动读卡机	从缓冲区取卡片信息
卡片送入缓冲区	加工卡片信息
等待卡片	等待卡片
释放卡片	释放卡片

就会产生死锁。这是因为在进程 P1 中 `wait(R1)` 与 `wait(R2)` 的顺序颠倒所引起的。类似地， R_1 、 R_2 是两个初值为 1 的信号量，则如下的两个进程就会发生死锁：

进程 P1	进程 P2
...	...
<code>wait(R1)</code>	<code>wait(R1)</code>
...	...
则： <code>wait(R2)</code>	则： <code>wait(R2)</code>

进程 P1 于 `wait(R2)` 处变成阻塞状态，而进程 P2 于 `wait(R2)` 处变成阻塞状态。

例 4-1 存储器共享的死锁。

除了 I/O 设备的共享会产生死锁外，存储器的共享在一定条件下也会产生死锁。假定系统中有 4 个单位的存储器，它为 4 个进程所共享。若每个进程都要求 2 个存储单位，当 4 个进程同时请求时就可能发生死锁。

假定 4 个进程 P1、P2、P3、P4，当每个进程首先轮流要求 1 个单位，那么第一次请求就把存储器分配完了，而第二次请求时，各进程都进入阻塞状态，结果这四个进程谁也不能向前推进，而形成死锁，如图 4-2 所示。

当 4 个进程 P1、P2、P3、P4 第一次请求就把存储器分配完了，P1 在第一次请求后就处于阻塞状态，当 P2、P3、P4 第二次请求时纷纷进入阻塞状态。从而出现了死锁，如图 4-2 所示。

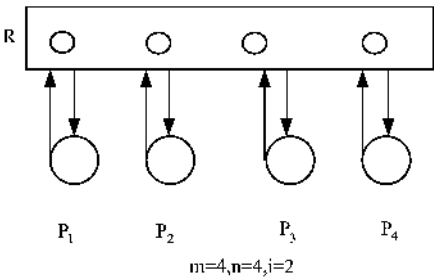


图 资源死锁示例

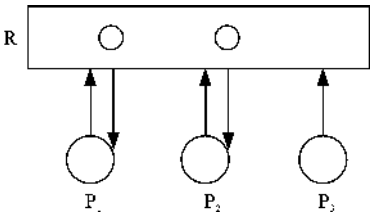
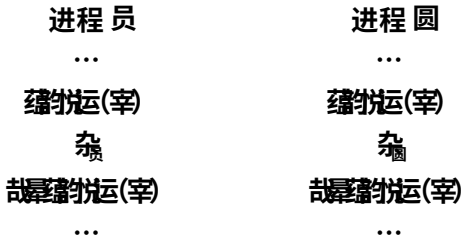


图 资源死锁示例

例 加锁法引起的死锁。

在加锁法中，我们用加锁 $\text{lock}(s)$ 和开锁 $\text{unlock}(s)$ 实现进程的互斥控制。



现在假定进程 P_1 最先进入临界区，它通过执行原语 $\text{lock}(s)$ ，将 s 从 0 改变为 1 然后执行 $\text{unlock}(s)$ ，但在 $\text{unlock}(s)$ 执行尚未完成之前，由于某种原因（如有高优先级进程 P_2 进入就绪抢占 P_1 ...）将使 P_1 从进程 P_1 转换到进程 P_2 。此时 P_1 占有临界资源，而等待处理机；另一方面进程 P_2 虽然占有了处理机但却得不到临界资源，进程 P_2 在执行 $\text{lock}(s)$ 时，由于 s 为 1 不能进入临界区。它反复测试变量 s ，等待 s 变为 0 在无外力作用下，这种状态永不改变，即 P_1 和 P_2 发生了死锁。

例 信息传送的死锁。

一般把系统中的资源分为两类：一类称为可逐次再使用资源，也称做永久性资源，记为 S_i 。它可供进程逐次使用，例如处理机、主存、I/O 设备等，都是永久性资源。另一类资源是消耗性资源，也叫做临时性资源，记以 M_i 。这类资源由一进程产生而被另一进程消耗掉，只能使用短暂的时间。例如进程间同步时交换的信息、数据文件等。对于消耗性资源同样也能引起死锁，如图 10.10 所示。

设进程 P_1 产生信息 M_1 ，而要求从 P_2 接收信息 M_1 ；进程 P_2 产生信息 M_2 ，而要求从 P_3 接收信息 M_2 ；进程 P_3 产生信息 M_3 ，而要求从 P_1 接收信息 M_3 。如果信息通信按下述进行：

- 进程 P_1 ：释放 M_1 ，请求 M_2 ；
- 进程 P_2 ：释放 M_2 ，请求 M_3 ；

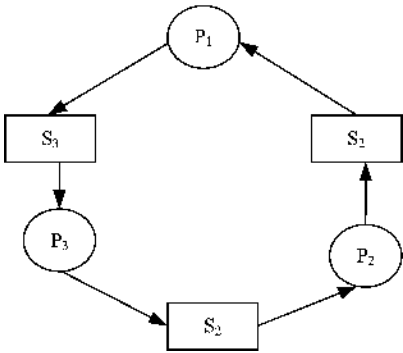


图 信息传送引起的死锁

进程 P₁: 释放 R₁, 请求 R₂;
则不会出现死锁。若按下述顺序进行:
进程 P₁: 请求 R₂, 释放 R₁;
进程 P₂: 请求 R₁, 释放 R₂;
进程 P₁: 请求 R₂, 释放 R₁;
则必然会出现死锁。

资源分配模型

为了描述和研究死锁，人们使用了各种工具。如资源分配图就是广为应用的一种工具。

资源分配图定义：资源分配图 G=(V, E)，其中顶点集 V=U∪R，U 是进程集合：{P₁, P₂, ..., P_n}，P_i 在图中用圈表示；R 是资源类集合：{R₁, R₂, ..., R_m}，R_j 表示系统中的资源，在图中用矩形表示，R_j 类资源的数量若为 n_j 则在矩形中画 n_j 个圆点。E 是边的集合。E 中有两类边，一类是请求边 (P_i, R_j)，表示进程 P_i 等待一个 R_j 类型的资源；另一类是分配边 (R_j, P_i)，表示进程 P_i 占有一个 R_j 类型的资源；

资源分配图在系统中是随时间变化的。当进程 P_i 请求 R_j 类型的资源时，将一条请求边 (P_i, R_j) 加在图中，若此请求被满足 (分给 P_i 一个 R_j 类型的资源)，则将这个请求边改成分配边 (R_j, P_i)。当 P_i 释放一个 R_j 时便去掉分配边 (R_j, P_i)。例如系统有两台宽行打印机和一台图形显示器，进程 P₁ 请求一台宽行打印机，则有图 4-1 所示的资源分配图。进程 P₁ 分配到一台宽行打印机并请求一台图形显示器，进程 P₂ 分配到一台图形显示器并请求一台宽行打印机，则有图 4-2 所示的资源分配图。

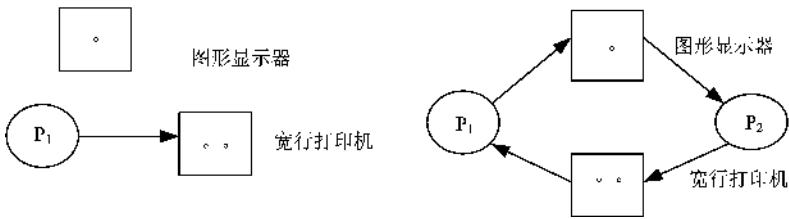


图 4-1 资源分配图的例子 图 4-2 资源分配图的例子

死锁条件

出现死锁的系统必然同时满足下列四个条件。

条件 1: 互斥。在出现死锁的系统中，必然存在需要互斥使用的资源。计算机系统中有存储器、I/O 外部设备、共享程序等各种资源。有些资源可以共享使用，有的必须独占使用，即互斥使用。若资源中所有资源均可共享使用，则进程不会处于等待资源的状态，因而不会出现死锁。因此，系统出现了死锁，说明必定有需要互斥使用的资源。

条件 2: 占有等待。在出现死锁的系统中一定有分配到了某些资源并且在等待另外资源的进程。如果这个条件不满足，则所有等待资源进程都不会占有任何资源，而资源的拥有者

也不会处在等待资源的状态中。因此，拥有资源进程迟早会释放出它们的所有资源，从而使等待这些资源的进程结束等待状态。所以，条件圆也是死锁必须满足的条件。

条件猿 非剥夺。在出现死锁的系统中一定有不可剥夺使用的资源。不可剥夺是指在进程未主动释放资源之前不可夺走资源。若资源都可剥夺，进程就不会进入僵持状态而出现死锁。

条件源 等待循环。在出现死锁的系统中，一定存在一个处于等待状态的进程集合： $\{P_0, P_1, P_2, \dots, P_n\}$ 。其中 P_0 等待的资源被 P_1 占有（ P_1 是成员 $\dots, P_n$ ），简单表示为 $P_0 \rightarrow P_1$ ， P_1 等待的资源被 P_2 占有，如图 缘原所示。

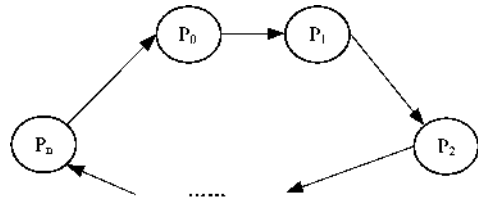


图 缘原 等待循环等待

死锁的必要条件源(循环等待)与资源分配图含有圈是等价的。系统中出现死锁则资源分配图中必有圈，但资源分配图中有圈不一定有死锁，如图 缘原所示， $\{P_1, P_2\}$ 虽存在循环等待圈，但当 P_1 释放 则类资源或 P_2 释放 则类资源后，循环等待圈即消失，如图 缘原所示。

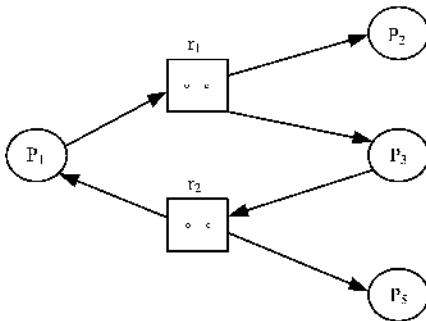


图 缘原 资源分配图含圈但无死锁的例子

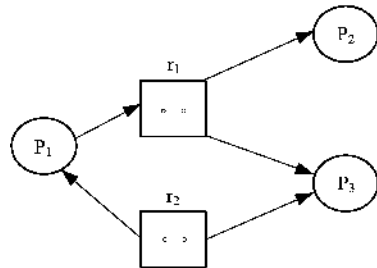


图 缘原 有圈图变为无圈图示例

资源分配图含圈而系统又不一定有死锁的原因是同类资源数大于 员 若每类资源中只有一个资源，则系统中出现死锁的充分必要条件是资源分配图含圈。

值得注意的是，上述死锁条件是必要条件而不是充分条件，这四个必要条件不是彼此独立的。比如条件源包含了前面三个条件。将它们分别列出是为了方便我们研究各种防止死锁的方法。

缘原 死锁预防

死锁问题有许多内容有待研究，本章主要介绍解决死锁所采用的方法。解决死锁一般采用两种方法，一种是设计无死锁的系统；另一种是允许系统出现死锁然后排除之。

设计无死锁的系统通常采用两种途径：一种叫死锁防止（阅读操作系统原理教材），通过破坏死锁存在的必要条件来防止死锁发生；另一种叫死锁避免（阅读操作系统原理教材），每次进行资

源分配时,通过判断系统状态决定这次分配资源后是否仍存在一条确保系统不会进入死锁状态的路径,否则即使现有资源能满足申请需要亦拒绝进行分配。

如果允许系统出现死锁,则应有能发现系统中有无进程处于死锁状态的死锁检测(阅读第 6 章第 6.4 节)手段。为此通常设立一个检测进程(平时处于睡眠状态),周期性地唤醒以检查系统中有无死锁进程。一旦发现死锁即行排除,有关排除死锁的方法称做死锁恢复(阅读第 6 章第 6.5 节)。

综上所述,死锁主要研究死锁防止、死锁避免、死锁检测及死锁恢复等问题。

破坏破坏互斥条件

如果允许系统资源都能共享使用,则系统不会进入死锁状态。但这种方法不是切实可行的,因为有些资源若共享使用则无法保证正确性。比如对临界资源的访问就必须互斥进行。

破坏破坏占有等待条件

方法 1: 进程申请到了它所需要的所有资源后才能开始运行。在运行过程中进程只能释放资源不能申请资源。由于资源是一次性全部分配,因此获得资源的进程不会进入等待资源状态。而那些所需要资源不能满足的进程则得不到任何资源,处于等待状态。这种方法虽然能保证系统不出现死锁,但是资源利用率很低。例如,某进程需要运行几十小时,在结束运行前需要打印结果,但是宽行打印机要在这个进程开始运行前就分配。

方法 2: 进程提出申请资源前必须释放已占有的一切资源。这样虽能提高资源利用率,但要仔细进行程序设计,有时仍需要提前申请资源才能保证正确性。这使用户倍感不便。

破坏占有等待条件还应考虑:若进程一次申请的资源较多,而系统又无法满足进程时只好让进程等待。如某进程 P_1 申请 R_1, R_2, R_3 但系统现在只有 R_1 故 P_1 等待。这时进程 P_2 申请 R_1 于是 P_2 被满足。之后某进程释放了 R_1, R_2 因系统已把 R_1 分配, P_1 仍需等待。若类似事件不断发生,则 P_1 将长期得不到运行而饿死,这种现象称做无限等待,其实质与死锁一样,必须有相应的防止措施,比如按申请资源的先后次序给进程赋予优先级。

资源预先分配的方法适于对外存空间的分配。因作业运行期间所需外存一般变化不大。

破坏破坏非剥夺条件

在进程主动释放占有的资源前即予以剥夺,也能保证系统不出现死锁,主要方法如下。

方法 1: 当进程 P_1 申请 R_1 类资源时,检查 R_1 中无可分配的资源:有则分配给 P_1 ;否则将 P_1 占有的资源全部释放而进入等待状态(等待 P_1 原占有的所有资源和申请的资源)。

方法 2: 当进程 P_1 申请 R_1 类型的资源时检查 R_1 中无可分配的资源:有则分配;否则检查占有 R_1 类资源的进程 P_2 ,若 P_2 处于等待资源状态,则剥夺 P_2 的 R_1 类资源分配给 P_1 。若 P_2 不处于等待资源状态,则置 P_2 处于等待资源状态(此时 P_2 原已占有的资源可能被剥夺)。

破坏破坏循环等待条件

采用资源顺序分配法可以破坏循环等待条件。该方法首先给系统中的资源编号,即寻找一个函数 $f: R \rightarrow N$ (R : 资源类集合, N : 自然数集合)。进程只能按序号由小到大的顺序申请资源。比如某进程已占有资源 $R_1, R_2, \dots, R_n$ 又申请 R_{n+1} 类资源。资源分配程序则检查是否

有：任给 $P_1, P_2, \dots, P_n$ (云(则 约云(则))。若不满足则拒绝分配。

例如：系统有输入设备 P_1 输出设备 P_2 磁带驱动器 P_3 编号为：

摇摇云(P_1 越 P_1)

摇摇云(P_2 越 P_2)

摇摇云(P_3 越 P_3)

进程 P_i 有如下请求序列：

摇摇…

(P_1 申请输入设备 P_1)

(P_2 申请输出设备 P_2)

(P_3 申请磁带设备 P_3)

语句 (P_1), (P_2) 都是正常的。语句 (P_3) 则是非法语句，因为 云(P_3 越 P_3) 云(P_3 越 P_3) 如果在语句 (P_2) 与 (P_3) 之间有“释放输出设备 P_2 ”的语句，则语句 (P_3) 合法。

采用资源顺序分配法系统不会出现循环等待。因若存在循环等待的一组进程 $\{P_1, P_2, \dots, P_n\}$ ，且 P_1 拥有资源 P_2 根据资源顺序分配原则有：云(P_1 约云(P_2 约… 约云(P_n 约云(P_1)，即云(P_1 约云(P_1)。这与资源编码的惟一性相矛盾，因此这种假设不成立。

给资源编码时应尽量使得编码能适应一般程序请求资源的先后次序。比如进程初始阶段用输入设备的可能性比较大，结束时用输出设备的可能性比较大，故通常前者编号应小于后者。

缘缘死锁避免

缘缘系统的安全状态

在预防死锁的方法中所采用的几种策略，总的来说，都是施加了较强的限制条件，从而实现简单，但却严重地损害了系统的性能。

在避免死锁的方法中，所施加的限制条件较弱，有可能获得令人满意的系统性能。在该方法中把系统的状态分为安全状态和不安全状态，只要能使系统始终处于安全状态，便可避免发生死锁。

缘缘安全状态

在避免死锁的方法中，允许进程动态地申请资源，系统在进行资源分配之前，先计算资源分配的安全性。若此次分配不会导致系统进入不安全状态，便将资源分配给进程；否则，进程等待。

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

虽然并非所有不安全状态都是死锁状态，但当系统进入不安全状态后，便有可能进入死锁状态；反之，只要系统处于安全状态，系统便可避免进入死锁状态。因此，避免死锁的实质在于：如何使系统不进入不安全状态。

缘缘安全状态之例

我们通过一个例子来说明安全性。假设系统有三个进程 P_1, P_2 和 P_3 ，共有 5 台磁带机。

进程 P_1 总共要求 4 台磁带机， P_2 和 P_3 分别要求 2 台和 1 台。设在 T_0 时刻，进程 P_1 ， P_2 和 P_3 已分别获得 2 台、2 台和 1 台，尚有 1 台空闲未分配，如表 4-1 所示。

表 4-1 进程资源分配与需求情况

进程	最大需求	已分配	可用
P_1	4	2	1
P_2	2	2	0
P_3	1	1	0

经分析发现，在 T_0 时刻系统是安全的，因为这时存在一个安全序列 P_2, P_3, P_1 ，即只要系统按此进程序列分配资源，每个进程都可以顺利完成。例如，再将剩余的磁带机取 1 台分配给 P_2 ，使之继续运行；待 P_2 完成，便释放出 2 台磁带机，使可用资源增至 2 台，以后再将这 2 台全部分配给 P_3 运行；待 P_3 完成后，将释放出 1 台磁带机， P_1 便能获得足够的资源，从而使 P_1, P_2, P_3 每个进程都能顺利完成。

安全状态向不安全状态的转换

如果不按照安全序列分配资源，则系统可能会由安全状态进入不安全状态。例如，在 T_0 时刻以后， P_1 又请求 2 台磁带机，若系统此时把剩余 1 台中的 2 台分配给 P_1 ，则系统便进入不安全状态。因为，再把其余的 1 台不管分配给 P_2, P_3 还是 P_1 ，都不能满足它们的要求，因而它们都无法推进到完成，彼此都在等待对方释放资源，结果将导致死锁。我们说，从给 P_1 分配了第 2 台磁带机开始，系统便进入了不安全状态。由此可见，在 P_1 请求资源时，尽管系统中尚有可用的磁带机，但却不能分配给它，必须让 P_1 一直等待到 P_2 和 P_3 完成，释放出资源后，再将足够的资源分配给 P_1 ，它才能顺利完成。

银行家算法

1961 年，Dijkstra 根据银行家为顾客贷款的思想提出了另一种保证系统杜绝死锁的方法（故人们称之为银行家算法）。1968 年，Dijkstra 将银行家算法推广到多类资源的环境中，形成了现在的死锁避免算法。

银行家算法中的数据结构

M ：系统中的进程个数；

N ：系统中的资源类数。

(A, B) ：现有资源向量。

$Request$ 表示 M 个未分配的 N 类资源

$(Max, Allocation)$ ：资源最大申请量矩阵。

$Request$ 表示第 M 个进程在运行过程中对第 N 类资源的最大申请量为 $Request$

$(Allocation, Request)$ ：资源分配矩阵。

$Request$ 表示进程 M 已占有 $Request$ 个 N 类资源。

$(Request, Request)$ ：进程以后还需要的资源矩阵。

$Request$ 表示进程 M 以后还需要 $Request$ 个第 N 类资源。

显然有 $Request \leq Request$ 。

(缘 砸紧紧灶 皂：进程申请资源矩阵。

砸紧紧蚤躁越表示进程蚤申请噪个第躁类资源。

资源分配程序的工作过程：当进程提出资源申请时，系统首先检查该进程对资源的申请量是否超过其最大需求量，及系统现有资源能否满足进程需要，若能则进而检查若把资源分配给该进程系统是否仍处于安全状态，若安全则予分配，否则置该进程为等待资源状态。

圆援银行家算法

若进程蚤申请资源，申请资源向量为砸紧紧灶，则有如下资源分配过程：

(员 如果砸紧紧灶跃紧紧灶，则报错返回。

(圆 如果砸紧紧灶跃紧紧灶，则进程蚤进入等待资源状态，返回。

(猿 假设进程蚤的申请已获批准，于是修改系统状态：

紧紧灶越紧紧灶+紧紧紧灶

紧紧紧灶越紧紧紧灶+紧紧紧灶

紧紧紧灶越紧紧紧灶+紧紧紧灶

(源 调用安全状态检查算法。

(缘 若系统处于安全状态，则将进程蚤申请的资源分配给进程蚤返回。

(远 若系统处于不安全状态，则不将进程蚤申请的资源分配给进程蚤恢复原来的资源分配状态，让进程蚤等待。

紧紧紧灶越紧紧紧灶-紧紧紧灶

紧紧紧灶越紧紧紧灶-紧紧紧灶

紧紧紧灶越紧紧紧灶-紧紧紧灶

猿援安全状态检查算法

设紧紧紧灶皂为临时工作向量。初始时紧紧紧灶越紧紧紧灶 令紧紧紧灶越圆…，灶。

(员 寻求躁使紧紧紧灶满足：紧紧紧灶≤紧紧紧灶 若不存在这样的躁则转至(猿)。

(圆 紧紧紧灶越紧紧紧灶+紧紧紧灶：紧紧紧灶越紧紧紧灶 转至(员)。

(猿 如果紧紧紧灶空集则返回(系统安全)。如果紧紧紧灶≠空集则返回(系统不安全)。

至此，已将死锁避免方法介绍完毕。采用死锁避免方法要求用户在提交作业时须说明对各类资源请求的最大量(即说明紧紧紧灶)。这对用户来讲是不方便的。况且在有些情况下对资源的最大申请量是不一定的。此外，系统处于不安全状态时只有发生死锁的可能性，并非一定会进入死锁状态。主要原因是进程虽然说明了在运行中所需的最大资源量，但进程活动期间对资源的请求是动态的，时而申请几个，时而释放一些。因此当系统处于不安全状态时，若碰上某些进程释放一些资源很可能系统又进入安全状态。

如果每类资源中只有一个资源，则可借助资源分配图来进行安全性检查，算法复杂度可降至韵(灶)。为了在资源分配图中能体现进程以后还需要的资源量(紧紧紧灶，在图中引入需求边约孕，则跃表示孕以后可能要申请则类型的资源。需求边用虚线表示以区别于请求边，当进程孕请求资源则时将分配边改成需求边。修改后的资源分配图与安全状态间的关系为：系统处于安全状态当且仅当图中无圈(证明留在习题)。故视资源分配图是否含圈即可判断系统是否处于安全状态，即我们只需关心图中形如孕→□→孕的顶点，并把它简化为孕→孕。于是图中顶点个数等于进程个数灶 因此算法复杂度为韵(灶)。

源 银行家算法举例

假定系统中有五个进程 {孕_园, 孕_员, 孕_圆, 孕_猿, 孕_源} 和三类资源 {粤, 月, 悦}, 各种资源的数量分别为 源, 缘, 苑. 在 栽_圆 时刻的资源分配情况如表 缘原园 所示.

表 缘原园 栽_圆 时刻的资源分配表

资源情况 进 程	孕 _园			孕 _员			孕 _圆			孕 _猿			孕 _源		
	粤	月	悦	粤	月	悦	粤	月	悦	粤	月	悦	粤	月	悦
孕 _园	苑	缘	猿	园	员	园	苑	源	猿	猿	猿	园	猿	猿	园
孕 _员															
孕 _圆	猿	园	园	园	园	园	员	园	园						
孕 _猿				猿	园	园	园	园	园						
孕 _源	怨	园	园	猿	园	园	远	园	园						
孕 _源	园	园	园	园	员	员	园	员	员						
孕 _源	源	猿	猿	园	园	园	源	猿	员						

(员) 栽_圆 时刻的安全性: 利用安全性算法对 栽_圆 时刻的资源分配情况进行分析, 如表 缘原员 所示可知, 在 栽_圆 时刻存在着一个安全序列 {孕_员, 孕_猿, 孕_源, 孕_圆, 孕_园}, 故系统是安全的.

表 缘原员 栽_圆 时刻的安全序列

资源情况 进 程	孕 _园			孕 _员			孕 _圆			孕 _猿			孕 _源		
	粤	月	悦	粤	月	悦	粤	月	悦	粤	月	悦	粤	月	悦
孕 _园	猿	猿	园	员	园	园	园	园	园	缘	猿	园	猿	猿	猿
孕 _员	缘	猿	园	园	员	员	园	员	员	苑	源	猿	猿	猿	猿
孕 _圆	苑	源	猿	源	猿	员	园	园	园	苑	源	缘	猿	猿	猿
孕 _猿	苑	源	缘	远	园	园	猿	园	园	园	源	苑	猿	猿	猿
孕 _源	园	源	苑	苑	源	猿	园	员	园	园	缘	苑	猿	猿	猿

(圆) 孕_圆 请求资源: 孕_圆 发出请求向量 砸_圆 = (员, 园, 园), 系统按银行家算法进行检查:

- ① 砸_圆 ≤ 孕_圆 的可用资源; 即 (员, 园, 园) ≤ (员, 园, 园);
- ② 砸_圆 ≤ 孕_圆 的当前分配; 即 (员, 园, 园) ≤ (猿, 园, 园);
- ③ 系统先假定可为 孕_圆 分配资源, 并修改 孕_圆 的当前分配和 孕_圆 的可用资源向量, 由此形成的资源变化情况如表 缘原圆 中的圆括号所示.
- ④ 再利用安全性算法检查此时的系统是否安全, 如表 缘原圆 所示.

表 缘原缘孕 申请资源时的安全性检查

资源情况 进播程	宰燥燥			晕燥燥			粤燥燥燥燥燥			宰燥燥燥燥燥燥燥燥			云燥燥燥
	粤	月	悦	粤	月	悦	粤	月	悦	粤	月	悦	
赓	圆	猿	园	园	圆	园	猿	园	圆	缘	猿	圆	燥燥燥
赓	缘	猿	圆	园	员	员	圆	员	员	苑	源	猿	燥燥燥
赓	苑	源	猿	源	猿	员	园	园	圆	苑	源	缘	燥燥燥
赓	苑	源	缘	苑	源	猿	园	员	园	苑	缘	缘	燥燥燥
赓	苑	缘	缘	远	园	园	猿	园	圆	园	缘	苑	燥燥燥

由所进行的安全性检查得知，可以找到一个安全序列 {孕, 孕, 孕, 孕, 孕}。因此，系统是安全的，可以立即将 孕 所申请的资源分配给它。

(燥 孕 请求资源：孕 发出请求向量 燥燥燥燥燥(猿 猿 园)，系统按银行家算法进行检查：

- ① 燥燥燥燥燥(猿 猿 园) ≤ 晕燥燥燥(源 猿 员)；
- ② 燥燥燥燥燥(猿 猿 园) 跃 粤燥燥燥燥燥(圆 猿 园)，让 孕 等待。

(源 孕 请求资源：孕 发出请求向量 燥燥燥燥燥(园 圆 园)，系统按银行家算法进行检查：

- ① 燥燥燥燥燥(园 圆 园) ≤ 晕燥燥燥(苑 源 猿)；
- ② 燥燥燥燥燥(园 圆 园) ≤ 粤燥燥燥燥燥(圆 猿 园)；
- ③ 系统暂时先假定可为 孕 分配资源，并修改有关数据，如表 缘原缘所示。

表 缘原缘 为 孕 分配资源后的有关资源数据

资源情况 进播程	粤燥燥燥燥燥			晕燥燥燥			粤燥燥燥燥燥		
	粤	月	悦	粤	月	悦	粤	月	悦
赓	园	猿	园	苑	圆	猿	圆	员	园
赓	猿	园	圆	园	圆	园			
赓	猿	园	圆	远	园	园			
赓	圆	员	员	园	员	员			
赓	园	园	圆	源	猿	员			

④ 进行安全性检查：可用资源 粤燥燥燥燥燥(圆 员 园) 已经不能满足任何进程的需要，故系统进入不安全状态，此时系统不分配资源。

如果在银行家算法中，把 孕 发出的请求向量改为 燥燥燥燥燥(园 员 园)，系统是否能将资源分配给它，请读者考虑。

缘原缘 死锁检测

若系统中制定死锁避免的措施，就应该设置一套机构用来检查系统中有无进程已进入锁状态。但一旦发现死锁则应立即排除，以确保系统继续正常运行。人们通常采用化简资源

分配图的方法来检测系统中有无进程处于死锁状态。资源分配图的简化过程如下：

(1) 在图中找出一个非孤立进程顶点 P_i ， P_i 的请求边均能立即满足。

(2) 若找到了这样的 P_i ，则将与 P_i 相连的边全部删去，转为 (1)；否则化简过程结束。

如果化简后所有的顶点都成了孤立点，如图 4-10(a) 所示，则称该图可完全化简；否则称该图是不可完全化简的，如图 4-10(b) 所示。不难证明系统中有死锁的充分必要条件是资源分配图不可完全化简。经过化简后非孤立点的进程处于死锁状态（证明留做练习）。

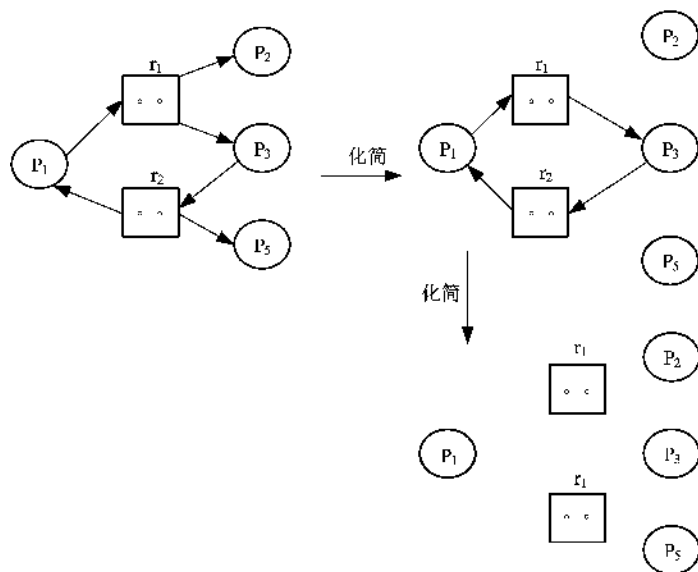


图 4-10(a) 可完全化简图

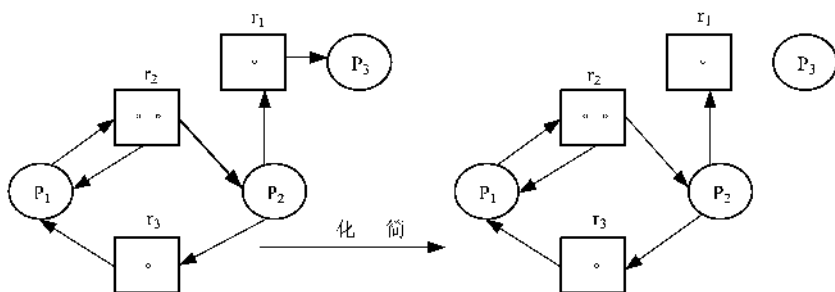


图 4-10(b) 不可完全化简图

上节所介绍的安全状态检查算法也可用来检测死锁，其检测原理与资源分配图的简化法一样。

死锁恢复

当死锁检测程序发现系统中有死锁进程时，可通过破坏死锁的必要条件立即排除之。

破坏互斥条件：在死锁进程形成的等待环中将某资源同时分配给多个进程，如在图

图 6-10 所示的死锁进程中, 若把通常要求互斥使用的打印机从 P_3 那里剥夺过来分给 P_2 , 死锁便排除了。剥夺时要保存中间结果。为了保证正确性, 有些资源不可剥夺。

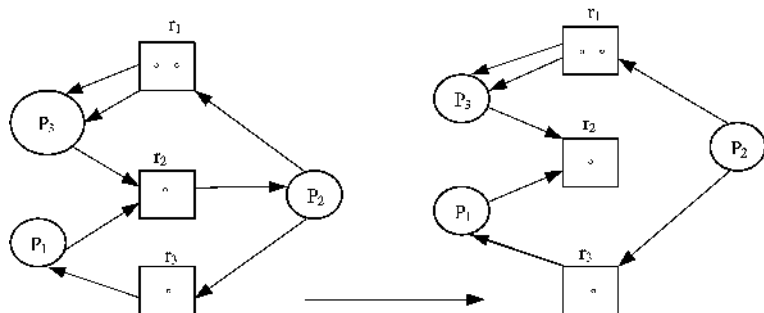


图 6-10 删除进程 P_3 示意图

破坏循环等待条件：从死锁进程集合中选择某个进程予以删除 (如删除图 6-10 中的 P_3) 死锁便排除了, 如图 6-11 所示。选择删除的进程时一般从优先级、进程运行的时间长度以及进程已用了多少资源等方面去考虑, 使系统损失最小。在多个进程中应该首先考虑优先级低的进程作为删除对象。但是在优先级相等或相差不大的情况下则考虑进程运行了多长时间, 还需要运行多长时间这一因素, 以减少损失。计算机中有些资源很宝贵, 使用一次费用较大, 选择被删除的进程时应尽量避免选择这类进程。

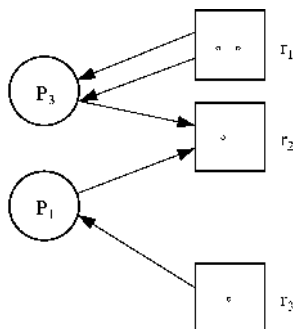


图 6-11 解除死锁后的资源分配图

选出被删除的进程后, 一种方法是把它全部删除, 另一种方法是让进程退回到系统可摆脱死锁状态的某一点上。让进程退到某点叫做重算 (rollback)。重算需保留运行过程中的一些状态, 故较复杂。

本章小结

计算机系统中进程间由于竞争资源而造成相互等待以致僵持不前的状态称做死锁。出现死锁时系统必然满足四个条件：互斥、占有等待、非剥夺和循环等待。通常采用两种方法解决死锁：一是设计无死锁的系统；二是允许系统出现死锁而后排除。前者包括死锁的防止和避免。死锁防止是通过破坏死锁存在的四个必要条件进行的, 死锁避免是通过保证系统随时处在安全状态而实现的。后者包括死锁检测和死锁恢复。死锁检测可以通过化简资源分配图来实现, 死锁恢复可以通过破坏死锁的四个必要条件来达到。死锁的各种处理方法都有一定的局限性, 采用综合方法处理死锁在一定程度上能解决由于这些局限而造成的问题。

在设计操作系统时除应小心对待死锁外, 还应小心对待无限等待问题。原则上无限等待虽可自行解决, 但实质上其后果与死锁一样。解决无限等待问题通常采用的方法是把进程等

待的时间长度作为提高进程优先权的一个因素。

现在的操作系统中死锁问题并不十分严重。多数系统采用资源顺序分配来解决死锁，或者对死锁不理睬，当操作员从外部发现系统僵持不前时重新启动一次系统。这是因为死锁在当今系统中并不经常发生。但计算机正向着大规模的并发方向发展，预计死锁问题会严重起来。众所周知，死锁是随着多道程序设计的诞生而出现的，并发性越高，出现死锁的可能性也就越大。此外，目前设计的多道程序是非智能的，但随着计算机要向着智能化方向发展，智能程序更具有动态性，即系统中进程对资源的申请更具有动态性，运行中需要何种资源更难预测，且数据成为资源的趋势日益显著，资源数目将大为增加。这一切均预示着现有对付死锁的方法无法满足将来系统的需要，有待进一步发展。

习 题

1. 死锁的四个必要条件彼此独立吗？试给出最少的必要条件。

2. 若系统中只有一个进程，可能发生死锁吗？

3. 设系统中只有一种资源，进程一次只能申请一个资源，进程申请的资源总数不会超过系统资源总数。问下列情况哪些会发生死锁？

	进程数	资源总数
(1) 5 个进程，每个进程需要 2 个资源	5	10
(2) 5 个进程，每个进程需要 3 个资源	5	15
(3) 5 个进程，每个进程需要 4 个资源	5	20
(4) 5 个进程，每个进程需要 5 个资源	5	25

4. 现在假设进程最多需两个资源，问下列情况哪些会发生死锁？

	进程数	资源总数
(1) 5 个进程，每个进程需要 1 个资源	5	10
(2) 5 个进程，每个进程需要 2 个资源	5	10
(3) 5 个进程，每个进程需要 3 个资源	5	10
(4) 5 个进程，每个进程需要 4 个资源	5	10
(5) 5 个进程，每个进程需要 5 个资源	5	10

5. 考虑由 5 个相同类型资源组成的系统，系统中有 5 个进程，每个进程最多需要 2 个资源。该系统是否会发生死锁？为什么？

6. 系统由相同类型的 5 个资源组成，系统有 5 个进程，每个进程至少请求一个资源。证明：当 5 个进程最多需要资源数之和小于 5 时，该系统无死锁。

7. 对哲学家就餐问题，若哲学家的生活过程如图 5-1 所示，问这五个哲学家在什么情况下进入死锁状态？重新设计一种拿筷子的方法，使死锁不会发生。新的设计的方法会不会使得有的哲学家被饿死？

8. 在某系统没有死锁预防和死锁避免机构。该系统每月运行 100 个作业，大约每个月发生两次死锁。当死锁出现时要求中止并重新启动大约 50 个作业。每个作业耗费 100 元钱，作业被中止时平均有一半的工作已被完成。性能管理者已估算出，若装配一个死锁避免机构，将使每个作业的执行时间增加 10%，平均周转时间增加 10%。由于系统当前有 100 个

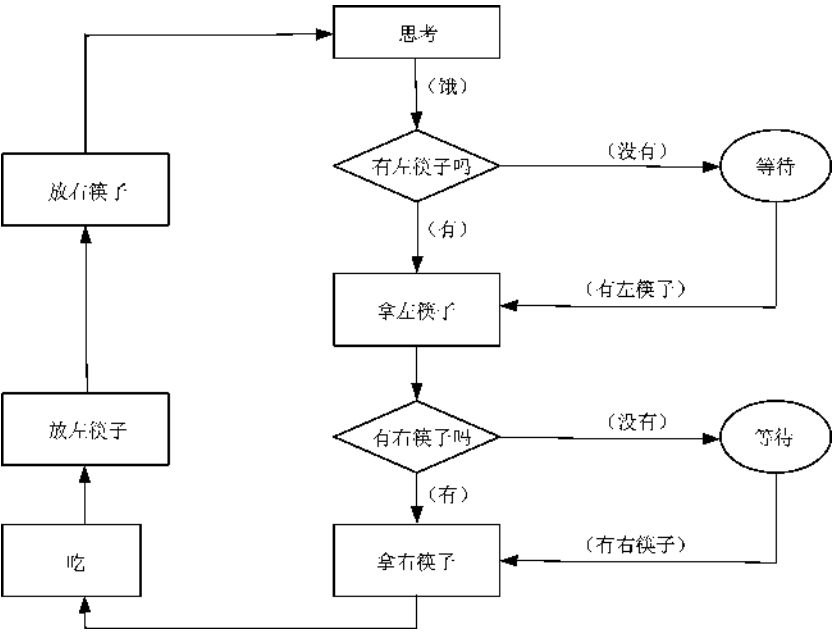


图 缘原哲学家生活过程

空闲时间，所以每月仍能运行完缘个作业。问：赞成装配死锁避免机构的理由是什么？反对装配死锁避免机构的理由又是什么？

在实际的计算机系统中，资源数和进程数是动态变化的。当系统处于安全状态时，如下变化是否可能使系统进入非安全状态？

- (葬) 增加粤进程数

(糟) 增加配管

(藻) 增加进程数
- (遭) 减少粤进程数

(凿) 减少配管

(枣) 减少进程数

假设系统状态如下：

进程号	粤进程数	配管	粤进程数
员	园园员园	员缘园园	园园员园
圆	员园园园	员苑缘园	
猿	员猿源缘	园猿缘园	
源	园元猿园	园元缘园	
缘	园园员缘	园元猿元	

使用银行家算法回答下列问题：

- (葬) 晕的内容是什么？
- (遭) 系统是否处于安全状态？
- (糟) 如果进程圆请求(园缘园园)，能否立即得到满足？

在解决死锁问题的几种方法中，哪种方法最容易实现？哪种方法使资源利用率最高？

例 5.1 在银行家算法的例子中，如果 P_0 发出的请求向量由 $\langle 1, 1, 0, 0 \rangle$ 改为 $\langle 1, 1, 1, 0 \rangle$ ，问系统可否将资源分配给它？

解 银行家算法有某些不足之处，使得该算法难于在计算机系统中应用，说明之。

例 5.2 在银行家算法中，若出现下述资源分配情况（假设有 4 类资源）：

进程	已分配资源	申请资源	最大需求
P_0	$\langle 1, 1, 0, 0 \rangle$	$\langle 1, 1, 1, 0 \rangle$	$\langle 3, 3, 1, 0 \rangle$
P_1	$\langle 1, 1, 1, 0 \rangle$	$\langle 0, 0, 0, 0 \rangle$	$\langle 2, 2, 2, 0 \rangle$
P_2	$\langle 1, 1, 1, 0 \rangle$	$\langle 0, 0, 0, 0 \rangle$	$\langle 2, 2, 2, 0 \rangle$
P_3	$\langle 0, 0, 0, 0 \rangle$	$\langle 2, 2, 1, 0 \rangle$	$\langle 2, 2, 2, 0 \rangle$
P_4	$\langle 0, 0, 0, 0 \rangle$	$\langle 2, 2, 1, 0 \rangle$	$\langle 2, 2, 2, 0 \rangle$

试问：

(1) 该状态是否安全？

(2) 若进程 P_0 提出请求 $\langle 1, 1, 1, 0 \rangle$ 后，系统可否将资源分配给它？

解 讨论死锁检测算法与死锁避免算法的联系与区别。

例 5.3 化简如图 5.1 所示的资源分配图。并说明有无进程处于死锁状态？

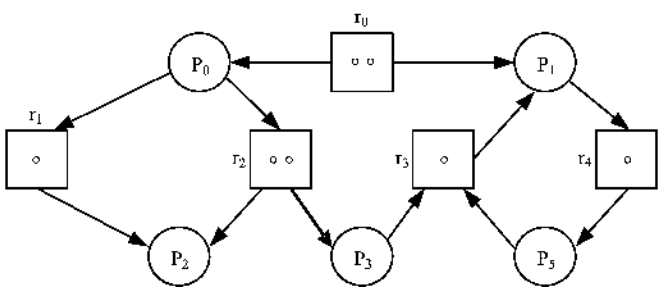


图 5.1 资源分配图

例 5.4 没有死锁预防、避免、检测和恢复措施的某系统是否一定会发生死锁？若发生了死锁会出现什么现象？系统管理员对此应采取什么措施？

解 设系统中每类资源的资源数为 M ，写出时间复杂度为 $O(M^2)$ 的死锁检测算法（ M 是进程个数）。

例 5.5 早期操作系统中，进程空间（输入输出流）总是固定的。现在的操作系统进程空间大小可以根据需要发生变化。现在的操作系统减少还是增大了出现死锁的可能性？

第 1 章 存储管理

存储器是计算机系统的重要组成部分，程序及其所访问的数据必须在内存中才能得到执行。为提高存储器的利用率及其响应速度，操作系统采用多道程序并发执行，因此存储器的有效管理直接影响系统的性能。本章重点讨论基本存储管理技术，主要介绍操作系统中有关存储管理的基本概念，几种常用存储管理技术的基本思想、实现算法及硬件支持。虚存管理技术和辅助存储器管理将在后续章节中进行阐述。

1.1 存储管理功能

存储管理的主要任务是：(1) 为多道程序的运行提供良好的环境和存储基础；(2) 方便用户使用存储器，提高存储器和存储器的利用率；(3) 从逻辑上来扩充主存储器。所以，存储器管理的主要功能包括主存空间分配和管理、地址转换和重定位、存储保护和共享以及存储扩充等四个方面。

1.2 主存空间分配和管理

内存空间是由存储单元（字节或字）组成的一维连续的地址空间，用来存放当前正在运行的代码及数据。主存分配和管理的功能是提供恰当的算法为每个进入内存的程序分配内存空间，并采用合适的数据结构对已分配的或空闲的内存空间进行管理。

主存分配和管理的实现分为连续分配和离散分配。连续分配是指为用户程序分配连续的内存空间，它包括：

(1) 单一连续分配。这是最早出现的存储管理方式，基本思想是在整个内存用户区由单一用户的一道程序独占。这种方式只适用于单用户单任务操作系统（如 DOS），而不适于多道程序运行环境。

(2) 分区分配。其主要思想是将内存用户区分成若干分区，每个分区存储一道程序。它又可以分为固定分区式（静态分配）和动态分区式（动态分配）。固定分区式中分区大小固定，作业运行期间不允许“移动”和申请新的内存空间；动态分区式的分区大小和数目可变，作业运行期间允许申请新的内存空间，由于会产生多个不可利用的内存零头（称“碎片”），所以需要采用紧凑技术，因此作业的物理地址空间在运行区间可能“移动”。

离散分配是指用户程序分散存储在许多不相邻的内存分区中，从而避免由紧凑“碎片”带来的额外的系统开销。根据离散分配时所用基本方式的不同，可以将其分为分页存储管理、分段存储管理和段页式存储管理三种方式。

后续的各节详细讨论了这三种基本的存储管理技术。

4.1.1 地址转换和重定位

地址转换和重定位的目的是在多道程序环境下将程序中的逻辑地址空间映射成内存空间中与之对应的物理地址空间。该功能需要在地址映射硬件的支持下完成。

一个程序的运行通常必须经过“编译—链接—装入”三阶段，如图 4-1 所示。编译过程形成若干个目标模块；链接程序将其和相关的库函数链接，形成装入模块；装入程序将装入模块装入内存，创建进程，形成内存中可运行的程序。

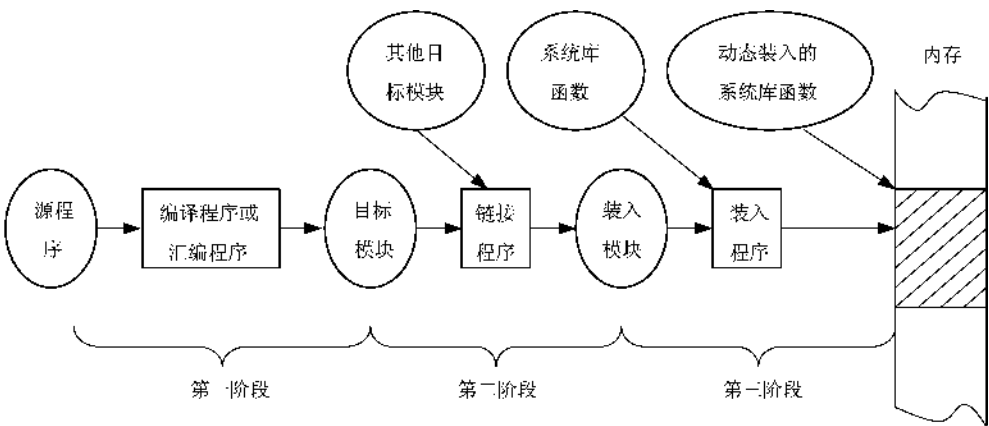


图 4-1 用户程序的处理过程

4.1.2 逻辑地址空间和物理地址空间

逻辑地址空间是指相对地址的集合。编译程序为目标程序指令顺序分配地址，这些地址称为相对地址或逻辑地址。逻辑地址空间是相对于用户或程序设计人员而言的，用户设计好的程序都存于自己的地址空间，只有当它运行时，系统才将它装入到内存中。

物理地址空间是指内存地址的集合，又称为内存空间。内存空间是一维线性空间，内存中的存储单元顺序编号，这些编号称为物理地址或绝对地址。物理地址空间的大小是由内存的实际容量决定的。物理地址空间是系统管理和维护的对象。

4.1.3 程序的链接

链接程序的功能是将经过编译或汇编后所得到的一组目标模块及其所需库函数，装配成完整的装入模块。实现链接的方法有三种：静态链接、装入时动态链接和运行时动态链接。

(1) 静态链接。即将编译后的目标模块及其调用的其他目标模块事先链接成完整装入模块的过程。这种方式形成的装入模块又叫可执行文件，运行时直接装入内存。所以在将几个目标模块链接装配成可执行文件时，必须对相对地址和符号地址进行修改，如图 4-2 所示。

(2) 装入时动态链接。用户源程序经编译后所得到的目标模块，在装入主存时，边装入边链接。这种方式便于目标模块的修改和更新，同时也利于目标模块的共享。

(3) 运行时动态链接。目标模块的链接推迟到执行时才进行，即在执行过程中，若发现某个调用模块尚未装入内存，则由操作系统找到该模块，将它装入内存，并把它链接到调用者模块上。该方式可解决装入时动态链接的低效装入和内存浪费问题。

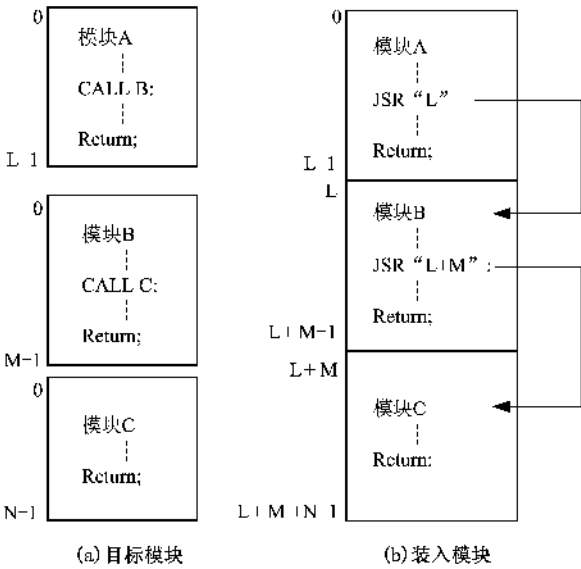


图 远源程序链接示意图

程序的装入

将装入模块装入内存，可采用三种方式：

(1) 绝对装入方式。编译程序产生绝对地址的目标代码，装入程序根据装入模块中的绝对地址，将程序和数据装入内存；装入内存后，目标程序中指令和数据的地址不能进行修改。优点是目标程序再次装入时，操作简单方便；但程序员必须熟悉内存使用情况，且装入的程序不能“移动”，不能动态增长。

(2) 可重定位方式。根据内存当前使用情况，在装入过程中将装入模块定位到内存中的位置。这种方式下，地址变换在装入时一次完成，程序运行过程中不能进行修改，故又称为静态重定位方式。

(3) 动态运行时装入方式。动态运行时的装入程序，在装入时并不立即将相对地址转换为绝对地址，而是推迟到程序真正执行时才进行，所以又称为动态重定位装入方式。这种方式的优点是可使大程序执行，由于不常用的程序和模块只在需要时加载，可减少内存空间的浪费。但为了让地址转换不影响指令的执行速度，需要有特定硬件的支持，如重定位机构等。

重定位技术

内存空间的地址从 0 开始按序编号，多道程序环境下，由于编译程序不能预知目标模块在内存的位置，所以装入时需要进行地址转换。在作业装入内存时，作业所在的物理内存空间一般与作业的逻辑地址空间不一致，为了保证加载执行指令时可正确访问存储单元，必须对程序指令和数据的地址进行修改，即将用户程序中的逻辑地址转换为运行时可由机器直接寻址的物理地址，这一过程称为地址映射，或称为重定位。

根据重定位时机的不同，重定位技术可分为两种：

(1) 静态重定位。即目标程序装入内存时，由装入程序对地址进行修改，把程序的逻辑地址空间转换成实际内存地址空间的过程。

(2) 动态重定位。即在程序执行访问过程中进行地址的映射。逻辑地址空间在装入过程

中不发生变化。

静态重定位方式的优点是不需增加硬件地址转换机构,便于实现程序的静态连接。其缺点显而易见,由于程序占用内存中的连续空间,在重定位后不能再移动,不利于内存空间的有效使用;且各个用户进程很难共享内存中同一程序的副本。

动态重定位的优点是程序占用的内存空间动态可变,而不需连续,并可实现同一程序副本的共享使用;缺点是需要附加的硬件支持,实现存储管理的软件也相对比较复杂。

存储保护和共享

存储保护

存储保护的主要目的是为多道程序共享内存提供保障,确保每道程序都在自己的内存空间中运行,保护不越界访问其他用户程序或操作系统的程序和数据。为了保护存储空间不受干扰,必须设置保护机制,通常由硬件完成,软件辅助实现。最简单的内存保护机制是采用界限寄存器存放正在执行程序的上下界,系统的越界检查程序对每条指令的地址进行检查,该过程由硬件实现,结合软件共同完成发生越界后的处理。

除界限寄存器保护之外,还有保护键法。这种方法通过设置在存储块中的保护键和程序状态字中的保护键开关字段来保护存储块。保护键可设置成对读写同时保护的或只对读、写进行单项保护的。对不同的进程赋予不同的开关代码,当开关代码和受保护的存储块中的保护键匹配,则访问允许,否则将产生访问出错中断。

对于允许多个进程共享的存储区域,存储保护要提供读写保护。即每个进程对共享内存区有自己的访问权限,如果某一个进程对共享区域的访问违反了权限规定,则操作系统启动存储保护机制禁止越权操作。

共享

共享主要是为内存中多个进程协作完成任务所设计的,存储共享指的是两个或多个进程共用内存中相同区域,它有两方面的含义:

① 是指多个用户程序共同使用存储空间,各个程序使用各自不同的存储区域;

② 是指多个用户程序共同使用主存中的某些程序和数据区,这些共享程序和数据区称为共享区。

存储共享的目的在于节省内存空间,提高内存利用率,并实现进程通信和数据共享。在多道程序环境下,多个程序可能执行同一代码段,多个合作进程需要访问相同的数据结构,所以,在存储管理中必须提供进程对某些主存区共享的灵活性。

存储扩充

由于物理内存的容量是有限的,当有用户程序所要求的内存空间超出了内存总容量时,若采用实存管理,作业将无法得到执行。或当大量用户程序要求同时运行,则可能造成大量程序必须等待。若从物理上增加内存,将增加系统成本,同时还要受到机器本身的限制。所以存储扩充的主要目的是借助于虚拟存储技术,使运行程序不受主存大小的限制,从逻辑上扩充内存容量,使用户感觉到可运行任意大小的程序,或使更多的用户程序能并发运行。第 5 章将重点讨论虚拟存储技术。

选择分区存储管理

选择固定分区管理

这是最早的支持多道程序设计的内存管理方式。其基本思想是将内存划分为若干个固定大小的区域，每个区域的边界固定。每个分区装入一道作业，允许多道作业并发运行；当有空闲分区时，则从外存后备队列中选择一个适当大小的作业装入该分区运行。

分区划分

按照分区尺寸是否相等，有两种分区划分方法：

(1) 分区大小相等。即所有的内存分区大小相等，作业以分区为单位进行分配。它存在两个明显的缺点，其一，当程序太小时，分配分区将造成内存空间浪费，产生内碎片（内部碎片）；其二，当程序太大时，可能一个分区不足以装入该程序，则使该程序无法运行，或者需要程序员使用覆盖技术设计程序，增加了设计的难度。

(2) 分区大小不等。即内存空间划分为各种大小不等的区域，适应于大中小不同的作业需求，小分区分配给小程序，对大程序则分配大分区，从而克服分区大小相等的缺点。

分区数据结构

为便于内存分配，通常系统将分区根据大小排序，并建立一张分区说明表，每个分区对应表中的一行。表中包含的内容有：

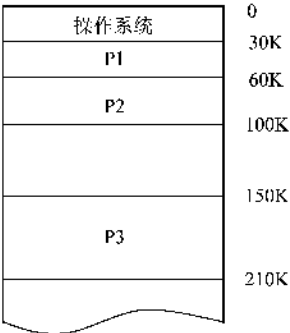
- (1) 大小：说明该分区的内存大小，一般以 K 为单位。
- (2) 起始地址：说明该分区在内存中的起始地址。
- (3) 状态：描述该分区的使用情况，若有作业正在使用，则为在使用，否则为未使用。

如图 10-10 所示，当前内存中存在多个分区，前四个分区的大小分别是 30K、60K、100K 和 150K，当前第一个分区、第三个分区和第四个分区已经分配，而第二个分区未分配；图 10-11 所示为根据分区说明表所显示的内存空间分配情况，其中 0~30K 的内存空间是操作系统常驻内核程序。

分区说明表大小是固定不变的，管理简单，可常驻内存。

分区号	大小 (K)	起始地址 (K)	状态
1	30	0	已使用
2	60	30	已使用
3	100	90	未使用
4	150	190	已使用
5	210	340	未使用

(10-10) 分区说明表



(10-11) 内存空间分配情况

图 10-10 选择固定分区数据结构示意图

内存分配和回收

固定分区的分配和回收过程最简单。分配的基本操作如下：

(1) 需要运行的作业向系统提出分配内存的请求；

(2) 系统根据用户请求，启动内存分配程序，查找分区说明表，从中选择一个满足要求的、且是未使用的分区，若找到则将它分配给该作业，修改分区说明表中该分区表项的状态为“已使用”，分配结束；

(3) 若找不到大小足够的分区，则拒绝为该作业分配内存，返回信息，结束操作。

当用户程序不再使用分给它的分区，则释放相应的内存空间。系统进行回收操作，根据分区起始地址或分区号，在分区说明表中找到相应的表项，将其状态置为“未使用”。

固定分区分配方法简单，但分区大小固定，必然造成存储空间的浪费。如在图 4-10 所示情况下，若作业 A 提出申请 100 字节的内存空间，系统将会把分区 1 分配给它，该分区中其余 100 字节将不能被其他任何程序使用而造成浪费。这种不能被其他任何作业或程序所使用的分区内的内存空间称为内存碎片。固定分区方法现已很少使用，但在某些控制系统中，由于控制程序固定，数据空间也一定，故仍可采用固定分区管理方法。

动态分区管理

动态分区分配是为解决内存浪费（内存碎片）问题而提出的。所谓动态分区是指分区的大小、数量和位置随着内存中进程的大小和数量动态变化。由于用户作业的大小不可能预先规定，且作业到来的分布情况无法预知，所以这种方法的基本思想是根据作业的实际需要，在装入内存时动态地分配连续的内存空间。

链式数据结构

为实现动态分区分配，系统一般可采用以下两种数据结构来记录内存的使用情况。

(1) 空闲分区表

用于管理内存中尚未使用的分区，如图 4-11 所示。内存的每个空闲分区占用该表的一项，每个分区表项包括有分区序号、分区始址及分区大小等信息。

(2) 空闲分区链

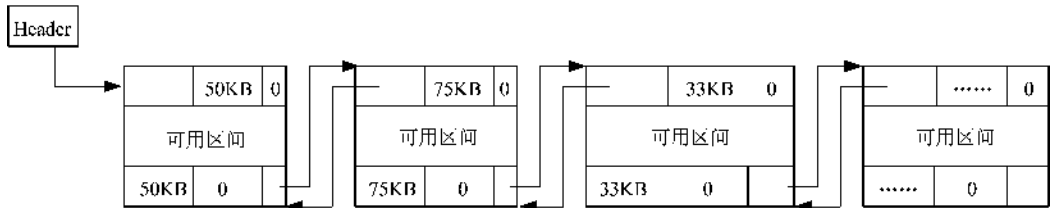
空闲分区链是指采用链指针将空闲分区链接，实现空闲分区管理的方法。在链式数据结构中，每个分区必须设置：

- ① 前向指针，位于分区的开头，用于链接前一个空闲分区；
- ② 后向指针，位于分区的尾部，用于链接后一个空闲分区；
- ③ 状态位，位于分区的开头，用于描述分区分配信息，状态位为“0”表示未分配，否则表示已分配；
- ④ 分区大小，位于分区的开头。

分区的中间部分为存放作业的空闲内存空间，通过前向指针和后向指针将空闲分区链设置成双向链结构。为了检索空闲分区的方便，在分区尾部可重复设置状态位和分区大小表目，如图 4-12 所示，其中可用区间是实际能存储数据的空间，它的大小等于分区空间减去指针和分区信息的大小，如第一个空闲块，总的大小为 100 字节，而实际可用区间小于 100 字节，若指针、状态位等信息占用 10 个字节，则可用区间只有 (100 - 10) 字节。

分区序号	分区大小 (字节)	分区始址 (字节)	状态
1	100	100	未使用
2	200	200	未使用
3	300	300	未使用
...	...	...	...

(空闲分区表示意图)



(空闲分区链结构示意图)

图 1 空闲分区链结构示意图

空闲分配算法

所谓分配算法是指某一时刻需装入作业时，从空闲分区表或空闲分区链中选择足够大的内存空间进行分配。常用的分配算法有四种。

(1) 首次适应算法。从空闲分区表或空闲分区链的开始处进行顺序扫描，选择第一个满足要求的空闲分区，从该分区中划出一块作业大小的空间分配给作业，并修改相应的数据结构。这种算法的优点是能保留高地址部分的大空闲区，为大作业的分配创造了条件，该算法被认为是最好且是最快的算法；缺点是低地址部分被不断划分，形成难以利用的外碎片（碎片），而且每次查找都是从低地址部分开始，这将增加开销。

(2) 循环首次适应算法。在为作业分配内存空间时，不再从空闲分区表或空闲分区链的起点开始查找，而是从上次找到的空闲分区的位置开始查找，直至找到第一个能满足要求的空闲分区，并从中划出一块与作业大小相等的内存空间进行分配。优点是使空闲分区分布均匀，减少查找开销；缺点是对大作业比较不利，可能会缺乏大的空闲空间，且需要设置查询指针。

(3) 最佳适应算法。在为作业分配内存时，选择既满足要求且是最小的空闲分区进行分配。算法的特点是能使每次分配后内存产生的剩余分区尺寸最小，从空间利用率上看是最佳的；缺点是容易形成外碎片。

(4) 最坏适应算法。作业分配的内存空间是整个空闲分区表或链中最大的。这种策略使分配后剩余的空闲分区仍然足够大。缺点是查找需要额外的开销。

由于动态分区的位置、大小和数量都在动态变化，所以，内存的动态分配需要考虑两个问题，即分配的效率和减少外碎片。外碎片是指程序之间不能再利用的内存空间。一个好的动态分配算法应该能快速响应满足各种内存大小的分配要求，而且不产生大量的内存碎片，同时在动态管理过程中能跟踪内存使用情况。在实际的管理中，为了不产生过多的外碎片，一般系统会规定一个不再划分的剩余分区大小，当产生的剩余空间小于该值时，空间不

再划分,而是整个分配给该作业。

动态分区分配与回收

在动态分区管理中,内存分配主要是根据分配算法,选择合适的空闲分区进行分配。当一个作业请求分配内存时,系统按如下步骤进行分配:

(1) 系统根据分配算法选择一块空闲分区,使其大小 $\geq$ 作业的大小。

(2) 判断 (1) 中选择的空闲分区是否小于事先规定的最小剩余空间。若是,将整个分区分配给作业。

(3) 若 (1) 中选择的空闲分区大于最小剩余空间,则从该分区中划出 $\geq$ 作业大小的分区进行分配;其余部分仍留在空闲分区表或空闲分区链中。

(4) 修改有关数据结构,将分区首地址参数返回给请求者,结束分配。

当进程运行完毕释放内存时,系统必须根据回收区的首地址,进行内存区的回收处理。由于是动态分区管理,所以回收过程中必须考虑是否需要与回收区地址前后空闲块的合并。分为四种情况,回收的具体步骤包括:

(1) 找到插入点。系统根据回收分区的首地址,顺序检索空闲分区表或空闲分区链,直到找到某个空闲分区地址 $\geq$ 回收区地址末尾地址;根据回收区的位置分为四种情况进行判断。

(2) 如果回收区所在位置的上下两相邻分区都是空闲区,则将回收区和上下两相邻分区合并为一个空闲区。新空闲区的起始地址为上相邻空闲区的起始地址,大小为三者之和。空闲区合并后,删除空闲分区表下相邻空闲区的表项或空闲分区链中下相邻空闲区的链指针,修改上相邻空闲区的分区大小。

(3) 如果回收区的上相邻区是空闲区,则将回收区与上相邻空闲区合并,修改空闲分区表或空闲分区链中上相邻空闲区的分区大小为两者之和。

(4) 如果回收区的下相邻区是空闲区,则将回收区与下相邻空闲区合并,修改空闲分区表或空闲分区链中下相邻空闲区的分区大小为两者之和,起始地址为回收区的起始地址。

(5) 如果两相邻区都不是空闲区,则将回收区插入到空闲分区表之前。若是空闲分区表,则将包括该空闲分区在内的所有表目先都依次下移一格,再插入;若是空闲分区链,则在空闲分区表和前一个空闲分区之间增加一个链结点,插入回收区的内容,结束回收操作。则将回收区作为一个新的空闲分区插入空闲分区表或空闲分区链。

动态重定位分区管理

内存紧凑

无论采用何种分配算法,动态分区管理都将产生外碎片。作业刚开始装入内存时,由于不受分区大小限制,可以提高内存利用率,但随着时间的推移,多道程序之间的空闲内存会因过小而不能再行分配,结果浪费的空间越来越多。动态重定位分区管理就是针对外碎片提出的改进方法,其基本思想是在动态分区管理的基础上,当找不到足够大的空闲分区满足用户作业需求且所有空闲分区总大小能满足作业需求时,将内存中所有作业移动,使小分区拼接成大分区,再进行分配。这种方法称为“拼接”或“紧凑”。

“紧凑”一般会使用户程序在内存中的位置发生变化,必须进行地址转换(重定位),否则

程序将无法执行。动态重定位需要有硬件支持，包括有重定位寄存器和加法寄存器，为了实现存储保护，还必须采用一对界限地址寄存器。

图 分配算法

动态重定位分区分配算法和动态分区分配算法基本相同，惟一的区别在于动态重定位分区管理的“紧凑”功能，当找不到足够大的空闲分区时，先紧凑，然后再分配。具体分配算法流程图如图 远影缘所示。

动态重定位分区管理的优点是可以消除碎片，能有效利用主存空间，提高多道程序系统的多道程度，提高处理器和 随机设备的利用率。缺点是需要硬件机构支持，增加了计算机成本，降低了速度；另外，动态重定位过程中的紧凑工作需要较大的系统开销。

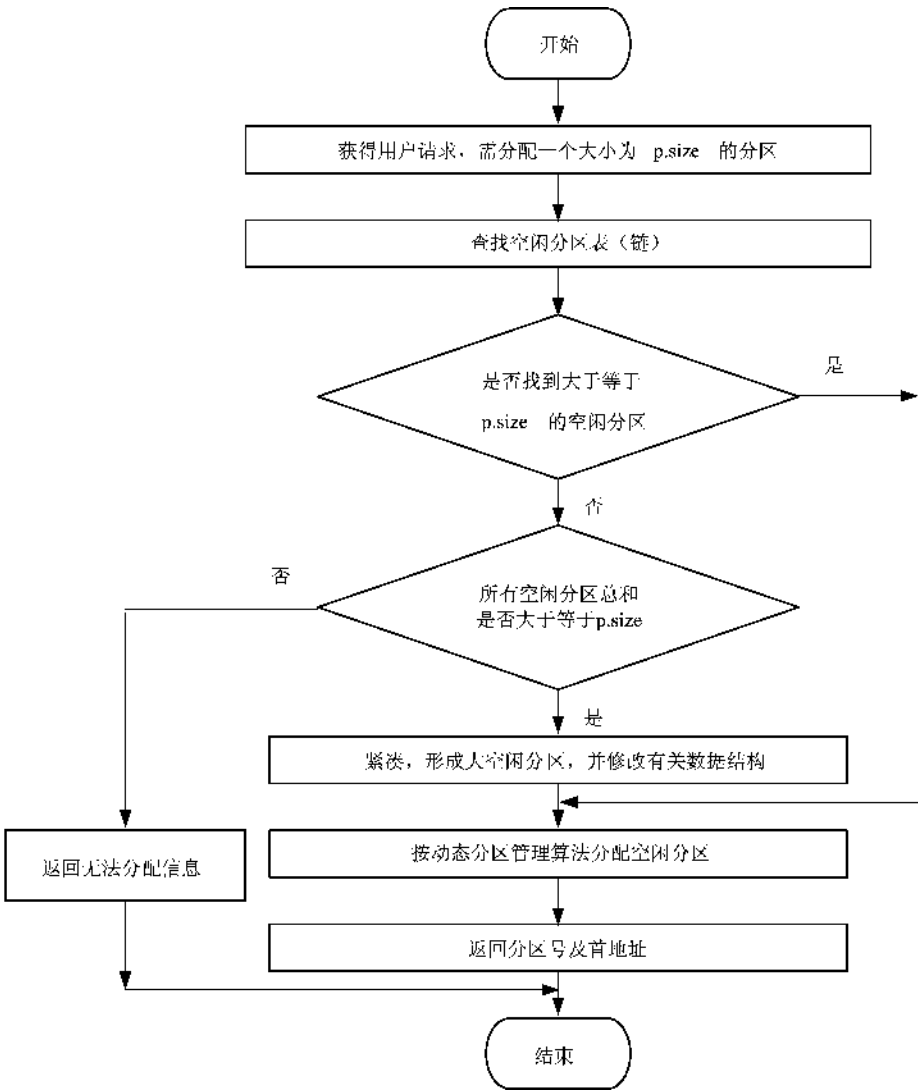


图 远影缘动态分区分配算法工作流程图

远章 覆盖与交换技术

远章 覆盖与交换技术

任何时刻，进程当前运行所必需的指令和数据都保存在内存中；而当前进程不需要执行的其他指令和数据暂时不入内存，需要时再装入到该进程不再使用的指令和数据所占用的内存空间中。这就是覆盖技术的基本思想。引入覆盖技术的目的是使在较小的可用内存空间上运行较大的程序成为可能，该技术常与分区存储管理技术配合使用。

图 远源所示为一汇编程序的覆盖过程。该汇编程序分两步执行：第一步，为程序创建一个符号表；第二步，产生机器语言代码。可以将汇编程序的代码段和数据分为四个部分：孕₁ (源K)、孕₂ (员K)、孕₃ (源K)、孕₄ (员K) 为第一步和第二步所使用的代码段及大小；阅₁ 表示符号表数据，其大小为 缘K；孕₅ 是整个汇编过程中所涉及的通用函数和程序，其大小为 源K。按照分区分配的思想，程序的运行需要 圆K 的内存空间，若现内存空闲空间为 圆K，则程序无法运行。

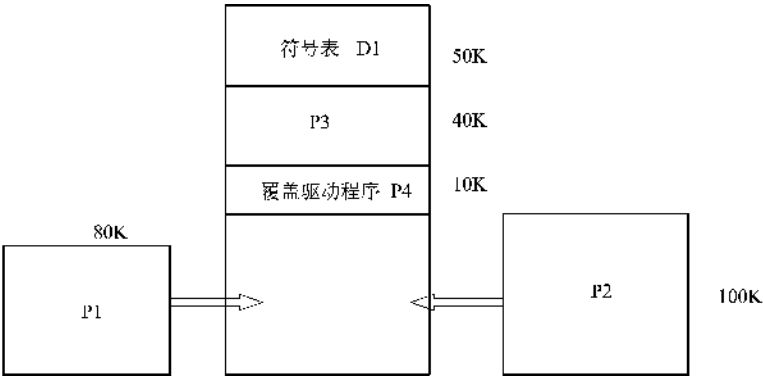


图 远源 汇编程序的覆盖示意图

采用覆盖技术可以让该汇编程序运行，因为 孕₁ 和 孕₂ 两段代码不是在同一时段需要，通过加载覆盖驱动程序 (设为 孕₄，大小为 员K)，在执行完 孕₁ 后，转向 孕₂ 运行，将 孕₂ 写入到 孕₁ 所占用的空间，再将运行控制权交给 孕₄。这样，整个汇编程序运行所需空间只为 员K。

覆盖技术的优点是：(员 由于执行前装入的数据和程序变少，程序的装入加快；(圆 不需操作系统额外的硬件支持，只需增加一些 覆盖的操作。缺点是：(员 编程人员必须设计和编写正确的覆盖结构，必须划分程序模块并确定程序模块间的覆盖关系，编程复杂度增加；(圆 必须有重定位算法和链接程序支持覆盖驱动程序读入代码的过程；(猿 覆盖技术是通过程序执行时间的延长来换取空间的节省。基于这些缺点，覆盖技术的使用并不是很广泛，但当缺乏对先进技术的硬件支持时，采用这种策略比较有效。

远章 覆盖与交换技术

员 交换技术的基本概念

交换技术在 圆世纪 远年代初期提出，最早出现在麻省理工学院的兼容分时系统 悦

中。早期操作系统版本引入了交换功能，并一直保留至今。交换技术，也称为对换或滚入滚出（或程序交换或程序移动），是指把内存中暂时不能运行的进程或暂时不用的程序和数据，换出到外存，以腾出足够的内存空间，把已具备运行条件的进程或进程所需要的程序和数据，换入内存运行。它是提高内存利用率的有效措施，是为了解决内存不足而无法同时容纳多达数十甚至数百个用户程序运行的问题而提出的。

按交换单位的不同，可分为整体交换和部分交换。整体交换也称为进程交换，是指交换整个进程，这是为解决内存紧张问题提出的，也可进一步提高内存利用率。部分交换指交换以“页”或“段”为单位进行，可分别称为“页面交换”和“分段交换”，这是在虚存的请求分页和请求分段存储管理基础上提出的。操作系统必须提供交换空间的管理和进程换入换出操作来支持交换的实现。

与覆盖技术相比，交换技术不要求程序设计人员给出程序段之间的覆盖结构，它主要是在进程或作业之间进行，而覆盖技术则主要是在同一个进程或作业之间进行。交换技术的运用，可以在较小的存储空间中运行较多的作业或进程，而覆盖技术的运用，可以在较小的存储空间中运行比其容量大的作业或进程。

交换空间的管理

具有交换功能的操作系统把外存分为文件区和交换区。文件区采用离散分配来存放文件，用于提高文件存储空间利用率。交换区用于存放从内存换出的进程，其管理的主要目标是提高进程换入换出的速度，常采用连续分配方式，较少考虑外存中的碎片。

交换空间的管理也采用空闲分区表或空闲分区链来记录使用情况。设置空闲分区表必须包括交换分区首地址和交换分区长度两个属性，并以盘块为单位记录每个空闲对换分区的相关信息；空闲分区链也类似。

交换分区空间的分配与回收和内存动态分区管理的分配回收算法相同。分配可采用四种算法中的任何一种。同样，交换区的回收过程也需区分四种情况进行处理：

- (1) 回收区与插入点的前一分区相邻接。
- (2) 回收区与插入点的后一分区相邻接。
- (3) 回收区同时与插入点的前后二个分区相邻接。
- (4) 回收区的前、后没有与之相邻接的空闲分区。

换入换出操作

发生换入换出操作的时机是当内核因执行某些操作而发现内存不足的时候。其主要任务是将内存中的一些进程换出到外存，并从交换区选择进程换入到内存执行。

换出过程主要分两步，第一步，从驻留内存的进程中选择符合换出条件的一个或多个进程，这一步骤需要考虑进程状态、进程优先级、驻内存时间长短等因素。一般首选阻塞或睡眠状态的进程，再考虑就绪状态进程；先将优先级低的进程换出，同时为防止优先级低的进程很快被换出，考虑将驻内存时间长的进程先换出。第二步，将选中的进程换出到对换区。通常先考虑换出那些非共享的程序和数据段，再考虑共享段。共享段换出前，须先对共享段引用计数减1，若其结果不为0则表示仍有进程正在使用，不能换出；否则说明共享段已不被其他进程需要，可以将它们换出；换出时应首先申请对换空间，申请成功后将程序和数据写入对换区；如果在传输中未出现错误，则释放该进程所占用的内存，并对进程控制块和内存分配表等数据结构进行修改；如果内存中还有可换出的进程，继续执行上述换出过程，

直到内存中再无阻塞进程为止。

换入过程通常也分为两步：第一步，选择符合条件的进程。检查就绪队列中所有进程的状态，从中找出“就绪且换出”状态的进程，当有多个这样的进程时，首先把其中换出时间最长的进程作为换入进程。第二步，根据换入进程的大小申请内存空间。若申请成功，直接换入进程；否则，先执行换出操作，腾出足够内存空间后，再换入进程。当换入进程数大于就绪队列中进程数时，重复执行上述两步，直到所有“就绪且换出”状态的进程全部换入，或已无法获得足够大的内存来换入进程时为止。

交换技术常与分区存储管理配合使用，其优点是可以增加并发运行的程序数目，并给用户提供了适当的响应时间，交换技术不增加编程复杂度，不影响程序结构。缺点也非常明显，上下文切换的开销非常大，没有考虑程序执行过程中地址访问的统计特性，而且在交换过程中需考虑程序换入时的重定位问题和交换中传送的信息量多少问题。

2. 分页存储管理

分页存储管理是将进程连续逻辑地址空间转换成主存中不连续物理地址空间的一种方式，是为了解决内存连续分配所产生的过多碎片，以及由于采用紧凑处理碎片而造成的过大的系统开销。分页存储管理又称为静态页式管理方式。

1. 基本原理

(1) 内存物理地址空间按一定的指数幂等分，从 0 开始连续编号，称为页框或页架 (frame)。

(2) 程序逻辑地址空间按页框大小分为若干片，依序从 0, 1, … 编号，称为页面 (page)。

(3) 程序逻辑地址空间从基地址“0”开始连续编址，用 P 表示逻辑地址，其中 $P / \text{页大小}$ 为逻辑地址所在页面号， $P \% \text{页大小}$ 为该地址所在页框中的相对地址，称为页内地址或偏移量。假设页面大小为 L ，某一维线性地址为 P ，则

$$P = \text{页框号} \times L + \text{页内地址}$$

其中 $\text{页框号} = \lfloor P / L \rfloor$ ， $\text{页内地址} = P \% L$ 。

(4) 以页为单位加载程序，将程序的多个页面装入到不连续的页框中，从而实现离散分配。由于程序最后一页经常装不满一个页框而将形成不可利用的碎片，称为“页内碎片”。

(5) 设置页表地址寄存器，用于保存当前执行进程页表的起始地址和长度。

页面的大小是由机器的地址结构决定的，要综合考虑内存利用率和换入换出的效率，不能太大，也不能太小。通常页面的大小是 2 的幂，如 4096 字节。

2. 数据结构

目前分页存储管理的数据结构主要有页表、两级页表和多级页表、哈希页表、反置页表等四种。

1. 页表

在分页存储管理中，程序的多个页面存储到不连续的多个物理页框中，为保证进程的正确运行，且能找到每个页面对应的页框，必须为每个进程记录相关信息，即需要设置页面映

射表，简称页表。页表用于保存程序各页入内存后的情况，包括页号和页框号，采用页表可以实现从页号到页框号的地址映射。

也可在系统设置进程页表、物理页面表和请求表三种数据结构。进程页表一个进程一张，主要完成逻辑页号到物理页框号的映射；物理页面表整个系统一张，用于描述物理内存空间的分配使用情况，可采用位示图或空闲页面链表等描述；请求表也是整个系统一张，用于描述系统内各进程页表的位置和大小，一般放在各进程的页表中。

两级页表和多级页表

大多数现代计算机系统支持很大的地址空间，由此页表本身将占用相当大的空间。若要求页表连续存储，显然不现实。为解决大型页表存放问题，可采用两级或多级页表方法。

两级页表的思想就是将页表再进行分页，将页表的各个页面存放在不连续的内存空间。为分页后的页表建立一张外层页表，外层页表的表项中记录页表页面的物理页框号。以 32 位逻辑地址空间为例，设页面大小为 4K 字节，每个表项由 4 字节组成，一级页表结构的页表项为 8192 个，需 32K 内存空间；若采用两级页表结构，则外层页表的表项为 256 个，可以采用如图 10-15 的逻辑地址结构描述。

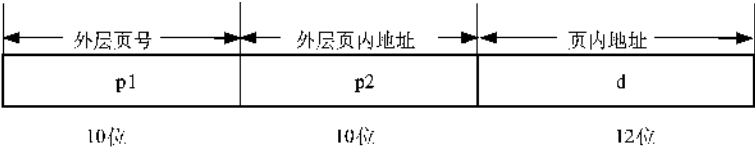


图 10-15 两级页表逻辑地址结构

然而，对于 32 位体系结构的机器，采用两级页表也不再适用，必须对页表的外层页表再进行分页，从而形成多级页表。例如，32 位的系统可采用三级页表。但是这也带来了一个问题，完成逻辑地址转换所需访问内存的次数将增加，逻辑地址转换为物理地址的时间开销增长，这是难以容忍的。

哈希页表

针对多级页表的问题，当地址空间大于 32 位时，可采用哈希页表。哈希页表中的每个表项由虚拟页号、所映射的页框号和指向链表中下一个元素的指针三个部分组成。虚拟页号为哈希值，每个表项都包含一个链表，该链表中元素的哈希值指向同一个位置。当利用哈希链表进行页式管理时，首先，由虚拟页号得到哈希值，查找哈希页表，将此页号与链表中的第一个元素的虚拟页号字段进行比较；若匹配成功，则相应的页框号就用于形成物理地址，若不成功，则沿链表依次寻找匹配的表项。

反置页表

反置页表也是为减少页表占用的内存空间而引入的。一般页表中页表项按页面号排序，而反置页表与页表不同，它按页框号排序，它为每个物理页框设置一个页表项，每个表项包含与该页框对应的页号及拥有该页面进程的标识符。整个系统只有一张页表，由于每个表项存放了该页面进程的标识符，所以可保证特定进程的逻辑页面映射到相应的物理页框上。Intel 公司推出的系统 386 和 486 便是采用反置页表，它采用了这一技术。

当利用反置页表进行地址变换时，首先，利用进程标识符和页号检索反置页表；若检索成功，则该表项的序号便是该页的页框号，该页框号和页内地址一起构成物理地址；若检索完整个页表都未找到与之匹配的页表项，则判断是否采用虚存管理；若只采用实存管理，则表示地址出错；若采用虚存管理，则表明此页尚未调入内存，应产生请求调页中断。

反置页表的优点是可有效减少页表所占用的内存。但其缺点也是明显的，第一，反置页表只包含已调入内存的页面，并未包含尚未调入内存的各个进程的页面，因此必须为每个进程建立外部页表（反置页表），用以描述进程在外存或交换区中的页面信息；第二，反置页表为每个页框设置页表项，由于页表项数目过大，从而会增加访问内存的查表时间。

远源地址转换机制

地址转换的基本任务就是将程序的逻辑地址空间转换为物理地址空间或虚存地址空间。分页存储管理系统中指令给出的地址和物理地址一一对应，因此地址变换机构的任务，实际上是将页号转换为内存中的页框号。

地址变换借助于页表寄存器（远源地址转换）和页表来完成。页表寄存器存放进程页表在内存的始址和页表长度，进程没执行前，页表始址和长度存放在本进程的远源中，调度到该进程时才装入页表寄存器。页表可采用一组专门寄存器实现，这样可提高地址转换速度，但因为寄存器成本偏高，且大多数现代计算机的页表偏大，所以进程页表一般常驻内存。

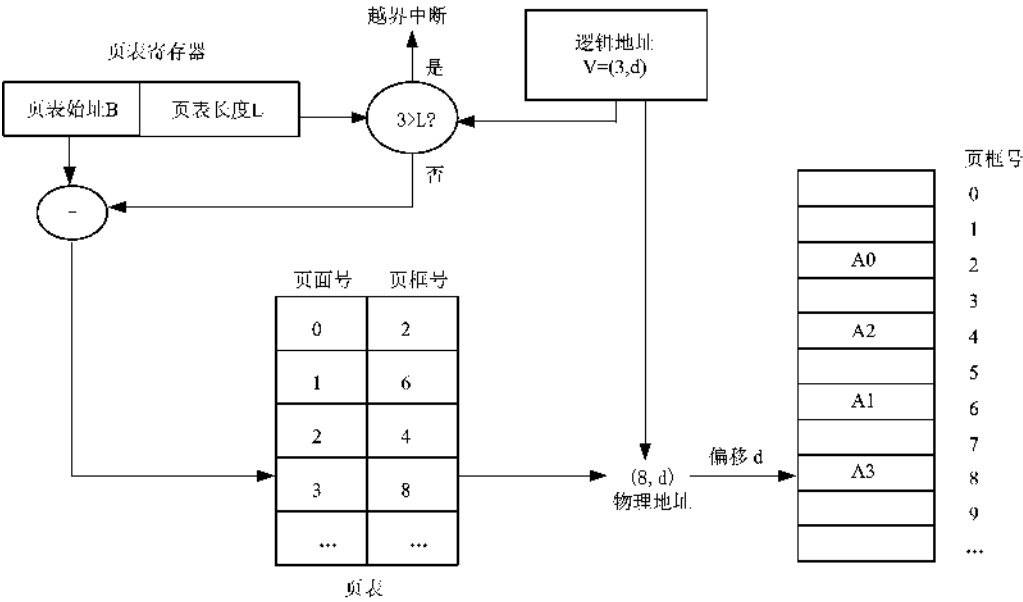


图 远源地址转换的地址转换机构

地址转换机构如图远源所示。指令所给出的逻辑地址分为两部分，逻辑页号和页内地址。进程访问某个逻辑地址的数据时，首先，系统的内存管理单元（远源地址转换）将页号与页表长度进行比较，若页号大于或等于页表长度，则表示本次访问的地

址超越进程的地址空间,产生地址越界中断;否则,从页表寄存器取出页表始址,根据页号查找页表,获得页框号,物理页框号与页内地址一起形成物理地址。这个过程通常由硬件完成,而不需要软件参与。

上述基本地址转换过程中,每次存取一个数据都需访问两次内存,第一次查找页表将操作数的逻辑地址变换为物理地址,第二次完成真正的读写操作。因此,计算机的处理速度降低近 1 倍。为了缩短查找时间,提高地址变换速度,可增设具有并行查寻能力的特殊高速缓冲存储器,称为“联想存储器”(或称“快表”),实现按内容查找。

具有快表的地址变换过程是:

(1) 读出快表给出的有效地址,由地址变换机构自动将页号与快表中的页号比较,这种比较是同时进行的。

(2) 若其中有相匹配的页号,表示快表中存在所要访问的页表项,直接读出对应的页框号,送物理地址寄存器。

(3) 若在快表中未找到对应的页表项,则访问内存中的页表,找到页框号送地址寄存器。同时,将此页表项存入快表中的一个寄存器单元中,亦即修改快表。

(4) 若快表已满,则操作系统必须根据某种算法找到一个不再需要的页表项换出。

快表的访问速度比内存的访问速度快得多,采用这种方式可以大大减少访问内存的时间。但由于成本的关系,快表的存储空间有限,所以配置快表的效果取决于对联想存储器访问的命中率。一般来讲,假设访问联想存储器的命中率为 80%~90%,则增加联想存储器后访问数据的有效访问时间要比不引入时缩短 20%~10%。

2.3.2 页面分配策略

分页存储管理的第一步就是为作业或进程分配足够的页面。系统通过存储页面表、请求表以及进程页表来完成内存的分配工作。实存页面分配过程较为简单,如图 2-3-1 所示。首先,请求表给出进程或作业要求的页面数;然后,在存储页面表中检查是否有足够的空闲页框,如果没有,则本次无法分配;否则设置页表和请求表中的相应表项,按一定的查找算法搜索出所要求的空闲页框,并将对应的页框号填入页表中。

2.3.3 分页存储管理的共享与保护

页面的共享和保护相对而言比较复杂。但是操作系统中的页面共享和保护非常重要,特别是分时系统,页面共享尤为重要。

为实现页面共享,可将代码段分为可重入代码(或称共享代码)和不可重入代码(或称独占代码)。所谓可重入代码指的是一段允许多个进程同时使用的代码,又称共享代码,可重入代码是不可修改的。可重入代码可提高内存利用率,但必须设计好可重入代码的起始地址和长度。另外,采用反置页表结构的页式管理机构很难实现共享。

只有可重入代码才能被多个进程共享,不可重入代码不能被多个进程共享;各进程在共享可重入代码时,必须使用相同的页号。

分页存储管理系统的页面保护一般采用存取控制字段来完成。存取控制字段通常保存在页表中,实现存储块内容的保护。当存取控制字段仅有一位时,称为保护位,用来描述该存储块中的内容是可读写或只读属性;若存取控制字段为两位,则可规定为读、只读和只执

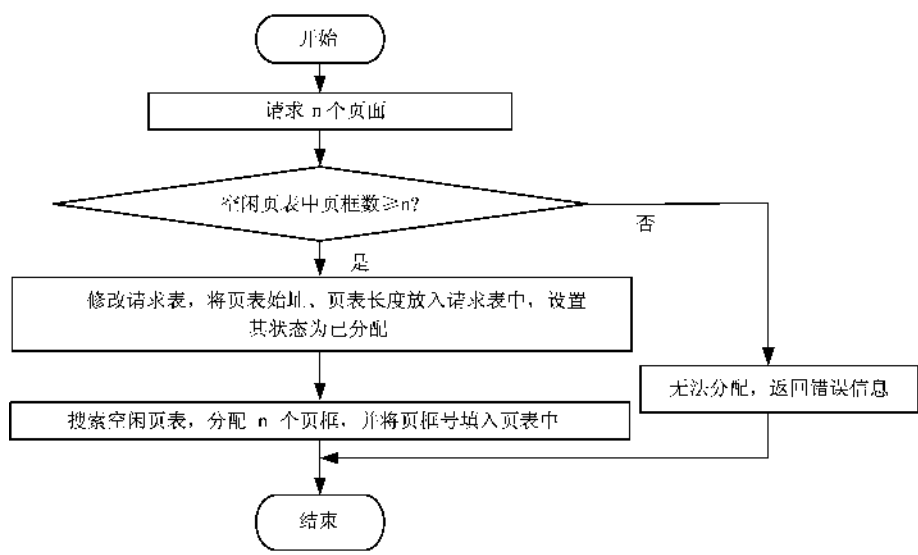


图 4-1 分页存储管理的页面分配过程

行等存取方式。进程查找页面时，从页表中得到正确的页框号计算物理地址，同时通过保护位验证该页的读写权限。若某一进程试图写一个只读存储块，则将引起操作系统的一次中断。

4.2 分段存储管理

分段存储管理（Segment Storage Management）主要是为了满足用户在编程和使用上多方面的要求而引入的另一种离散分配存储管理方式。这些要求包括有：

- ① 方便用户编程。用户希望能按照程序模块来划分段，并按段分配内存。
- ② 分段共享。能实现以信息逻辑单位为基础的分段共享方式，而在分页系统中每一页都只是存放信息的物理单位，不便于实现信息共享。
- ③ 分段保护。能实现多道程序环境下，内存中以信息的逻辑单位为基础的保护，采用分段的组织和管理方式，对于实现保护功能，将是更有效和方便的。
- ④ 动态链接。在作业运行之前，并不把几个目标程序段链接起来，而是先将主程序所对应的目标程序装入内存并启动运行，当运行过程中又需要调用某段时，才将该段（目标程序）调入内存并进行链接。动态链接要求以段作为管理的单位。
- ⑤ 动态增长。在实际使用过程中，往往有些段特别是数据段会不断地增长，但在运行前或运行中是无法确切知道数据段会增长到多大的。这种情况的出现使得分段存储管理方式得到发展。

4.2.1 分段基本原理

分段管理的基本思想就是把程序按内容或过程（函数）关系分成段，每段有自己的名字。

一个用户作业或进程所包含的段对应一个二维线性虚拟空间，也就是二维虚拟存储器。分段管理程序以段为单位分配内存，通过地址映射机构把段式虚拟地址转换为实际内存物理地址。程序可划分为多个模块，如代码段、数据段、共享段。具体实现分为以下几个步骤：

(1) 将作业的地址空间划分为若干个段，每个段定义一组逻辑信息。如主程序段、子程序段、数据段、栈段等，每个段设置段名，并从连续编号开始...

(2) 装入程序分段装入，并为每个分段分配段号。作业的逻辑地址表示为(段号，偏移量)，其中段号为逻辑地址所在的段号，偏移量为逻辑地址在该段中的偏移量，或称相对地址。

(3) 系统以段为单位分配主存，每一段分配连续的分区，段表保存作业的各段进入内存后的情况。设置段表地址寄存器，保存当前执行进程段表的起始地址和长度。

段长由相应逻辑信息组的长度决定，各段长度不等。整个作业的地址空间是二维结构，由段号(名)和段内地址组成。段号之间无顺序关系，每段定义的是一组逻辑上完整的程序或数据。每段都是首地址为零、连续的一维线性空间，段长可根据需要动态增长。

分段存储管理的优点在于没有内碎片，通过紧凑可消除外碎片，便于实现内存共享和保护，并可针对不同类型的段采用不同的保护机制和共享机制。

2.3.2 段式存储管理的数据结构和地址转换机制

2.3.2.1 段式数据结构

为使程序能正常运行，亦即能从物理内存中找出每个逻辑段所对应的位置，操作系统必须为每个进程建立一张段映射表，称之为“段表”。每个段在表中占有一条表项，用于记录该段在内存中的起始地址(又称为“基址”)和段的长度。段表存放在一组寄存器中，这有利于提高地址转换速度，但更常见的是将段表放在内存。配置段表后，执行中的进程通过查找段表，找到每个段所对应的内存区，从而实现从逻辑段到物理内存区的映射。

2.3.2.2 段式地址转换机制

分段管理系统中，整个进程的地址空间是二维结构，为实现从进程的逻辑地址到物理地址的映射，系统设置段表寄存器，用于存放段表基址和段表长度。当进行地址变换时，系统首先将逻辑地址中的段号与段表长度进行比较，若段号大于段表长度，表示段号太大，访问越界，则产生越界中断信号；若未越界，则查找内存中的段表，由段号得到段的首地址。然后，再检查段内地址是否超过该段的段长。若超过，发出越界中断信号；若未越界，则将该段的基址与段内地址相加，得到实际的物理地址。

与分页存储管理系统相同，段表存放在内存，每访问一次数据，都必须访问两次内存，从而成倍地降低了处理速率。解决方法和分页系统类似，增设联想存储器，用于保存最近常用的段表项。一般情况下，段表项数目比页表项数目少，其所需的联想存储器也相对较小，所需查找时间减少，因此可以显著地减少存取数据的时间。采用联想存储器能使存取速度仅慢约1%~5%。

2.3.3 段的共享与保护

段是信息的逻辑单位，分段存储管理的最大优点就是易于实现段的共享和保护，即允许若干进程共享同一段，允许对不同段进行不同的保护策略。和分页管理机制类似，分段存储管理将代码分为可重入代码(又称纯代码，允许多次重复执行)和不可重入代码。可重入代码不允许

任何进程对其进行修改。和分页管理系统中页面共享和保护相比，分段存储管理系统中，段是逻辑上独立的单位，因而更加方便实现共享和保护。但为了实现段共享，在系统中必须要设置相应的数据结构、共享段的操作以及保护机制。

4.2.1 共享段表

为实现段共享，系统必须设置一张共享段表，所有共享段都在共享段表中占用一条表项，用于记录共享段的信息和有关共享此段的进程信息，如图 4-2-1 所示，具体包括有：

(1) 段表信息，即共享段的信息，包括段名、段长、内存始址、状态、外存始址等信息。

(2) 共享进程计数，用于描述多少个进程同时共享该共享段。当某进程不再需要共享时，系统将计数减 1，只有共享进程计数值为 0 时，系统才回收该段所占内存空间。

(3) 存取控制字段，用于描述不同进程对该共享段的存取权限。

(4) 进程信息，即共享该段的进程的信息，包括进程名和进程号。

(5) 段号，用于描述不同进程共享该段的段号。

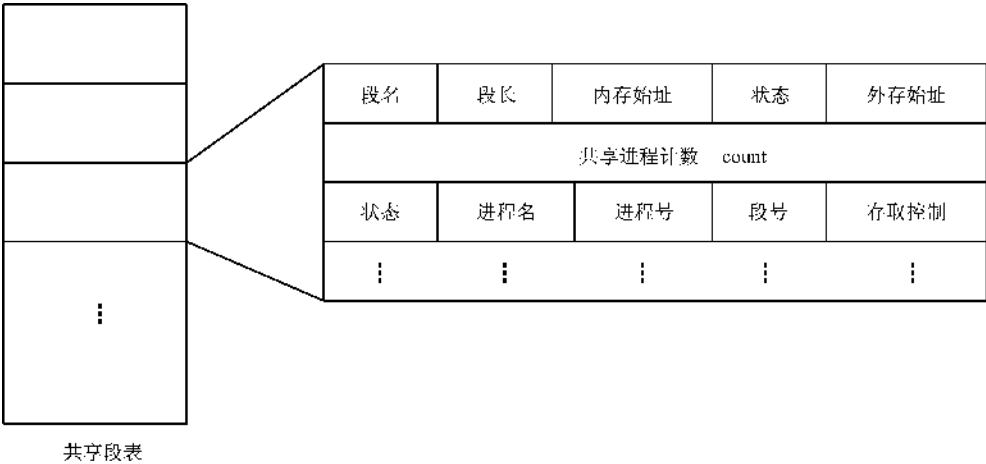


图 4-2-1 共享段表示意图

4.2.2 段的保护

分段存储管理中常采用以下几种措施来实现段保护：

(1) 越界检查。段表地址寄存器中存放段表长度信息，每个段表项包含段长字段，在逻辑地址转换为物理地址时需经过两步检查：第一步，判断段号是否越界，第二步，判断段内偏移量是否越界，从而保证每个进程只能在自己的地址空间中运行。

(2) 存取控制检查。段表的每个表项中设置了存取控制字段，用于规定对该段的访问方式，包括只读、只执行和读写权限。这样可以保证不同权限的进程对段的正确使用，以及信息的安全性。特别是对于共享段，该字段显得尤为重要。

(3) 环保护机制。该机制是将所有进程进行分层，最靠近圆心的层（环）称为 0 环，具有最高特权，该层属于操作系统内核，管理 I/O 操作、存储管理等功能，在该层中可使用特权指令，可访问所有段和所有页面；某些重要的实用程序和操作系统服务，如系统调用、管理程序等，占据 1 环，具有次高特权；共享库的过程和函数具有更低一级的特权，占据 2 环；而

一般的应用程序安排在 环上，具有最低的特权。一个进程限制自己访问相同环或较低特权环中的段和数据；需要访问较高特权环中的数据或段时，必须通过调用被访问环所提供的服务来完成。

远源源 分页存储管理和分段存储管理的区别

分页存储管理和分段存储管理的共同点体现在两者都采用离散分配方式，且都要通过地址映射机构来实现地址变换。但在概念上两者完全不同，其主要区别表现在以下三点：

(员 页是信息的物理单位，分页管理是为了实现离散分配方式，以减少内存的外零头，提高内存的利用率。或者说，分页管理是出于系统管理的需要；而段是信息的逻辑单位，含有一组意义相对完整的信息，分段管理的目的是为了能更好地满足用户的需要。

(圆 页的大小固定且由系统确定，逻辑地址由页号和页内地址组成，可由机器硬件实现；段长不固定，取决于用户所编写的程序，通常由编译程序在对源程序进行编译时，根据信息的性质来划分。

(猿 分页管理中，作业地址空间是一维的，是单一的线性地址空间；而分段管理中，作业地址空间是二维的，程序员在标识一个地址时，既需给出段名，又需给出段内地址。

远源远 段页式存储管理

分页存储管理和分段存储管理都有各自的优缺点。事实上，远源源和远源源系列都试图将两者进行结合，形成段页式管理模式。其基本思想描述如下：

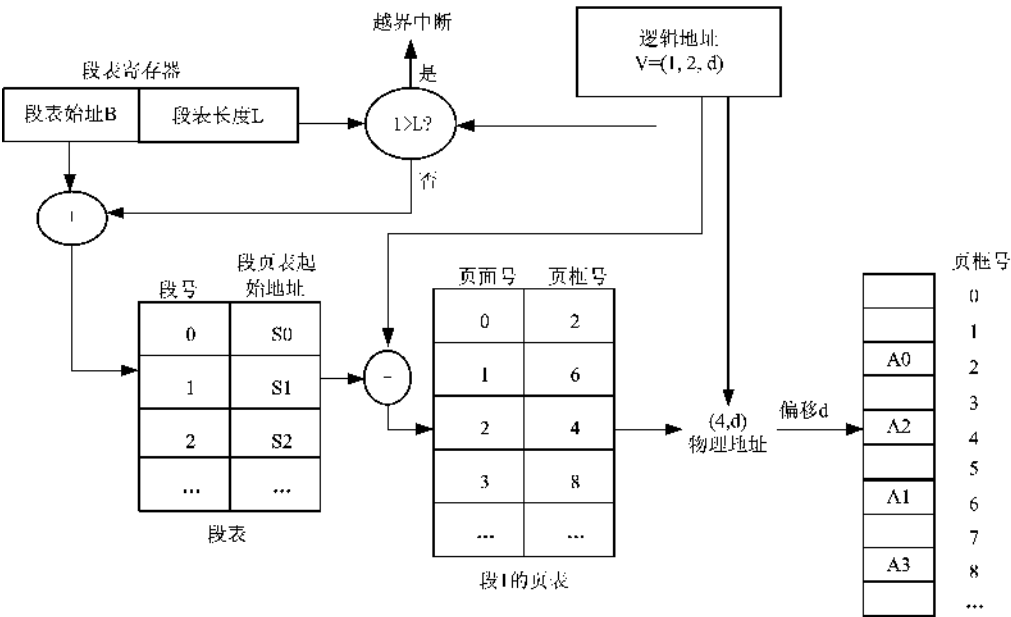


图 远源源 段页式管理的地址转换机构

(员 首先内存空间按 圆的指数幂等分,称为页框或页架(葉架),并进行连续编号 园 员 圆 ...

(圆) 将作业地址空间分段, 每段设置段号, 从园车续编号, 园员圆 …

(豫) 在作业的每段内,按内存页框大小分页面,从园开始对各页连续编号园员圆 …

④ 逻辑地址表示采用三维数组 约 责 凿, 其中 泽 为逻辑地址所在的段号, 责 为逻辑地址在该段中的页面号, 凿 为该地址所在 页 责 中的页内偏移量。

(缘 系统以页面为单位分配内存给每个进程。

④ 为进行地址转换，系统为每个进程建立一张段表，并为该进程的各段分别建立一张页表。同时采用段表地址寄存器来指出进程的段表起始地址和段表长度。地址转换机构如图 4-10 所示。

段页式存储管理是分页技术和分段技术的结合，因此它具有两种技术的全部优点，包括不存在外碎片、段可动态增长、便于共享和控制存取访问。但其中也存在硬件成本增加、软件复杂性和管理开销增加以及存在内碎片等缺点。

本章小结

存储器管理在操作系统中占有重要地位，其主要功能包括主存分配和管理、地址转换和重定位、存储保护和共享以及存储扩充等四个方面。

用于多道程序操作系统的存储管理基本方法可以有多种，包括固定分区、动态分区、动态重定位分区等三种常见连续分配存储管理方式，以及分页存储管理、分段存储管理和段页式存储管理等离散分配存储管理方式，一个特定系统采用哪种策略取决于系统硬件提供的支持。这些存储管理方式在很多方面都不尽相同。

固定分区管理将内存划分为若干个固定大小的区域,以分区为单位运行多道作业;动态分区管理根据作业大小连续分配内存,采用动态重定位方式实现地址转换;动态重定位分区管理在动态分区管理的基础上,增加对碎片的整理以提高内存利用率。这三种方式都是以分区为单位管理内存,采用界限寄存器对存储进行简单的保护,而信息共享一般较难实现。

引入覆盖技术的目的是在较小的可用内存空间上运行较大的程序。交换技术是提高内存利用率的有效措施。这两种技术常与分区存储管理配合使用。覆盖技术的优点是不需要操作系统额外的硬件支持，在增加一些辅助操作的基础上能使程序的装入加快；缺点是加大了编程复杂度，必须有重定位算法和链接程序的支持，且其实现是通过时间延长来换取空间节省。交换技术的优点是可以增加并发运行的程序数目，为用户提供适当的响应时间，且不增加编程复杂度，不影响程序结构，缺点是上下文切换的开销非常大。

分页存储管理以页为单位为作业分配不连续的内存地址空间，向作业提供越界保护和存取控制保护，可以实现信息共享，但限制多。分段存储管理以段为基本单位，通过动态重定位机构将二维逻辑地址空间转换为一维的物理地址空间，它同样提供对作业的越界保护和存取控制保护，但较分页存储管理而言，其信息共享更容易实现。段页式存储管理方法结合考虑分页存储管理和分段存储管理方式的基本思想，所以它拥有两者的所有优点，同时也有两者所有的缺点。

习 题

- 1. 存储管理的主要功能包括有哪四个方面？
- 2. 用户程序在计算机中主要分为哪些处理阶段？
- 3. 可采用哪几种方式将程序装入内存？有哪几种链接方式？
- 4. 什么是重定位？动态重定位的概念是什么？如何实现？
- 5. 动态分区分配方式中有哪些分配算法？在回收的时候，应该如何做？
- 6. 什么是覆盖？为实现交换技术，系统应具备哪些条件？换入换出操作如何实现？
- 7. 分页存储管理的基本思想是什么？分页管理中的页表中各项的作用是什么？需要哪些硬件支持？

考虑以下段表

段号	段基址	段长
0	1000	100
1	2000	100
2	3000	100
3	4000	100
4	5000	100

- 试分别求出逻辑地址 (0, 100), (1, 200), (2, 300), (3, 400), (4, 500) 的物理地址是多少？
- 8. 为什么说分段存储管理较之分页存储管理更容易实现信息共享和保护？
 - 9. 分页和分段的主要区别是什么？

第 7 章 虚拟内存管理

第 7 章我们讨论了各种存储管理的不同策略，所有的这些策略都有一个共同的目标，那就是提供多道程序并发执行，但是这需要执行前将整个程序都装入内存。为了在有限内存空间运行更多、更大的程序，可以利用虚拟存储技术。本章主要介绍虚存提出的背景、定义和特征，并重点介绍采用虚存技术的两种存储管理的实现，并简要说明 8086 系统中的存储管理技术。

7.1 虚拟内存基本概念

由于物理内存的容量是有限的，当用户程序所要求的内存空间超出了内存总容量时，采用实存管理策略，作业将得不到执行；或当大量用户程序要求同时运行，则可能造成多数程序必须等待。增加物理内存，将使系统成本提高，同时还要受到机器本身的限制。虚拟存储技术提供了一个扩充存储容量的方法，借助虚拟存储技术，运行程序将不受主存大小限制，从逻辑上扩充内存容量，使用户感觉可运行任意大小的程序，或可让更多的用户程序并发运行。

7.2 虚拟内存程序访问局部性原理

所谓程序访问局部性（~~程序访问局部性~~）原理是指 1955 年 1955 年 1955 年对指令和数据的存取在时间上、空间上和顺序上往往集中在一定的范围内。1955 年 1955 年 1955 年的研究表明，程序的执行呈现局部性规律，即在一段较短时间内，程序的执行仅限于某个部分；相应地，它所访问的存储空间也局限于某个区域。可以把局部性原理总结为以下三点：

（1）时间局部性。由于程序存在循环结构、临时变量和子程序调用，如果某条指令被执行或某个数据结构被访问，则在不久的一段时间内该指令可能再次执行、该数据结构可能再次被访问。

（2）空间局部性。若某个存储单元被访问，则在不久之后，其附近的存储单元也可能被访问。即程序在一段时间内所访问的地址，可能集中在一定的范围内。

（3）顺序局部性。通常情况下，指令跟踪程序的执行是按照在主存中的连续地址进行的。只有在遇到转移指令时，才会跳转到一个不连续的地址空间，但跳转到新的地址空间后，指令又按顺序执行。

由于程序遵循局部性规律，可以只把当前作业所需要的那部分程序和数据装入内存就能启动运行。这样用户程序不再受可用内存容量的限制，简化了程序设计任务，用户程序的装入和交换的数据量减少，增加了多道程序的度，提高了 8086 利用率和系统吞吐量，加快了程序的运行。

2.1.1 虚存的定义

所谓虚存，就是用户能作为可编址内存对待的存储空间，它是由操作系统提供的一个假想的特大的存储器，它不是实际的内存，而是操作系统对物理内存的逻辑扩充。

虚存技术就是基于程序运行的局部性规律，将大程序中当前正在使用的部分放在内存，而其余部分放在磁盘，运行过程中，操作系统根据程序执行时的要求和内存的实际使用情况，换入换出程序其余部分，为用户提供比真实内存空间大得多的地址空间的技术。

实现虚存技术需要二级存储结构（如硬盘）和动态地址转换机制的支持。理论上虚存是“无限大”的，但实际上虚存大小受地址字长的限制，如果表示地址的字长为 2^m 位，则最大的地址空间是 2^m 。另外，从实现的观点看，用户程序和数据都必须完整地保存在外存中。所以虚存并不能达到无限大，其传送速度也不是无限快的，要受外存的容量、传送速度和使用频率等各方面物理因素的限制。用户的虚拟空间一般不能超过硬盘中作业的存放空间。

2.1.2 虚存的特征

虚拟存储器具有四个基本特征，分别是离散性、多次性、对换性和虚拟性，其中最基本的特征是离散性，所表现出来的多次性是虚拟存储器的重要特征。

（1）离散性

离散性是指内存分配采用离散分配方式。即将一个作业分成多个部分，根据执行需要将部分装入内存中不连续的地址空间，为作业动态装入提供方便，从而实现虚拟存储器功能。

（2）虚拟性

虚存是从逻辑上扩充内存容量，使用户编程所用到的地址空间远大于实际内存容量。虚拟性是虚拟存储器的重要特征，也是实现虚拟存储器管理的最重要目标。

（3）多次性

多次性是指作业不是一次性装入内存，而是被分成若干部分，分多次调入内存。亦即在作业运行前不必将作业全部装入，只需将当前要运行的那部分程序和数据装入内存；以后运行到哪一部分就再分别将它装入或换入内存。多次性是虚拟存储器的重要特征，任何其他的存储管理方式都不具有这一特征。可以认为，虚存是具有多次性特征的存储器系统。

（4）对换性

对换性是指系统允许作业的部分在运行过程中换入内存或换出到对换区。进程运行期间，允许将内存中暂不使用的程序和数据换出至外存的对换区，等待以后需要时再将其换入到内存；还允许将暂时不运行的进程调至外存，待它们重新具备运行条件时再调入内存执行，以腾出尽量多的内存空间供其他进程运行。换入换出过程在进程运行期间可发生多次。

2.1.3 虚存的实现方法

虚存的实现方式是建立在离散分配存储管理方式的基础上的，主要有以下几种方式：

（1）请求分页管理（请求段方式管理）。是在实存页式管理的基础上建立的一种管理机制。其基本思想是作业运行的过程中，页面只有需要时才请求被换入内存，而暂不运行的页面置换到外存，从而减少对换时间和所需内存数量，增加多道程序的道数。换入和换出以页面为基本单位。

系统必须提供必要的硬件支持来实现请求调页和置换功能,包括有页表机制、缺页中断机构和地址变换机构。页表机制是请求页式管理的数据结构,缺页中断机构是在缺页时的处理机制,地址变换机构是进行页面访问时,用户逻辑地址空间到内存物理地址空间的映射。另外操作系统必须提供缺页中断处理和地址变换的软件功能。

④ 请求分段管理 (请求页式管理的变种)。这是在实存分段管理基础上建立的,基本思想是作业运行前,只装入需要用到若干段用户程序和数据,而将暂时不用的段换出到外存。这是通过增加请求调段和分段置换功能来完成的。换入和换出以段为单位进行。

同样,为实现请求分段管理,系统必须提供必要的硬件支持,包括段表机制、缺段中断机构和地址变换机构。

另外,在现有的一些操作系统中,还建立了基于段页式管理机制的段页式虚存管理机制,如 Intel 80386 微处理器可支持段页式虚存管理。

9.4 虚存请求分页存储管理

虚存请求分页管理 (请求页式管理) 也称为动态页式管理,是在分页存储管理的基础上发展起来的。两种机制的思想基本相同,其主要的不同之处在于:在请求分页管理中,只有当前需要的一部分页面在内存;而分页存储管理中,一个进程的所有页面全部在主存中。

虚存请求分页管理将内存的物理空间扩展到虚存的物理地址空间,即允许内存中只存在一个进程的若干页,便可启动进程运行,该进程的其余页面存在内存以外的虚存空间中。

9.4.1 虚存数据结构的组成

请求分页管理中的主要数据结构还是页表,其基本作用是实现用户逻辑地址空间到物理地址空间的映射。它对分页存储管理的页表机制进行了扩充,包括以下内容:

(1) 页号,用于描述程序逻辑地址空间的页面号。

(2) 页框号,描述页面装入后的内存页框编号。

(3) 状态位,或叫存在位,用于指示该页是否已经调入内存,供操作系统实现虚拟存储功能时参考。

(4) 访问字段,用于记录本页在一段时间内被访问的次数,或记录最近已经有多长时间未被访问,提供给置换算法在选择换出页面时参考。

(5) 修改位,表示该页在调入内存后是否被修改过。虚存管理中系统为内存中的每一页在外存都保留了一份副本,当该页以写的方式访问页面时,设置该页的修改位,在置换该页时必须将该页回写到外存上,以保证外存中所保留的始终是最新副本;若没修改,则在置换时不需将该页写回外存,从而减少系统的开销和启动磁盘的次数。该位也可由操作系统软件来修改,在保存到磁盘后,修改位置位。

(6) 外存地址,用于指出该页在外存上的地址,供调入该页时使用。

9.4.2 虚存地址转换

9.4.2.1 缺页中断机构

请求分页管理中,如果当所要访问的页面不在内存,便产生缺页中断,请求操作系统将

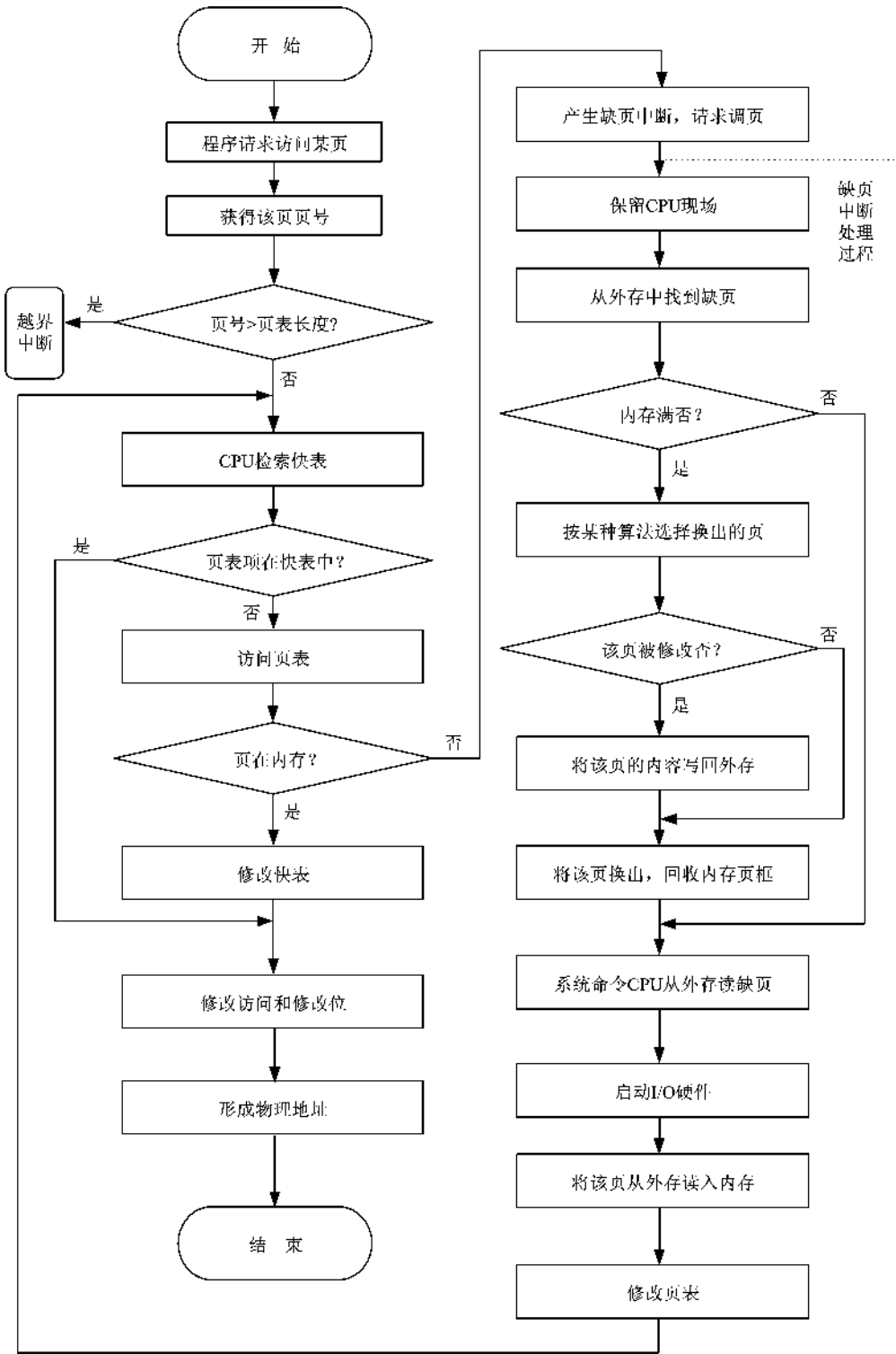


图 页请求分页系统中的地址转换过程

所缺的页面调入内存。缺页中断作为中断，它同样需要经历诸如保护现场环境、分析中断原因、转入缺页中断处理程序进行处理、恢复现场环境等几个步骤。但缺页中断又是一种特殊的中断，它与一般的中断相比，有着明显的区别，主要表现如下：

第一，在指令执行期间产生和处理中断信号。通常，系统都是在一条指令执行完后去检查是否有中断请求到达。若有，便去响应；否则继续执行下一条指令。然而，缺页中断是在指令执行期间，发现所要访问的指令或数据不在内存时产生和处理的。

第二，一条指令执行期间，可能产生多次缺页中断。基于这些特征，系统的硬件机构应能保存多次中断时的状态，并保证最后能返回到中断前产生缺页中断的指令处，继续执行。

7.2.2 地址变换机构

请求分页管理的地址变换机构是在分页存储管理的基础上，增加了某些功能而形成的，如产生和处理缺页中断、从内存中换出一页的功能，等等。图 7-2-1 描述了请求分页管理中的地址变换过程。地址变换机构首先检索快表，试图从中找出所要访问的页；若找到，便修改页表项中的访问位，对于写指令，还须将修改位置为“1”，然后利用页表项中给出的页框号和页内地址，形成物理地址；若没有找到，则查找内存的页表，通过状态位确定该页是否调入内存，若已调入内存，则从页表中查出页框号形成物理地址，并将此页的页表项写入快表；当快表满时，应先按照某种算法换出一些页表项，然后再写入；若访问页还没有调入内存，则产生缺页中断，请求操作系统从外存中把该页调入内存，再进行处理。

同样，为了提高有效访问时间，请求分页管理也可以采用具有快表的地址转换机制，为存储大型页表，也可以有多级页表的地址转换和反置页表的地址转换。这些机制和分页存储管理中相对应的机制类似，在此不一一详述。

7.3 虚拟页面调度策略

由于增加了虚拟内存技术，进程的页面有一部分在虚拟存储器上，所以请求分页管理的页面调度策略相对于分页存储管理的策略要复杂得多，具体从三个方面加以说明。

7.3.1 虚拟页面调入策略

为使进程能正常运行，必须事先将要执行的那部分程序和数据调入内存。虚存页面调入策略定义的是系统何时将所需页面由外存调入内存中，以及从何处调入所需页面。

(1) 何时调入页面

常用的两种调入策略有预调页策略和请求调页策略。预调页策略是在发生缺页时，采用局部性原理，一次调入该页和若干相邻的页面。这种策略能提高调页的效率。缺点是采用以预测为基础的预调页策略，若调入的页在以后的执行过程中很少被访问，则造成低效率。目前预调页的成功率仅约 30%，故主要用于进程首次调入时，由程序员指出应该先调入哪些页。

请求调页策略发生在进程提出请求调入页面时，只调入发生缺页时所需的页面。这种调入策略实现简单，在目前的虚拟存储器中大多采用此策略。但容易产生较多的缺页中断，增加外存访问次数和磁盘访问的启动频率，另外调页时花费的开销较大，容易产生“抖动”（thrashing）现象。“抖动”概念及相关内容将在 7.3.2 节介绍。

(2) 从何处调入页面

每当发生缺页请求时，系统应将缺页调入内存，对于不同的系统，其所采用的方法各不

相同,大致可以分成三种。第一种,若系统拥有足够的对换区空间,则可全部从对换区调入所需页面,以提高调页速度;但在进程运行前须将与该进程有关的文件从文件区拷贝到对换区;第二种,如果系统缺少足够的对换区空间,对于不被修改的文件,则直接从文件区调入;对于可能被修改的文件,则从交换区调入,换出时也换出到交换区;第三种,在分时系统中,凡未运行过的页面都从文件区调入。对曾经运行过而又被换出的页面,则从交换区调入。但是某进程所请求的共享页面有可能已由其他进程调入内存,此时无须再从交换区调入。

虚存页面分配策略

虚存请求分页管理的页面分配策略比分页存储管理的页面分配策略复杂得多。在为进程分配页框时,必须考虑三个问题:第一,确定保证进程正常运行所需要的最少的页框数;第二,确定为每个进程分配的页框数目是否可变;第三,确定采用何种分配算法来进行页面分配。

(1) 最小页框数

最小页框数是指能保证进程正常运行所需的最少的页框数(物理块数)。若系统为某进程所分配的页框数少于此值时,进程将无法运行。进程应获得的最小页框数与计算机的硬件结构有关,取决于指令的格式、功能和寻址方式。对于采用单地址指令且是直接寻址方式,所需的最少页框数为 2,其中一块用于存放指令,另一块用于存放数据;若机器允许间接寻址,则至少需要 3 个页框。对于功能较强的机器,指令长度可能是两个或多于两个字节,指令本身及源地址和目标地址所涉及的区域都有可能跨两个页面,这时至少要为每个进程分配 4 个页框。

(2) 页面分配和置换策略

请求分页管理存在两种分配策略,固定分配和可变分配。置换策略是用于确定缺页中断发生且物理内存已满情况下,从内存换出哪些页面以腾出空间给新页面。也有两种策略,即全局置换和局部置换。若组合考虑,可以有三种适用的策略。

第一种,固定分配局部置换(即固定分区局部置换策略)。即根据进程的类型,为每个进程分配固定数目的内存页框,且整个运行期间不再改变。采用该策略时,如果进程运行过程中出现缺页,则只能从该进程的 4 个页面中选择一页换出,然后再调入一页。这种策略的难点在于,为每个进程分配的页框数难以确定。太少会造成频繁的缺页中断,降低系统吞吐量,太多则使内存中驻留的进程数减少,可能造成 CPU 空闲或其他资源空闲,在实现进程交换时会花费更多的时间。

固定分配策略分配页框时,可采用平均分配算法、按比例分配算法和优先权分配算法。平均分配算法是将系统中所有可供分配的页框,平均分配给各个进程。这种算法看似公平,实际不公平,可能会使某些进程产生很高的缺页率。按比例分配算法是根据进程的大小按比例分配页框。实际应用中,采用优先权分配算法,优先权高的进程分配较多的页框,从而可以照顾重要的紧迫的作业尽可能快地地完成。常见的做法是把内存中可供分配的所有页框分成两部分:一部分按比例地分配给各进程,另一部分则根据各进程的优先权,适当地增加其相应份额后,分配给各进程。

第二种,可变分配全局置换(即可变分区全局置换策略)。即先为系统中的每个进程分配一定数目的内存页框,操作系统自身也保持一个空闲页框队列;当某进程发生缺页中断时,由系统从空闲页框队列中取出一块分配给该进程;当空闲页框队列中的页框用

完时,操作系统从整个内存空间选择一页调出,该页可能是系统中任一进程的页面。这种策略实现容易,但可能使整个系统产生大部分时间都在进行页面投入换出工作中现象。

第三种,可变分配局部置换(又称按需调页或按需调页)。同样是根据进程的类型,先为每个进程分配一定数目的内存页框,但当进程缺页时,只允许从该进程的页面中选出一页换出,而不影响其他进程的运行。若进程缺页中断频繁,则系统须为该进程追加分配若干页框,直至进程的缺页率减低到适当程度为止;反之,若一个进程的缺页率特别低,则可在不引起其缺页率明显增加的情况下,适当减少分配给该进程的页框数。

3. 虚拟内存页面置换算法

页面置换算法(又称页面调度或页面调度)是指在需要调入页面且内存已满时,确定将要换出页面的算法。置换算法的好坏将直接影响系统的性能,不适当的算法可能会导致进程发生“抖动”。即刚被换出的页很快又被访问,需重新调入,为此,又需要再选一页调出,而此刚被换出的页,很快又要被访问,因而又需将它调入,如此频繁地更换页面,以致一个进程在运行中,将把大部分时间花费在完成页面置换的工作上。置换算法的出发点是将未来不再使用的或短期较少使用的页面换出。但由于对未来的情况不确定,所以一般采用局部性原理进行预测。常用的算法有以下几种:

(1) 最佳置换算法(又称最佳置换)

即选择不再使用的或者在最长时间内不再被访问的页面进行置换。这是一种理想化的算法,具有最好的性能,但实际上却难于实现,只能作为其他置换算法性能评价的依据。

(2) 先进先出置换算法(又称先进先出或先进先出)

选择最先进入内存的页面进行置换,即选择在内存中驻留时间最久的页面进行淘汰。该算法是最直观的算法,实现简单,可通过链表按时间先后顺序链接。但缺点是它与进程实际运行的规律不相适应,性能较差,将可能出现内存抖动现象,即当分配页面数增加,缺页率和缺页次数反而提高的异常现象,故实际应用极少。

(3) 最近最久未使用置换算法(又称最近最久未使用或最近最久未使用)

即选择最近一段时间内最长时间没有被访问过的页面进行置换。这种算法认为,过去一段时间内不曾被访问过的页面,在最近的将来也可能不再会被访问。该算法的性能接近于最佳置换算法。但算法中必须记录页面使用时间情况,所以算法赋予每个页面一个访问字段,用来记录页面自上次被访问以来所经历的时间,实际使用中常采用移位寄存器或栈来帮助实现。如果采用移位寄存器,进程访问页面时,该页面上的移位寄存器左边最高位置 1 定期右移并在最高位补 0 寄存器值最小的页面先淘汰;若采用栈来保存当前使用的各个页面的页号,则被访问的页面从栈顶往下移动,栈底的页面就是下一个将要淘汰的页面。但由此带来了系统开销和硬件开销增大的缺点。

(4) 时钟置换算法(又称时钟)

又称最近未使用算法(又称最近未使用或最近未使用),是最近最久未使用和先进先出的折衷算法。将内存中的所有页面都通过指针链接成循环队列,每页设置一位访问位,若某页被访问时,其访问位被置 1 置换时首先检查访问位,如果是 0 就选择该页换出;若为 1 则重新将它复 0 暂不换出而给该页第二次驻留内存的机会,再按照先进先出的算法检查下一个页面。当检查到队列中的最后一个页面时,若其访问位仍为 1 则再返回到队首再去检查第一个页面。由于该算法是循环地检查各页面的使用情况,故称为时钟置换算法。

于对另一些较少的页面进行访问。如果能够预知程序在某段时间间隔内要访问哪些页面,并能将它们提前调入内存,将会大大地降低缺页率,减少置换工作,提高 CPU 的利用率。

所谓工作集是指在某段时间间隔 Δ 里,进程实际要访问的页面的集合。Belady 认为,为使程序能有效地运行,较少地产生缺页,就必须使程序的工作集全部在内存中。然而,由于无法预知程序在不同的时刻将访问哪些页面,因此仍然只能利用程序过去某段时间内的行为,作为程序在将来某段时间内行为的近似。具体地说,便是把某进程在时间 Δ 的工作集记为 $W(\Delta)$,把变量 Δ 称为工作集“窗口尺寸” (working set window size)。

试验表明,任何程序遵循局部性原理装入内存时,都有一个临界值要求。当内存分配小于临界值,缺页率(即内外存之间的交换率)将会急剧增加;而当内存分配大于这个临界值时,再增加内存分配也不能显著减少缺页率,如图 9-10 所示。这个内存要求的临界值就是工作集的“窗口尺寸”。

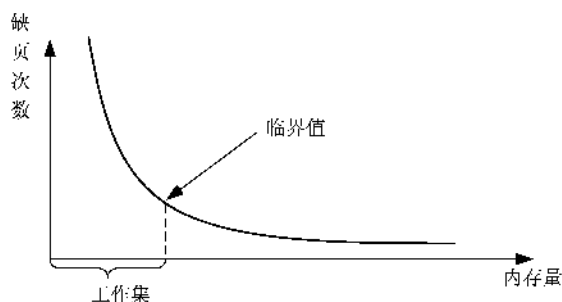


图 9-10 工作集和缺页率的关系

采用工作集模型非常简单,它能在保证尽可能高的多道程序道数的同时防止抖动,并能提高 CPU 的利用率。其困难之处在于工作集窗口大小的确定。由于工作集 $W(\Delta)$ 是二元函数,与时间 Δ 有关,也与窗口尺寸 Δ 有关。正确选择工作集窗口 Δ 大小,对存储器的有效利用和系统吞吐量的提高都将产生重要影响,若 Δ 选择过大,进程不易产生缺页,但存储器也不会得到充分利用;另一方面,若 Δ 选择过小,可能使进程在运行过程中频繁产生缺页中断,降低系统吞吐量。因此必须选择适中。

抖动现象 (thrashing)

抖动现象是虚存请求分页管理机制下的产物。在系统监控 CPU 的使用过程中,一旦调度程序发现 CPU 的利用率降低,它立即提高多道程序的度,即引入新的进程参加运行。当新进程进入内存时,客观上又只能从其他运行进程处获得物理块,于是将进一步加剧进程的缺页情况,使等待页面调入换出的进程数目也随之增多;这又使 CPU 的利用率进一步下降。调度程序又引入新的进程,产生恶性循环,使进程缺页率急剧上升,有效访问存储器的时间也急剧增加。换言之,运行进程的大部分时间都用于进行页面的换入换出,而几乎不能完成任何有效的工作。我们称这时的进程处于“抖动”状态。

图 9-11 示出了 CPU 的利用率与多道程序度之间的关系。在开始阶段,随着程序度的提高,CPU 的利用率也随之提高,并在随后达到某一峰值。此后如果继续增加多道程序度,将产生抖动,从而导致 CPU 的利用率急剧下降。

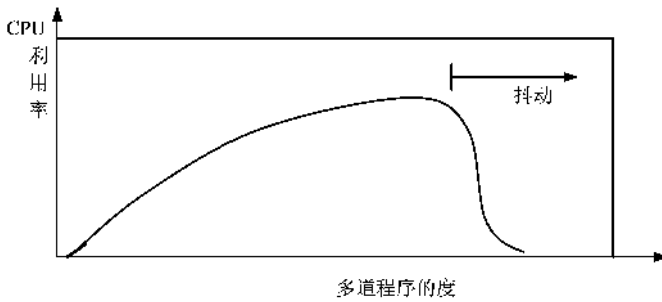


图 7.1.1 多道程序的度和 系统利用率间的关系

为保证系统有较大的吞吐量，必须防止抖动的发生。目前，常用的方法有以下几种：

（1）采取局部置换策略

系统采取可变分配局部置换策略，当发现缺页后，仅在进程本身的内存空间范围内置换页面，不允许从其他进程获得新的物理块。这种方法可以把抖动局限于较小的范围内。但并不很好，它不能从根本上防止抖动的发生。且当某进程发生抖动后，它会长期处于磁盘 I/O 的等待队列中，这样会使其他进程缺页中断的处理时间增长，从而延长了等待的访问时间。

（2）在 系统调度程序中引入工作集算法

这是利用局部性原理和工作集算法，在调度程序发现 系统利用率低时，先检查每个进程的驻留集是否足够大。只有每个进程在内存中都有足够大的驻留集时，才能再从外存上调入新的作业。这样，才不致因新作业的调入而导致缺页率增加。

（3） 系统准则

系统代表的是产生缺页的平均时间，杂是指系统处理进程缺页的平均时间。这是 系统最早于 1969 年提出的，其基本思想就是利用产生缺页的平均时间（系统等于系统处理进程缺页的平均时间）杂的原则调整多道程序的度。理论和实践都表明，此时的 系统利用得最好。

（4）挂起若干进程

当多道程序度偏高时，为了防止发生“抖动”，可用的一个简单易行的办法是挂起一些进程，以便腾出内存空间来分配给抖动的进程。被挂起的进程通常是选择优先权最低或较低的。当内存非常拥挤时，也可以选择一个并不很重要的、但却较大的进程挂起，以便能一次释放出较大的内存空间；或者是将具有最多剩余执行时间的进程挂起。

7.1.2 请求分页管理的共享与保护

在请求分页管理中实现共享与保护和分页存储管理中实现类似。但是，请求分页管理中还需在页表中增加有效 标志位，来保证该页是否为合法页面，假如页表中某一表项设置为“无效”，则表明该页不是进程的逻辑地址空间。

总而言之，请求分页管理具有如下优点：

（1）可有效解决碎片问题。

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

但其主要缺点是：

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

② 增加了系统开销，例如缺页中断处理机制。

③ 请求调页的算法如选择不当，有可能产生抖动现象。

④ 虽然消除了碎片，但每个作业或进程的最后一页内总有一部分空间得不到利用，如果页面较大，则页内碎片仍然较大。

7.2.2 请求分段存储管理

请求分段存储管理（request segmentation）是在分段存储管理的基础上建立的，其基本思想和分段存储管理类似，只是在请求分段管理中，程序运行前只需先装入若干段，便可启动运行，在执行过程中，进程可请求操作系统将所缺的段调入内存。同样，请求分段存储管理方式需要有硬件支持。

7.2.2.1 请求分段表机制

请求分段式管理中所设置的主要数据结构仍是段表，但因为作业只有一部分段装入内存，其余仍在外存上，故需在段表中增加若干项，以供程序在调进、调出时参考，如图 7-2 所示，在基本段表项段名（号）、段长和段起始地址三个属性的基础上，还增加了以下各项：

① 存取方式。用于标识本分段的存取属性是只执行、只读，还是允许读写。

② 访问字段。其含义与请求分页的相应字段相同，用于记录该段被访问的频繁程度。

③ 修改位。用于表示该段进入内存后，是否已被修改过。

④ 存在位。指示本段是否已换入内存。

⑤ 增补位。用于表示本段在运行过程中是否进行过动态增长。

⑥ 外存始址。指示本段在外存中的起始地址，即起始盘块号。

段名	段长	段基址	存取方式	访问字段	修改字段	存在位	增补位	外存始址
----	----	-----	------	------	------	-----	-----	------

图 7-2 请求分段中的段表项

另外，为实现请求分段管理，操作系统除了要进行地址映射外，还需跟踪物理内存的使用情况，以便在装入新段时合理分配内存空间，所以系统还设置了系统段表和空闲段表。系统段表用以描述系统内所有的占用段，而空闲段表描述内存中所有的空闲段，也可以结合到系统段表中进行管理。

7.2.2.2 地址转换和硬件支持

7.2.2.2.1 缺段中断机制

请求分段系统采用请求调段策略，操作系统根据段表表项进行地址变换或产生缺段

中断。若进程访问的段尚未调入内存，缺段中断机制产生缺段中断信号，中断处理程序根据信号调入所需段。和缺页中断机制类似，在一条指令执行期间可能产生多次中断。和缺页中断机制不同的是，指令和操作数必定不会跨越段边界上。但一般会有存取权限的违例中断和段的越界中断。缺段中断的处理过程如下：

(员 获得缺段中断信号，阻塞请求进程；

(圆 判断内存中是否存在合适大小的空闲区，若有，则转向第四步；

(猿 判断空闲区空间总和是否能满足要求，若能，则紧凑，形成合适大小的空闲区，若不能，则淘汰一个或多个段，形成合适大小的空闲区；

(源 从交换区读入所需段的内容，修改段表及内存空闲区链表；

(缘 唤醒请求进程，让进程获取权限，继续执行，缺段中断处理完毕。

圆 地址变换机制

由于进程所访问的段并不一定全在内存，所以，虚存请求分段系统的地址变换机制要比实存分段存储管理的地址变换机制复杂一些。当发现所访问的段不在内存时，操作系统必须先将所缺的段调入内存，修改段表，再利用段表进行地址变换，所以，地址变换机制中还包含了缺段中断请求及处理等功能。具体过程如下：

(员 获得访问要求，即逻辑地址 (杂 憎) ；

(圆 判断段内偏移量 憎是否小于段长，若不小于，则进行分段越界处理，否则继续；

(猿 判断该请求是否符合存取方式，若不符合，则进行分段保护处理，否则继续；

(源 判断段 杂是否已经调入主存，若没有，则进行缺段中断处理，否则继续；

(缘 修改访问字段，根据存取方式设置修改位的值；

(远 形成访问的物理地址，结束地址变换过程。

第 4 章 存储管理

存储管理策略对进程调度具有很深的影响。存储管理子系统需要决定内存中驻留的进程，管理进程的虚地址空间中的内容，监视可用的存储空间量，并实现进程在内存和交换区之间的对换，以提供更多的内存空间。早期的 哉 系统采用对换技术在内存和对换设备之间传送整个进程，月 系统版本是第一个主要实现请求调页策略的系统，它在内存和二级存储设备之间换入 换出存储页，而不是整个进程。哉 系统中采用交换和请求调页两种策略来完成存储管理。这样，当进程运行时，并不需要整个进程的映像都在内存，而只需在内存保留当前用到的页面。当进程访问某些尚未调入内存的页面时，内核按需要装入该页，这种策略就是请求调页策略，其优点是使进程的虚地址空间到物理地址空间的映射具有更大的灵活性。允许进程的大小比可用的物理存储空间大，还允许更多的进程同时在内存执行。采用对换策略的优点是实现较为简单，系统开销较小。

第 4.1 节 交换

交换策略是指进程在内外存之间进行换入换出的具体方法和交换规则。对磁盘交换区的使用方法关系着进程存储管理的效率。早期的 哉 系统是在内存和交换区之间传送整个进程，这样交换过程简单，但比较费时和影响系统的执行效率。现代 哉 系统采用部分交换

方式，也就是说，操作系统规定一些关键的进程描述信息常驻内存，不参与交换，而只部分交换进程内容，从而提高了系统的执行效率。

交换空间的分配

交换空间是磁盘上的一块连续区域，该连续区域独立于磁盘的管理，在操作系统中单独作为一个块设备进行管理，称为交换设备。核心分配交换空间是以一组连续的块为单位进行。交换设备上的空间分配是临时性的，取决于进程调度模型。驻留在交换设备上的进程最终要调入内存运行，释放所占用的交换区，因此速度是至关重要的问题。系统一次进行多块的操作要比多个单块操作快得多，所以交换区的分配是连续空间，并不考虑碎片问题。

内核通过驻留在内存称为“映射图”的表来管理交换设备的空间。映射图是一个数组，映射图数组中的每一表项包含两个属性，可分配交换空闲区的起始地址及该地址上可用资源的单位数。内核根据映射图的类型解释地址和单位的含义。它也用于管理除交换设备以外的其他一些资源，一般采用最先适应的算法进行分配，如图 7-1 所示为一个从地址 0 开始，具有可用资源数为 1024 的初始交换映射图。

随着进程的调度和交换，磁盘资源会不断地被分配和回收，而相对的数组信息也会不断地被修改，映射图也随之不断被改变，使之始终保持空闲资源的准确信息。例如，经过一段时间之后，某一时刻的映射图如图 7-2 所示。

地址	单位数
0	1024

图 7-1 初始交换映射图

地址	单位数
0	1024
1024	512
1536	512

图 7-2 某一时刻的交换映射图

这时映射图中的信息表明从地址 0 开始有 1024 块可用磁盘块，从地址 1024 开始，有 512 块可用磁盘块，从地址 1536 开始有 512 块可用磁盘块。系统根据映射图来分配内存空间的操作由最先适应算法完成，内核从映射图中查找第一个含有足够大的能满足要求的表项。若这一需求占用了该表项的全部资源，内核将此表项从数组中删除，并压缩映射图；否则，内核根据分配资源的数量，修改该表项中的地址和数量域。如在图 7-2 基础上，若系统要给交换的进程分配一个等于 512 个磁盘块的空间时，可将第 1 项分配给它，同时删除此表项；若要求的空间大于 512 个盘块，则从表的第二项开始分配，修改表项的地址为剩余空闲区的起始地址，并将单位数改为剩余空闲区的数量。

操作系统中设置了多个交换设备，内核循环选用交换设备，以保证交换设备上有足够大的连续存储空间。管理员也可以动态创建和取消交换设备。如果某个交换设备正在被取消，内核就不再往该交换设备上交换数据。其上的数据被不断取走，直到该交换设备被腾空而被取消为止。

进程的换出

如果内核需要内存空间，可把进程换出到交换设备中。引起进程换出的事件包括四个：
(1) 系统调用 fork 为子进程分配空间；

- (四) 系统调用 调用扩大一个进程的大小；
- (五) 进程由于其栈的自然增长而变大；
- (六) 为了运行以前被换出、而现在又应被换入的进程。

当核心决定一个进程应从内存换出时，首先，确定该进程中将要被换出的页面；然后，内核在交换设备上为该进程分配所需的交换空间，将该进程锁在内存，以防止在换出过程中再选它作为换出对象，内存将交换空间的首地址保存在该进程的区表表项中；接着，内核直接在交换设备和用户地址空间交换数据，并在一次 换出的过程中尽可能多地交换数据，如果硬件不能在一次操作中传送多个页面，则内核软件要反复传送内存页面。数据传送的精确速率及传送机制主要取决于磁盘控制器的能力和存储管理的实现。若采用页式结构，内核必须将需要换出的不连续的页面进行汇集，磁盘驱动程序要用这些页面地址启动 换出的操作。

交换工作仅需局部进行，也就是说，核心不必把整个进程的虚拟地址空间写到交换设备上，相反，它只需将进程占用的全部内存空间复制到所分配的交换空间中，而不管未分配内存的那部分虚地址。当内核换入该进程时，由于知道该进程的虚地址映射，因此能将该进程重新分配到正确的虚地址上，而不需要从数据缓冲区中将进程的地址空间重新映射到内存中。

一般来讲，在进行换出工作时，僵死进程不必换出，锁在内存的进程不能换出，因为僵死进程不占物理存储空间；内核选择正在睡眠的进程来换出，选择哪一个取决于进程的优先权和它在内存驻留的时间；如果没有正在睡眠的进程，则选择“就绪”进程，这取决于进程中用户自定义的优先权值及进程在内存中驻留的时间。

进程的换入

操作系统中的换入操作由 园号进程完成，有的系统将 园号进程命名为调度程序 或是交换程序，它负责进程调度和交换工作。操作系统初始化结束后，园号进程进入无限循环状态，它不断从交换区中换入进程，当内存不够时再将进程换出到交换区。如果没有进程换入 换出，园号进程进入睡眠状态，系统通过时钟中断定时唤醒。系统中使用的具体交换策略会因操作系统的不同而不同，交换策略中采用的算法也会因此不同。

交换程序被唤醒后，它就进行换入操作。首先，它查找所有处于“就绪且换出”状态的进程，从中选取换出时间最长的进程；接着判断是否有空闲的内存空间，如果有足够的内存，则分配内存，将进程从交换设备读入，释放交换空间；如果成功完成，继续重复执行上述步骤。直到交换设备上没有“就绪”进程或内存空间不足。若是前者，则交换进程转入睡眠状态；若是后者，则试图换出另一个进程，重新启动换入进程，继续重复上述操作。

请求调页管理

操作系统中采用的另一种策略是请求调页存储管理方式。请求调页式存储管理是指将内存空间和外存空间都划分成固定的块（页面或页），进程运行时系统只在内存保留当前活动进程的某些页面，而不是将整个进程的映像都放在内存中。当进程访问某些不在内存的页面时，发出缺页中断，内核程序再将所缺的页面调入内存。这就形成了外存交换区与内存空间之间以存储页面为单位的动态页面调入方式，页式存储管理是请求调页管理的基础，请求调页虚存管理模式具有进程大小不受物理内存空间限制、内存管理比较合理等特点。

根据程序执行的局部性原理可知，每次进程运行所需要的代码通常只是进程正文段中的

一小部分指令,而程序执行中所访问的数据也可能是整个数据集中的一小部分内容,因此为了提高进程的执行效率,操作系统核心程序为系统中的活动进程提供工作集管理方式。工作集中存放着内存中最近被访问过的页面集合。进程执行时,首先在工作集中查找相关页面,如果没有找到所需页面,则发出中断请求。每次缺页中断处理完成时,核心程序同时对工作集内容进行一次更新,并从交换设备读入所缺页面。工作集的大小随进程执行而变化,这取决于进程对做存储器访问的情况,较大的工作集窗口意味着进程不会经常出现缺页现象。但实现纯工作集模型是不现实的,所以系统采用设置访问位的方法进行粗略模拟,进程访问一内存页时,该页的访问位置 1。内核定期抽查内存的访问情况,如果某页最近被访问,则内核认为该页是工作集的一部分,否则,该页在内存中的“年龄”不断增长,直到它被换出。

当进程存取一个不属于工作集中的页面时,产生有效性页面错(或称有效性错),则内核挂起该进程,换入所需页面,然后进程又重新执行。

9.2.2 页表数据结构

操作系统核心提供四个主要的数据结构用来支持低级的存储管理和请求调用,包括页表、磁盘块描述字(或称页表项)、页框数据表(或称页框表)和交换使用表(或称页交换表)。

(1) 页表

操作系统核心把一个进程的整个逻辑地址空间分成若干段,称为区(或称段),如进程的正文段、数据段等,并作为进程中可被共享和保护独立实体,在操作系统中将每个区对应一个页表,用来存取物理存储器。页表包含有以下的字段:

- ① 页地址:表示该页的内存地址,采用物理页号进行描述。
- ② 保护位(或称页保护位):用于设置进程对该页的读、写或执行权限。
- ③ 有效位(或称页有效位):表示该页内容是否有效。当有效位为 1 时,表示该页在内存。
- ④ 访问位(或称页访问位):表示该页最近是否被访问过。
- ⑤ 修改位(或称页修改位):表示该页内容是否被修改。系统根据修改位来确定该页内容是否需要写回磁盘。
- ⑥ 写时拷贝位(或称页写时拷贝位):指示该页内容被修改时,内核是否为该页建立一个新的拷贝,一般用于系统调用。
- ⑦ 年龄位(或称页年龄位):表示该页作为进程工作集中的成员之一存在多长时间了。

(2) 磁盘块描述字

每一个页表项都与一个磁盘块描述字相关联。磁盘块描述字主要用于对虚拟页的磁盘副本进行说明。它包括交换设备号、块号和类型三个字段。共享一个区的所有进程共享页表项和磁盘块描述字。虚拟页的内容或者在一个交换设备的特定块中,或在一个可执行文件中(即不在交换设备上)。当一虚拟页存在交换设备上时,磁盘块描述字描述该页的逻辑设备名和块号,类型设置为对换;如果该页在可执行文件中,则磁盘块描述字含有该文件中的逻辑块号,则内核可以很快将逻辑块号映射到磁盘地址空间,此时类型属性中设置为文件。

磁盘块描述字还为系统调用设置了两个特殊条件,一个是页“请求填入”,另一个是页“请求清零”。当进程调用系统调用时,内核从文件系统中将可执行文件读入内存,由于采用请求调页管理模式,内核并不为可执行文件预先分配内存,而是首先为可执行文件分配页表项和磁盘块描述字,并将页表项设置为“请求填入”或“请求清零”。在用算法读取文件时,

进程每一页都产生有效性错,然后出错处理程序查看该页是“请求填入”还是“请求清零”。若是“请求填入”,则表示该页的内容将立即被可执行文件的内容覆盖,所以不用清除;若是“请求清零”,则表示该页的内容应该被清掉。

④ 页框数据表

页框数据表对整个内存中的每个页框进行描述,它是通过页号进行索引。每项包含有:

① 页面状态:说明该页是在交换设备上,还是在可执行文件中;同时还指示该页当前是否为该页服务(从交换设备上读数据);另外还指示该页是否可被重新分配。

② 引用计数:表示访问该页面的进程个数,同时也表示当前引用该页的有效页表项的数目,它可能不同于共享含有该页的区的进程数目,如在共享算法中。

③ 交换设备号:为交换或文件系统中的逻辑设备号。

④ 块号:含有该页一个拷贝的块号。

⑤ 指向空闲链表中其他页表项的指针和指向页散列队列中其他页表项的指针。

操作系统内核将页表中的所有表项链接到空闲页链表和散列链表中。空闲页链表是页的缓冲池,其中的页面可被重新分配,内核按最近最少使用的顺序从空闲链表中分配页面;散列链表存储的是已分配页面,在给定设备号和块号的前提下,可以快速定位对应的页面。当系统为某一进程分配页面时,内核从空闲链表的表头摘下一个或多个空表项,修改其交换设备号和块号,并将它放入到恰当的散列队列中。

⑤ 交换使用表

每个交换设备上的页面在交换使用表中都占有一项,该表项含有一个引用数,表示当前有多少页表项指向交换设备上的该页。

图 9-10 为页表项、磁盘块描述字、页框数据表和交换使用表等请求调页中各个数据结构之间的关系。一个进程的虚地址映射到一个指向内存页面为页表的页表项;该页表项的磁盘块描述字指示该页的一个副本存放在交换设备上的某块中,且它的内存引用数为 1 对应的交换使用表数为 1 这也说明当前只有一个页表项指向该交换的副本。

⑥ 地址变换机构

在操作系统的地址转换管理中,系统为每个进程设立进程区表(用于记录进程的逻辑地址空间到物理地址空间的映射),用于完成进程从逻辑地址空间到物理地址空间的映射,本进程区表描述了该进程所拥有的区。整个系统设置一张区表,用于管理所有进程的区,包含有以下字段:

① 类别:表示该区是正文段、数据段还是其他区;

② 该区在进程中的起始虚地址;

③ 该区的页表地址;

④ 区的大小,即页表的页数;

⑤ 保护域,指示对应进程对该区的存取权限;

⑥ 引用数,指示该区被多少进程共享;

通过第三项描述信息“页表地址”,各区对应一个页表,页表中指出了该区各页在内存的起始地址信息,从而构成了地址转换机构的基础信息,其地址转换结构如图 9-11 所示。

⑦ 偷页进程

偷页进程是一个核心进程,它的作用是将不再是进程工作集中的页面换出内存,又称为页面淘汰进程或页面窃取进程。它是系统初启时由内核创建的,并一

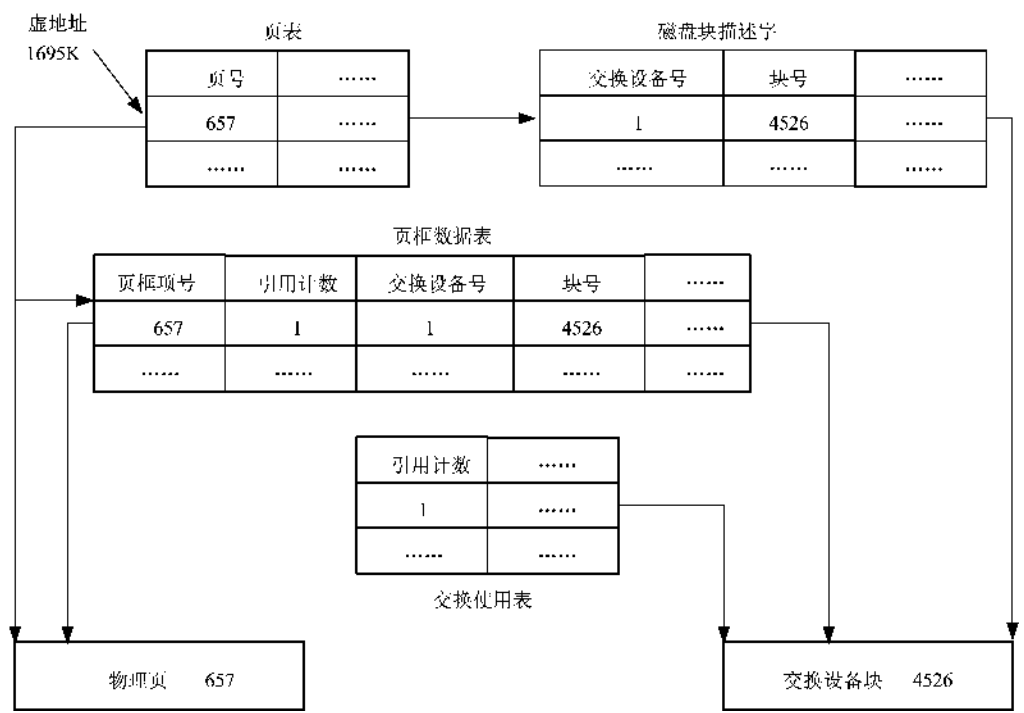


图 7-1 请求调页方式中各数据结构间的关系

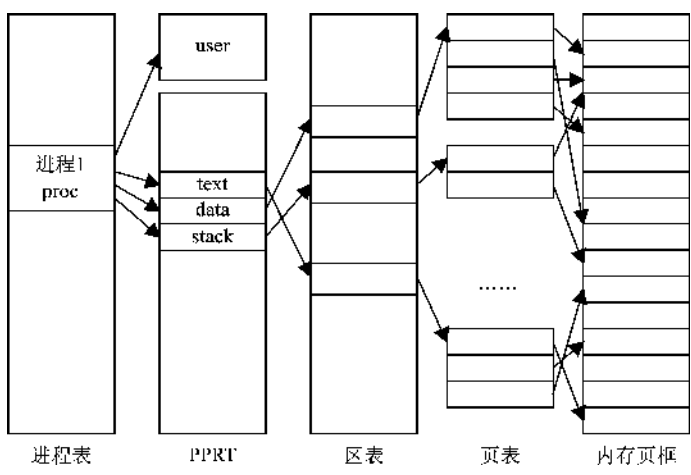


图 7-2 请求调页方式中地址转换结构图

直存在于系统的整个活动期，每当空闲页很少时就被调度工作。被调度工作时，首先遍历整个区表，检查所有活动的、非上锁的区，增加所有有效页的年龄值。当进程在一个区中出现有效性错时，内核封锁含有该页的区，以便页面淘汰时不能取走该页。

内存页面有两种状态：一种是页年龄增大，但还不能交换；另一种是该页可被交换，且

该页所占据的页框可被重新分配给其他虚拟页。第一种状态表示进程最近存取过该页，所以该页是进程工作集中的成员，当前该页还被进程使用；访问该页时需置访问位，可由硬件完成，也可由软件实现。偷页进程复位这类页面的访问位，且记住从上次该页被访问以来已检查了多少遍。当遍数超过了某个阈值，内核就将该页置为第二种状态。

如果有两个或多个进程共享一个区，这些进程更新一组页表项的访问位，这对偷页进程没有影响。若某一页面是任意进程工作集的一部分，它就驻留在内存；若它不在任何进程的工作集中，则可换出。

系统中可用空闲内存页面低于下限值时，内核唤醒偷页进程，让它换出一些页面，直至空闲内存页面数超过上限值。这种方式可以减少抖动发生的概率，系统管理员也可以通过设置上限值和下限值以获得最好的系统性能。

当偷页进程决定换出某页时，需考虑交换设备上是否已有该页拷贝。如果交换设备上没有该页的拷贝，则换出该页，将它放到要被换出的页面链表上，继续运行，直到要换出的页面链表达达到一定数量，内核一次性写入交换设备；如果该页的拷贝已存在，且没有进程修改它在内存中的内容，则将该页表项的有效位置 园 页框数据表表项的引用数减 员 该表项放入空闲链表以供进一步分配；如果页面拷贝已存，但进程修改了该页的内容，则先将修改页放到被换出的页面链表，在全部写入交换设备后，释放它在交换设备上占据的空间。

一般来讲，请求调页管理方式中，页换出内存有两个阶段。第一阶段，偷页进程找出应换出的页面，将页号放到要被换出的页面链表中；第二阶段，内核在适当的时候将页面拷贝到交换设备上，清除页表表项中的有效位，页框数据表的引用数减 员 如果引用数为 园 则将页框数据表项放在空闲链表的尾部，直到该页面被重新分配前，该物理页的内容仍是有效的。

源 缺页处理

缺页又称为页面故障(字错误或页面错。系统产生两类页面错：有效性错和保护错。当出现页面错时，缺页处理程序可能从磁盘上的某页读入内存，并在 隙的执行期间睡眠，这和一般的中断处理程序不同。

(员 有效性错处理程序

对于在进程虚拟地址空间之外的页面，及虽然在虚拟地址空间内但当前未在内存的页面，其有效位为 园 表示缺页。如果一个进程企图存取一个有效位为零的页面，将产生有效性错，内核调用处理程序，进行相应的处理。其具体工作过程是：

① 首先，内核向被中断进程发出“段越界”软中断信号。

② 判断发生有效性错时进程的状态，可以分为五种状态，第一种是在交换设备上，但不在内存中，第二种是在内存的空闲页面链表上，第三种是在一个可执行文件中，第四种是标志为“请求填入”，第五种是“请求清零”。

③ 如果是第一种状态，则从磁盘块描述项中找到存放该页面的交换设备号和块号，修改页表项，从相应的交换设备上读入该页面。

④ 如果是第二种状态，则处理程序从页缓冲区中找出该页，删除磁盘块描述项中的块号，重置页表表项，指向刚找到的页表，将引用数加 员；并在需要时，将该页从空闲链表中取出。如果另一个进程也在同一页发生有效性错，但还没完全将该页读入内存，则处理程序不必再将其读入内存，直接睡眠，直到另一个处理程序完成。

⑤ 如果是第三种状态,则内核从原始文件中读入该页。

⑥ 如果是第四或第五种状态,则内核在内存分配空闲页面并修改相应的页表项。

⑦ 最后,处理程序清除修改位,设置有效位,结束处理,重新计算进程的优先权,返回用户态,检查是否收到新的任何软中断信号。

（四）保护错处理程序

这是由进程引起的错误,是进程对有效页面的存取权限不符合规定而造成的。一般来讲,硬件会为保护错处理程序提供发生错误的虚拟地址,然后由处理程序处理。如果处理程序判断错误是由于页面设置了“写时拷贝”标志而引起的,且该页还被其他进程共享时,内核分配一个新页并将老页的内容拷贝到新页上,其他进程仍然保持引用老的页面;如果设置了“写时拷贝”位,但没有被共享,则内核允许进程重新使用该页,清除“写时拷贝”位,解除该页与其他磁盘拷贝的联系,然后,将该页的页框数据表项从页面队列移出,该页的交换使用数减 1。当交换使用数等于 0 时,释放交换空间。

保护错处理程序结束执行时,要设置修改位和保护位,清除“写时拷贝”位,重新计算进程优先权,并检查是否有软中断信号。

以上两种存储管理方式各自有自己的优缺点。简单地说,请求调页式管理采用虚拟地址空间技术,进程的地址空间可以远远大于实际的物理地址空间,允许较多的进程同时装入内存,使系统的内存管理能力增大。但请求调页管理中,逻辑地址到物理地址的转换比较复杂,这种复杂程度有可能会影响到系统性能。交换式管理的优点是实现过程简单,系统开销小,但进程的频繁交换会影响系统的执行效率。

综上所述,操作系统采用以上两种存储管理策略是适合系统进程特点的,两种策略在应用中可以做到优势互补,使进程的管理简单便捷,使系统的整体效率得到提高。

本章小结

虚拟存储技术是为了提高多道程序并发执行的程度、增加 CPU 利用率而提出的,它允许把大的逻辑地址空间映射到较小的物理内存上,虚拟存储器具有新的特性,包括离散性、多次性、对换性和虚拟性等。

虚拟内存的管理有两种基本方法,即请求分页存储管理和请求分段存储管理模式。它们是在分页管理和分段管理基础上,增加了缺页中断和缺段中断而形成的。请求分页存储管理根据实际程序执行的顺序,动态申请存储区。内存中只存放进程当前运行所必需的页面,通过缺页中断将所缺的页面从外存读入内存。当内存中没有剩余的页框存放读入的页面时,可通过页面置换算法将一些内存页面换出到外存,页面置换算法包括有最佳置换算法、先进先出算法、最短驻留时间算法等,这些算法的实现必须要有硬件的支持,并会产生一定的系统开销。由于页面在内外存之间频繁换入换出,将带来抖动,解决抖动的方法有四种,分别是局部置换策略、工作集算法、虚拟内存准则和挂起若干进程。各种解决方法有各自的优缺点。请求分段存储管理是分段管理技术加上虚拟存储技术两者的结合,它以段为基本单位实现动态存储管理过程,同请求分页存储管理一样,它需要有缺段中断管理机制和重定位机制等硬件的支持。

操作系统中的存储管理采用了交换和请求调页两种存储管理技术。为实现交换和请求调页,系统设置了更多的数据结构,以实现各页面的共享和保护。

习题

1. 虚存的概念是什么？有哪些特征？

2. 请求分页存储管理的基本思想是什么？实存分页管理中的页表和虚存请求分页管理的有什么不同？请求分页管理中的页表中各项的作用是什么？

3. 请求分页存储管理需要哪些硬件支持？

4. 请求分页管理中常采用哪几种页面置换算法？有哪些分配方法和置换方式？

5. 某虚存的用户空间共有 100 个页面，每页 4096 字节，主存 10240 字节。假定某时刻系统为用户的第 0 页、10 页分别分配的物理块号为 20、100。试将虚拟地址 0000H 和 000AH 变换为物理地址。

6. 在一个请求分页系统中，采用 LRU 的页面置换算法，假如一个作业的页面走向为 1、2、3、4、5、6、7、8、9、10、11、12、13、14、15、16、17、18、19、20。当分配给该作业的页框数分别为 3 和 4 时，试计算访问过程中所发生的缺页次数和缺页率。

7. 什么是有效访问时间？试描述有效访问时间和缺页率之间的关系。

8. 什么是工作集？如何利用？

9. 什么是抖动？产生抖动的原因是什么？如何减少抖动？

10. 试说明请求分段系统中的缺段中断处理过程。

11. 请求分页系统中采用什么数据结构对交换区进行管理？

12. 请求分页系统进程的换入、换出是如何实现的？

13. 请求分页系统中有哪些类型的缺页？各自如何处理？

14. 请求分页系统中请求调页管理中设计了哪些数据结构？各个数据结构包含哪些内容？各自的含义是什么？

第 1 章 设备管理

在计算机系统中，除 CPU 和内存之外，其他的大部分硬件设备都称为外部设备。外部设备包括常用的输入输出设备、外存设备以及终端设备等。这些设备种类繁多，特性各异，操作方式的差别也很大，从而使得操作系统的设备管理变得十分复杂。

1.1 设备管理

1.1.1 设备

计算机设备种类繁多，从操作系统的观点来看，其重要的性能指标有：数据传输率、数据的传送单位、设备共享属性等，从不同的角度出发，设备可分成不同的类型。因此可以从不同的角度对它们进行分类。

按设备中的数据传输单位分类

(1) 块设备

块设备把信息存储在固定长度块中，每块都有自己的地址。一般块的大小在 512 至 4096 个字节之间，基本特征为能独立读写单个块，典型的块设备为磁盘、磁带等。

(2) 字符设备

字符设备以字符为单位发送或接收一个字符流，不考虑任何块结构。典型的字符设备如行式打印机、鼠标等。

按资源分配的角度分类

(1) 独占设备

独占设备是指设备在用户作业的整个运行期间必须为此用户所独占，才能保证传送信息的完整性。对多个并发进程而言，应该互斥地访问该类设备。通常慢速设备属于独占设备。

(2) 共享设备

共享设备是指在一段时间可以被多个进程以共享方式使用的设备。实际上是几个进程交替对设备进行读写，而宏观上看是同时对它们进行访问。

(3) 虚拟设备

虚拟设备是指通过虚拟技术将一台独占设备改造成若干台逻辑设备，供若干个用户进程同时使用，通常把这种经过虚拟技术处理后的设备称为虚拟设备。

1.1.2 设备控制器

设备一般由机械和电子两部分组成，为了提供更加模块化、更加通用的设计，一般将这两部分分开。电子部分称作设备控制器，它处于 CPU 与设备之间，它接收来自 CPU 的命令，控制设备工作，使得 CPU 可以从繁杂的设备控制事务中解

脱出来。设备控制器是一个可编址设备，当它仅控制一个设备时，它只有一个设备地址；当它可连接多个设备时，则应具有多个设备地址，每一个地址对应一个设备。设备控制器应具有以下功能：

④ 接收和识别来自 悦载的各种命令。悦载向设备控制器发送的命令有多种，如读、写等，设备控制器应能够接收并识别这些命令。为此设备控制器中应设置控制寄存器存放接收的命令及参数，并对所接收的命令进行译码。

⑤ 实现 悦载与设备控制器、设备控制器与设备之间的数据交换。为了实现数据交换，应设置数据寄存器存放传输的数据。

⑥ 记录设备的状态供 悦载查询。应设置状态寄存器记录设备状态，用其中的一位来反映设备的某种状态，如忙状态、闲状态等。

⑦ 识别控制器的每个设备的地址。系统中的每一个设备都有一个设备地址，设备控制器应能够识别它所控制的每个设备地址，以正确地实现信息传输。

愿源瑶系统结构

计算机的 系统结构是主机和外部设备之间的数据传送系统，目前主要有总线型和通道型两种结构。

愿源瑶总线型结构

微型机 系统多采用总线 系统，如图 愿源瑶所示。

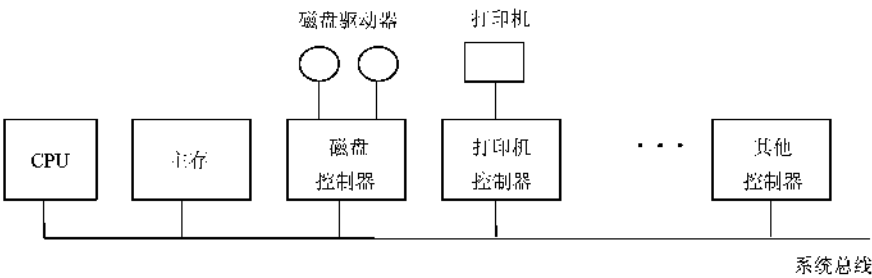


图 愿源瑶总线型 系统结构

从图 愿源瑶可以看出，悦载和存储器直接连到总线上，设备通过设备控制器连接到总线上。

愿源瑶通道型结构

通常，为主机配置的 设备较多时，如果所有设备的控制器都通过一条总线与 悦载通信，则会使总线和 悦载的负担过重。为此，在 系统中不采用单总线结构，而是增加了一级 通道，以实现对各设备控制器的控制，减轻 悦载的负担。

在具有通道计算机系统中，存储器、通道、控制器和设备之间采用四级连接，实施三级控制，如图 愿源瑶所示。

从图中可以看出，一个存储器可以连接若干个通道，一个通道可以连接若干个控制器，一个控制器可以连接若干个设备。悦载执行 指令对通道实施控制，通道执行通道命令对控制器实施控制，控制器发出设备控制信号对设备实施控制，设备执行相应的输入输出操作。

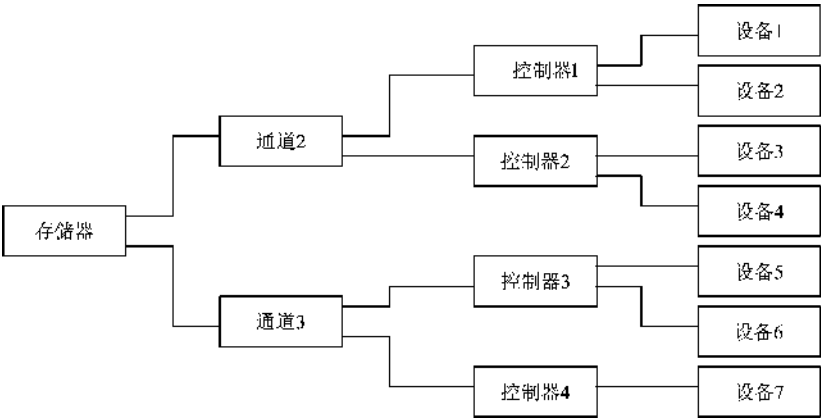


图 4-1 简单通道 I/O 系统

虽然各种 I/O 通道的基本功能是相同的，但其形式和规模却相差很大。根据信息交换的方式，可以将通道分为三种类型：

- (1) 字节多路通道，以字节作为信息传送的单位，往往服务于多台低速的 I/O 设备。
- (2) 选择通道，在同一时间里只能为一台设备服务，能连续地传送一批数据，主要用于连接高速的外部设备。
- (3) 成组多路通道，结合了字节多路通道分时操作和选择通道高速传送的优点，广泛应用于连接高速和中速的设备。

I/O 数据传送控制方式

设备管理的主要任务之一是控制设备和主机之间的数据传送，随着计算机技术的发展，数据传送控制方式也在不断地发展。本节介绍几种常用的方式，它们之间的主要不同在于 I/O 过程中处理机的干预程度。

I/O 程序直接控制方式

主机在执行程序过程中遇到一条与 I/O 相关的指令时，可以通过给相应的 I/O 控制器发送命令来执行这条指令。在程序直接控制 I/O 时，I/O 控制器执行相应的操作并设置 I/O 状态寄存器中相应位。I/O 部件并不主动地通知主机，也就是说，它并不中断主机当前的处理过程。因此，主机必须负责定期检查 I/O 部件的状态寄存器，直到发现 I/O 操作完成为止。在这种方式中，主机还负责从主存中读取需输出的数据，并将输入的数据存入主存。

由于在程序直接控制 I/O 方式中主机直接控制 I/O 操作的过程，包括测试设备状态、发送读/写命令和传送数据，因此，指令集中应包括以下几类 I/O 指令。

- (1) 控制类：用于激活外部设备，并告之做何种操作。例如，指示磁带到带或前移一个记录。
- (2) 状态类：用于测试与 I/O 部件及其外部设备相关的各种状态条件。
- (3) 传送类：用于在主机寄存器和外设之间传输数据。

图 10-1-1 给出了使用程序直接控制 I/O 的一个例子，从外部设备读取一块数据到内存，每次读一个字（4 字节）的数据。对读入的每个字，CPU 必须循环地执行状态检查，直到确定该字已经在 I/O 部件的数据寄存器中。然后，CPU 从 I/O 部件读入该字到 CPU 的寄存器中，并最终存入主存。

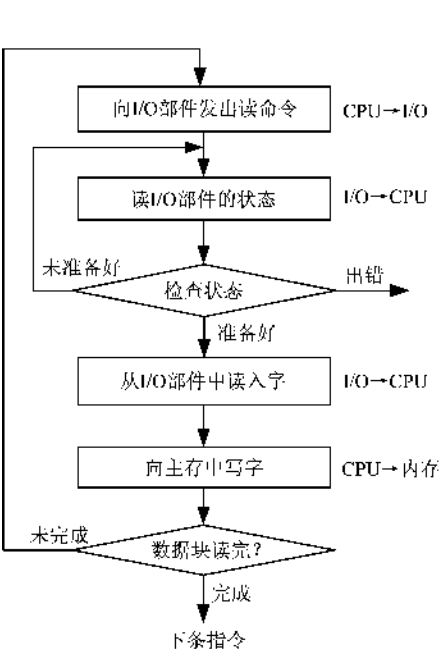


图 10-1-1 程序直接控制方式示例

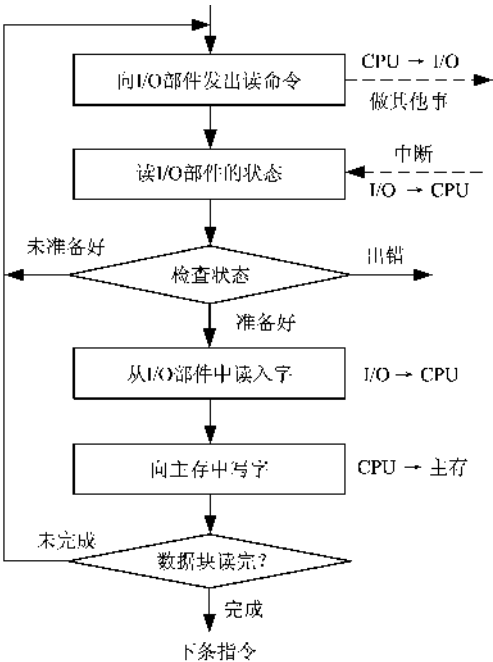


图 10-1-2 程序中断控制方式示例

程序中断控制方式

程序直接控制 I/O 的问题是 CPU 必须花费大量时间等待相应的 I/O 部件准备好接收或发送数据。CPU 在此等待期间，必须反复地测试 I/O 部件的状态，其结果是严重地降低了整个系统的性能。

解决办法是，CPU 向 I/O 部件发出命令后，继续做其他一些有用的工作。当 I/O 部件准备好与 CPU 交换数据时，I/O 部件中断 CPU，要求服务。CPU 和前面一样执行数据传送，然后恢复它以前的处理。

首先从 I/O 部件的角度考虑是如何工作的。对于输入，I/O 部件接收到 CPU 的读命令，然后开始从相关的外设读数据。一旦数据进入到 I/O 部件的数据寄存器，I/O 部件通过中断信号线向 CPU 发一个中断信号，表示 I/O 部件已准备好数据。然后，I/O 部件等待直到 CPU 请求数据。当 CPU 发出这个请求后，I/O 部件把数据放到数据总线中，然后准备下一次的 I/O 操作。

从 CPU 的角度来看，输入过程如下：CPU 发一个读命令，然后保存当前程序的上下文环境，转去执行其他程序。在每个指令周期的末尾，CPU 检查中断。当有来自 I/O 部件的中断

时, 系统保存当前运行程序的上下文, 转去执行中断处理程序处理该中断。这时, 系统从 I/O 部件读一个字的数据传送到 系统的寄存器, 并存入主存。接着, 系统恢复发出 I/O 命令的程序的上下文, 继续运行。

图 8-10 给出了一个采用中断驱动方式读入一个数据块的示例。与图 8-9 相比可看出, 中断驱动 I/O 方式比程序直接控制 I/O 方式效率更高, 因为 系统不必进行无用的等待。

然而, 中断驱动 I/O 方式仍然消耗了大量的 系统时间, 因为每次将一个字的数据从 I/O 部件传送到内存或从内存传送到 I/O 部件, 都必须经过 系统。

8.3.2 通道控制方式

尽管中断驱动 I/O 方式比程序直接控制 I/O 更有效, 但在主存与 I/O 部件之间传送数据时, 系统仍然需要积极的干预, 并且任何数据传送都必须完全通过 系统。对于以字节为单位传送的外设, 这种处理方式可以满足需求, 但对于存储类型的外设, 一次有大量的数据传送, 这种处理方式就会让人感到 系统干涉太多。

当外设与主存之间需要交换大量的数据时, 一种有效的技术就是 通道 (Channel)。通道的功能可由一个独立的 通道部件在系统总线上完成, 也可整合到 I/O 部件中, 由此 I/O 部件完成。不论哪种情况, 通道的传送方式都是相同的: 当 系统需要读或写一个数据块时, 它给 通道部件发命令, 命令中一般包含下述信息。

- (1) 操作类别: 读或写。
- (2) 所涉及的 I/O 设备的地址。
- (3) 读取或写入数据在内存中的首地址。
- (4) 读取或写入数据的字数。

发出命令后, 系统继续进行其他的工作。它把这次 I/O 操作委托给 通道部件, 由 通道部件处理。通道部件直接从主存或往主存中传送整个数据块, 每次传送一个块, 当传送完成后, 通道部件向 系统发中断信号。因此, 仅在数据块传送的开始及结束时需要 系统处理, 如图 8-11 所示。

在将数据读出、写入主存的过程中, 通道部件需要控制总线。由于 通道部件对总线的竞争使用, 有时会出现下面这种情况: 系统需要使用总线, 总线正被 通道部件控制, 系统不得不等待总线空闲, 这时需要暂停一个总线周期。最终的结果是, 在 通道传送期间, 系统的执行速度会慢下来。尽管如此, 对于大量的 I/O 传送来说, 通道比中断驱动和程序直接控制 I/O 更有效。

8.3.3 通道控制方式

I/O 通道控制方式与 通道方式类似, 也是一种以内存为中心, 实现外设与内存直接交换数据的控制方式。与 通道方式相比, 通道所需要 系统干预更少, 每次可以完成多个不连续的数据块传送, 而且可以做到一个通道控制多台设备, 从而进一步减轻了 系统的负担。

I/O 通道具有自己的指令系统, 并能实现指令所控制的操作。所以, 在 I/O 通道控制方

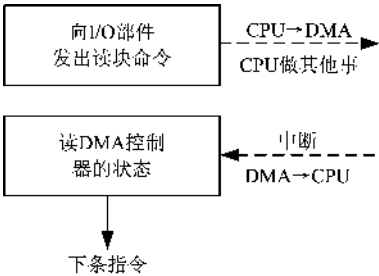


图 8-11 通道控制方式

式中, 系统只需要发出启动指令, 指出要求通道执行的操作和使用的 I/O 设备, 该指令就可以启动通道并从该通道内存中调出相应的通道程序执行。

中断技术

中断是操作系统中重要的概念, 它最初是为克服对 I/O 接口控制采用程序查询所带来的系统低效率而产生的, 从最初的外部中断到现在的内部中断, 计算机系统形成了一个完整的中断系统。

中断的基本概念

中断 (Interruption) 是由 I/O 设备或其他非预期的急需处理的事件引起的, 它使正在执行的程序中止, 转而执行另一个更紧迫事件的程序, 一旦处理完成, 自动返回原来被中断的程序。

中断源和中断分类

(1) 中断源: 引起中断的事件及发生中断请求的来源。

(2) 硬件中断和软件中断: 硬件中断是指由某个硬件中断请求信号引发的中断, 软件中断是由执行软件中断指令所引起的中断。对它们的处理是相同的, 其区别在于: 硬件中断通过中断请求信号形成向量地址, 而软件中断由指令提供中断号, 然后再转换为向量地址。

(3) 强迫中断与自愿中断: 强迫中断是由于故障、外部请求等所引起的强迫性中断, 非程序本身安排的。自愿中断是程序有意安排的, 以中断方式引出服务程序, 实现某种功能。

(4) 内中断和外中断: 内中断指来自主机内部的中断请求, 如掉电、系统故障等。外中断指中断源来自主机外部, 一般指外围设备中断, 如时钟中断、磁盘中断等。

(5) 可屏蔽中断和不可屏蔽中断: 外中断请求是由于某个外部设备的需要而提出的, 但是系统可以对此施加种种控制, 其中之一就是屏蔽技术。系统可向外部设备接口送出屏蔽字代码, 每位可以屏蔽一种中断源, 不允许它提出中断请求, 或者不允许它已经发送的中断请求信号送达系统。相应地, 可将中断请求分为可屏蔽中断与不可屏蔽中断两类。有些微处理器芯片如 8086 和 8088, 就将其中断请求输入分为可屏蔽中断 (INTR) 和不可屏蔽中断 (NMI) 两种。

(6) 向量中断与非向量中断: 如何根据中断源获得服务程序入口地址, 以实现程序切换, 这是一个重要的问题。根据这一过程是直接依靠硬件实现, 还是通过软件实现, 可分为向量中断与非向量中断。

中断优先级

在计算机系统中, 由于外部中断信号是随机产生的, 可能在同一时刻出现多个中断信号, 这样就存在谁先被响应和谁先被处理的优先次序问题。为了能够按照轻重缓急进行处理, 操作系统对不同的中断赋予不同的优先级。

对于一个实际的系统来说, 规定多少中断级, 每个中断应该分在哪一级, 都要由软、硬件设计者视系统的设计而定。一般说来, 高速设备的中断优先级高, 慢速设备的中断优先级低, 因为高速设备的中断被系统优先响应可以让系统更快地向高速设备发送下一个 I/O 请求, 提高高速设备的利用率。例如, 多道程序系统中的操作系统把中断级别分为三种:

- (3) 时钟中断，中断优先级为 远级。
- (4) 磁盘中断，中断优先级为 缘级。
- (5) 终端等其他外设中断，中断优先级为 源级。

对同一中断优先级内的若干中断源，可以按照中断寄存器中从左至右的顺序来决定处理的先后次序。在多级中断系统中，**系统**按照由高到低的顺序响应。当正在处理低级中断时，若出现了高级中断，则高级中断的处理立即打断低级中断的处理，多级中断的处理如图 4-10 所示。

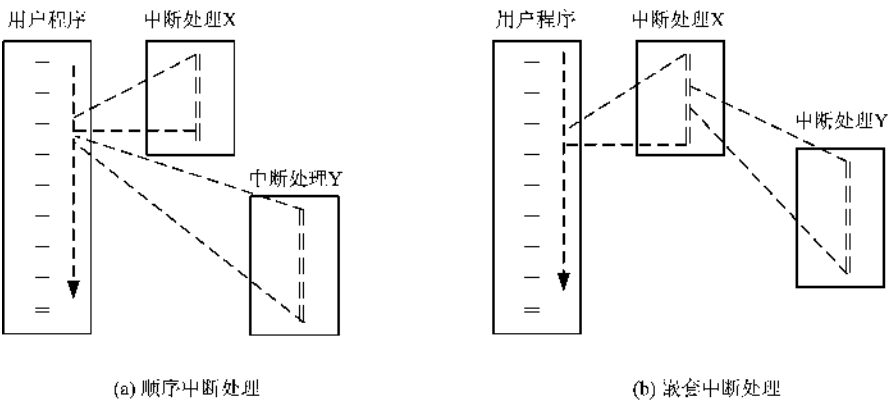


图 4-10 多级中断中的控制转换

4.1.2 中断处理

一个中断过程由以下几步组成：中断请求、中断响应、中断处理、中断返回。

4.1.2.1 中断请求的提出

由中断源向 **系统**发出，要满足以下两个条件：

- (1) 外设工作已经完成，即外设工作处于结束状态。
- (2) 该外设的中断没有被系统屏蔽。

4.1.2.2 中断响应

当设备提出中断请求后，**系统**中止现行程序的执行，转去为某个设备服务的过程，称为中断响应。中断响应要满足以下两个条件：

- (1) 中断源有中断请求。
- (2) **系统**允许接受中断请求。

系统响应中断的时间，应为一条指令执行完毕以后。

4.1.2.3 中断处理

系统在中断响应后转入中断服务程序，进入中断服务处理。不同的计算机系统对中断处理各不相同，但对大多数机器而言，中断处理过程如图 4-11 所示。

4.1.2.4 中断返回

处理完中断以后返回原先中止的程序，继续执行。

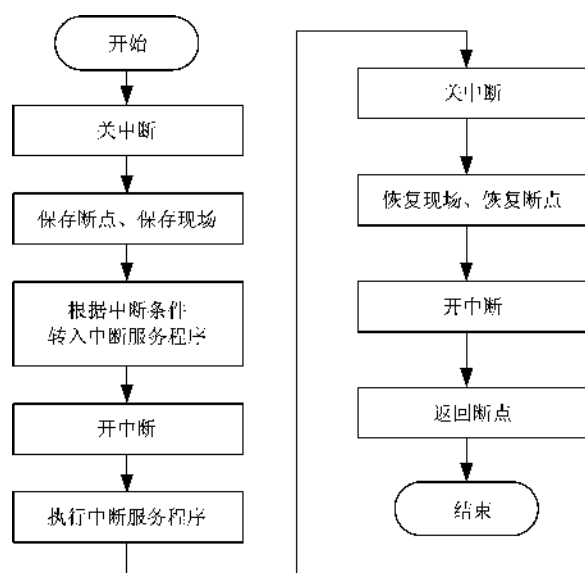


图 8-1-10 中断处理过程

愿缓冲技术

缓冲区是连接在 悦载与外部设备之间的数据存储区,缓冲技术是一项提高 悦载与 随机设备并行度的重要技术。

愿艱澀缓冲的引入

假设一个用户进程希望从磁带上读入一个数据块，数据块的长度为 2048 字节，读入的数据放在用户进程的虚地址 00000000 至 00000007 处。完成任务的最简单方法是执行一个 read 命令，然后等待数据，这个等待可以是一种忙等方式，或是一种更可行的方式，进程阻塞等待中断的发生。

这种方法存在这样一个问题：进程在等待较慢的 I/O 完成时，操作系统可能会将此进程交换出内存，进程虚地址空间 $00000000H$ 至 $FFFFFFFH$ 却必须保留在主存中，否则，某些数据就有可能丢失。若采用页式存储管理，至少 $00000000H$ 至 $FFFFFFFH$ 所在的页面必须保留在主存中，虽然这时操作系统可能已经选中该页将其淘汰出主存。同样的问题在输出时也可能发生，假设一个数据块正在从用户进程空间写到 I/O 部件中，在此传送过程中，进程被阻塞，但相应进程空间却不能换出主存。

为解决这些问题，有时候在发出请求之前就开始执行输入传送，在发送请求一段时间之后才开始执行输出传输，这项技术称作缓冲。

愿羧羧羧缓冲区的种类

根据缓冲区设置个数的多少，缓冲有单缓冲、双缓冲和多重缓冲之分，并将多重缓冲组

组织成循环缓冲形式，如图 4-10 所示。

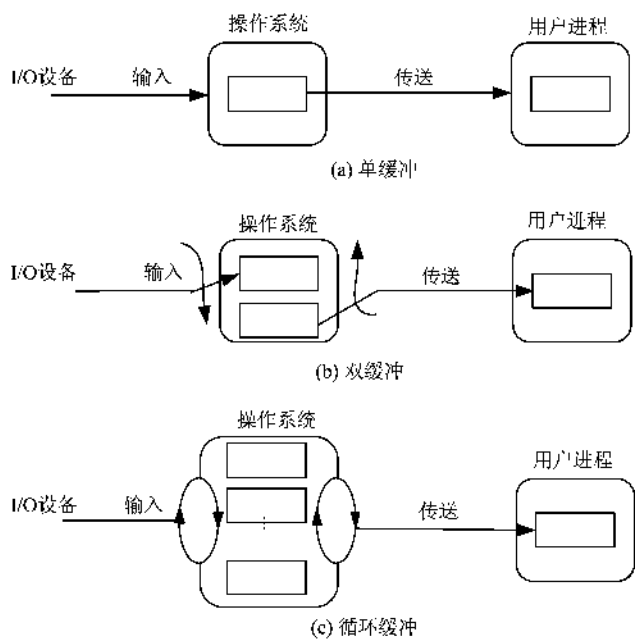


图 4-10 不同的缓冲方式

单缓冲

单缓冲是操作系统提供的一种最简单的缓冲形式，如图 4-10(a) 所示。当用户进程发出一个 I/O 请求时，操作系统便在内存中为它分配一个缓冲区。由于只设置了一个缓冲区，设备和处理机交换数据时，应先把要交换的数据写入缓冲区，然后，需要数据的设备或处理机从缓冲区取走数据，故设备与处理机对缓冲区的操作是串行的。

在块设备输入时，先从磁盘把一块数据输入到缓冲区，假设所花费的时间为 t_1 ，然后由操作系统将缓冲区数据传送到用户区，假设所需的时间为 t_2 ，接下来，用户进程对这一块数据进行计算，假设计算时间为 t_3 ，则系统对每一块数据的处理时间为 $t_1 + t_2 + t_3$ 。通常， t_2 远小于 t_1 或 t_3 。如果没有缓冲区，数据将直接进入用户区，则每块数据的处理时间为 $t_1 + t_3$ 。在块设备输出时，先将要输出的数据从用户区复制到缓冲区，然后再将缓冲区中的数据写到设备。

在字符设备输入时，缓冲区用于暂存用户输入的一行数据。在输入期间，用户进程阻塞以等待一行数据输入完毕；在输出时，用户进程将一行数据送入缓冲区后继续执行计算。当用户进程已有第二行数据要输出，而第一行数据尚未输出完毕，则用户进程阻塞。

双缓冲

引入双缓冲 (图 4-10(b))，可以提高处理机与设备的并行操作程度。在块设备输入时，输入设备先将第一个缓冲区装满数据，在输入设备装填第二个缓冲区的同时，操作系统可以将第一个缓冲区中的数据传送到用户区供处理机进行计算；当第一个缓冲区的数据处理完后，若第二个缓冲区已装满数据，则处理机又可以处理第二个缓冲区中的数据，而输入设备又可装填第一个缓冲区。显然，双缓冲的使用提高了处理机和输入设备并行操作的程度。只

有当两个缓冲区都为空，进程还要提取数据时，该进程阻塞。采用双缓冲时系统处理一块数据的时间可以粗略地估计为 $\frac{1}{2}(\frac{1}{\alpha} + \frac{1}{\beta})$ 。如果 $\alpha = \beta$ 则可使块设备连续输入；如果 $\alpha < \beta$ 则可使处理机连续计算。

在字符设备输入时，若采用行输入方式和双缓冲，则用户在输入完第一行后，系统执行第一行中的命令，而用户可以继续向第二个缓冲区输入下一行数据，因此用户进程一般不会阻塞。

循环缓冲

双缓冲方案在设备输入输出速度与处理机数据速度匹配时能获得较好的效果，但若两者速度相差甚远，双缓冲的效果则不够理想。为此，引入了循环缓冲技术（图 10.10）。

循环缓冲包含多个大小相等的缓冲区，每个缓冲区中有一个链接指针指向下一个缓冲区，最后一个缓冲区的指针指向第一个缓冲区，这样多个缓冲区构成一个环形。循环缓冲用于输入输出时，还需要两个指针 in 和 out 。对于输入而言，首先要从设备接收数据到缓冲区中， in 指针指向可以输入数据的第一个空缓冲区；当用户进程需要数据时，从循环缓冲中取一个装满数据的缓冲区，并从此缓冲区中提取数据， out 指针指向可以提取数据的第一个满缓冲区。显然，对输出而言正好相反，进程将处理过的需要输出的数据送到空缓冲区，而当设备空闲时，从满缓冲区中取出数据由设备输出。

缓冲池

循环缓冲一般适用于特定的 I/O 进程和计算进程，因而当系统中进程很多时，将会有许多这样的缓冲，这不仅要消耗大量的内存空间，而且其利用率也不高。目前计算机系统中广泛使用缓冲池。

缓冲池由多个缓冲区组成，其中的缓冲区可供多个进程共享，且既能用于输入又能用于输出。缓冲池中的缓冲区按其使用状况可以形成三个队列：空缓冲队列、装满输入数据的缓冲队列（输入队列）和装满输出数据的缓冲队列（输出队列）。除上述三个队列之外，还应具有四种工作缓冲区：用于收容输入数据的工作缓冲区、用于提取输入数据的工作缓冲区、用于收容输出数据的工作缓冲区及用于提取输出数据的工作缓冲区。

当输入进程需要输入数据时，便从空缓冲队列的队首摘下一个空缓冲区，把它作为收容输入工作缓冲区，然后把数据输入其中，装满后再将它挂到输入队列队尾。当计算进程需要输入数据时，便从输入队列取得一个缓冲区作为提取输入工作缓冲区，计算进程从中提取数据，数据用完后再将它挂到空缓冲队列队尾。当计算进程需要输出数据时，便从空缓冲队列的队首取得一个空缓冲区，作为收容输出工作缓冲区，在其中装满输出数据后，再将他挂到输出队列队尾。当要输出时，由输出进程从输出队列中取得一个装满输出数据的缓冲区，作为提取输出工作缓冲区，当数据提取完后，再将它挂到空缓冲队列的末尾。

设备分配

设备分配的任务是按照一定的算法将设备及有关资源分配给申请者进程。除分配设备外，有些系统还要分配缓冲区、控制器和通道。为了提高系统的适应性和均衡性，应考虑设备的独立性，即不能因物理设备的更换而影响用户程序的正常运行。同时，还必须考虑系统

的安全性,设备分配不能导致死锁现象的发生。

设备分配策略

前面已经指出,按照资源分配的角度,可以把设备分成独占设备、共享设备和虚拟设备三类,与此相应的分配策略就是独立方式、共享方式和虚拟方式。

独立方式就是把一台设备固定地分配给一个用户或进程,直到它运行结束。这种方式对用户来说是方便的,但往往造成资源浪费。

共享方式是指几个作业共享磁盘、磁带这类容量大、速度相对较快的设备,它们一般由文件系统或存储管理系统来分配和管理。

虚拟方式是通过虚拟技术将一台独占设备虚拟为共享设备,供多个进程以共享方式使用。大多数低速设备都属于独占设备,为了提高利用率而引入虚拟设备技术,通过快速的以共享方式使用的设备,把一台慢速的以独占方式工作的物理设备改造成为若干台虚拟的共享设备。这样,从用户角度看,每个用户都感觉到系统为自己提供了一台物理设备。

虚拟设备属于逻辑设备,但在操作系统中逻辑设备有更广泛的意义。下面仅就逻辑设备和虚拟设备技术作简单的说明。

物理设备和逻辑设备

为了提高操作系统的适应性和可扩展性,现代操作系统中都实现了用户程序独立于具体的物理设备,即设备无关性。在用户程序中不直接使用物理设备名,而只使用逻辑设备名。

逻辑设备名是实际物理设备属性的抽象,它并不局限于某个具体设备。例如在计算机系统中,一台名为“逻辑设备”的具有打印属性的逻辑设备,他可能是“物理设备”或“虚拟设备”,甚至有可能是由软件虚拟的,根本不存在的一台打印机。逻辑设备究竟和哪个具体物理设备相对应,这要由系统根据当时的设备使用情况来决定,或由用户指定。

在计算机系统中,用户程序使用逻辑设备名而不直接使用物理设备名,这种方式可以在某些物理设备发生意外情况无法正常工作或设备型号有变动时,不需要修改程序,只需要重新修改逻辑设备名与物理设备名之间的连接即可。

虚拟设备系统

虚拟设备的意思是指同时外部设备联机操作,又称为假脱机技术,是操作系统中采用的一项将独占设备改造成共享设备的技术。虚拟设备系统是对脱机输入输出工作的模拟,它必须有高速大容量且可随机存取的外存支持。在虚拟设备系统中,用一道程序来模拟脱机输入时外围设备控制器的功能,把低速输入设备上的数据传送到高速磁盘上;再用另一道程序来模拟脱机输出时外围设备控制器的功能,把数据从磁盘上传送到低速输出设备上。这样,便可以在主机的直接控制下,实现脱机输入输出功能。

虚拟设备系统的组成如图 8-1 所示,主要包括以下三个部分:

(1) 输入井和输出井

这是在磁盘上开辟出来的两个存储区域。输入井模拟脱机输入时的磁盘,用于收容设备输入的数据。输出井模拟脱机输出时的磁盘,用于收容用户程序的输出数据。

(2) 输入缓冲区和输出缓冲区

这是在内存中开辟的两个缓冲区。输入缓冲区用于暂存由输入设备送来的数据,以后再

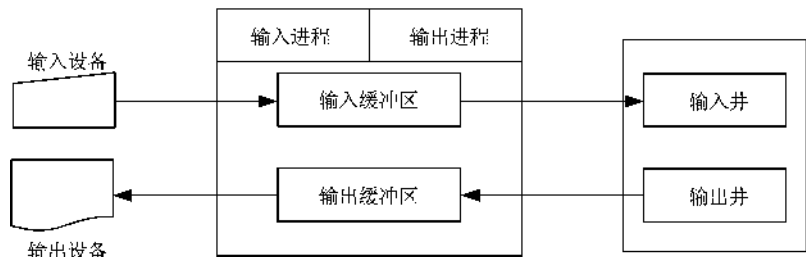


图 愿原系统组成

传送到输入井。输出缓冲区用于暂存从输出井送来的数据，以后再传送到输出设备。

③ 输入进程和输出进程

输入进程模拟脱机输入时的外围设备控制器，将用户要求的数据从输入设备通过输入缓冲区再送到输入井。当 悦载需要输入数据时，直接从输入井读入内存。输出进程模拟脱机输出时的外围设备控制器，把用户要求输出的数据，先从内存送到输出井，待输出设备空闲时，再将输出井中的数据，经过输出缓冲区送到输出设备上。

④ 设备分配算法

设备分配除了与设备的使用性质相关外，还与系统所采用的分配算法有关。常用算法如下：

① 先请求先服务。当有多个进程对同一设备提出 请求时，该算法根据这些进程发出请求的先后次序，将这些进程排成一个设备请求队列，设备分配程序总是把设备首先分配给队首进程。

② 优先级高者优先。按照进程优先级的高低进行设备分配。当多个进程对同一设备提出 请求时，哪个进程的优先级高，就先满足哪个进程的请求。对优先级相同的 请求，则按先请求先服务的算法排队。

⑤ 设备分配的安全性

所谓设备分配的安全性是指在设备分配中应保证不发生进程的死锁。

在进行设备分配时，可采用静态分配方式和动态分配方式。静态分配方式是在用户作业开始执行之前，由系统一次分配该作业所要求的全部设备、设备控制器和通道。这些设备、设备控制器和通道一旦分配，就一直为该作业占用，直到该作业被撤销为止。静态分配方式不会出现死锁，但设备的使用效率低。

动态分配是在进程执行过程中根据执行需要进行设备分配。当进程需要设备时，通过系统调用命令向系统提出设备请求，由系统按照事先规定的策略给进程分配所需要的设备、设备控制器和通道，一旦使用完之后便立即释放。动态分配方式有利于提高设备的利用率，但如果分配算法使用不当，则有可能造成进程死锁。

在设备的动态分配方式中，分为安全分配和不安全分配两种方式。在安全分配方式中，每当进程发出 请求后便立即进入阻塞状态，直到所提出的 请求完成后才唤醒进程并释放设备。当采用这种分配策略时，一旦进程获得某种设备后便阻塞，使该进程不可能再请求其他设备，因而这种分配方式是安全的，但进程推进缓慢。

在不安全分配方式中，允许进程发出 I/O 请求后仍继续运行，且在进程需要时又可发出第二个 I/O 请求，第三个 I/O 请求……。仅当进程所请求的设备已被另一个进程占用时才进入阻塞状态。这样，一个进程有可能同时操作多个设备，从而使进程推进迅速，但这种设备分配方式有可能产生死锁。

设备分配程序

操作系统的设备分配程序负责为对系统提出 I/O 请求的进程分配设备，为确保进程与设备之间能进行通信，还应分配相应的控制器和通道。若所用通道是选择型通道，则设备应独占该通道和控制器。若是其他两种类型的通道，则无需独占通道。

设备分配中的数据结构

为了实现对 I/O 设备的管理和控制，需要对每台设备、通道、设备控制器的有关情况记录。设备分配依据的主要数据结构有设备控制表（DCT）、设备控制器控制表（COCT）、通道控制表（CHCT）和系统设备表（SDB）。图 8-1 给出了这些表的数据结构。

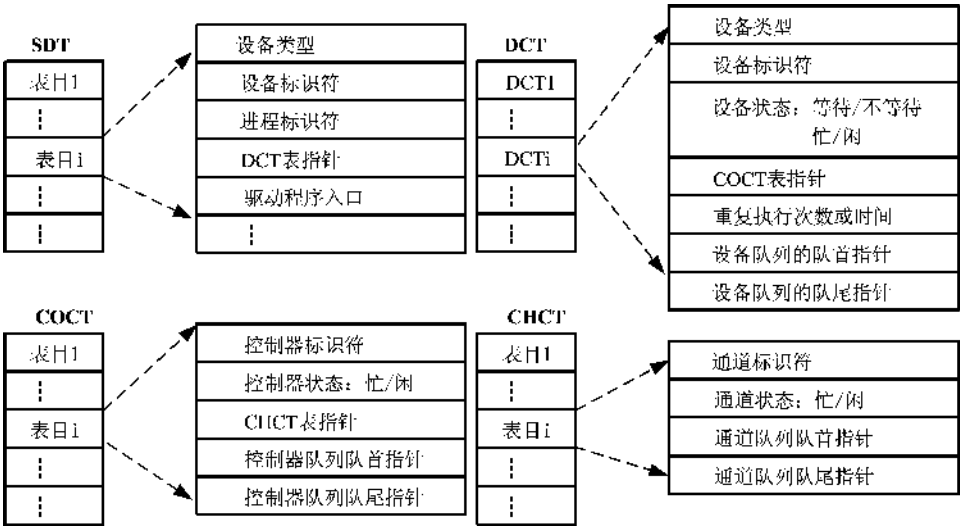


图 8-1 设备分配的数据结构

系统设备表 SDB 是系统范围的数据结构，它记录了系统拥有的全部 I/O 设备。每个设备一个表目，表目中的进程标识符项是指已经分配到此设备的进程，若未分配，应是无效内容。

设备控制表 DCT 中的设备状态项，有忙/闲和等待/等待两类标记。若设备自身忙，则应置位忙标记。若此时与设备连接的所有控制器或通道都忙，而对请求设备的进程无法分配相应的控制器或通道的话，则应在 DCT 表中置位等待标记，并且进程的 I/O 请求应插入该控制器或通道的等待队列。

DCT 中的重复执行次数或时间项表示，某些系统中若发生信息传送错误，系统并不立即认为传送失败，而允许重试。只要在规定的重复次数或时间内恢复正确的传送，则认为此次传送成功，否则才宣布此次传送操作失败。

就绪、就绪或就绪中的队列分别存放等待此设备、等待此控制器或等待此通道的进程。进程的等待按一定的策略排队。

设备分配流程

当系统中已经具备了上述数据结构，并且确定了一定的分配原则后，若某进程提出请求，便可进行设备分配。分配包括分配设备、分配控制器和分配通道，设备分配的流程如图 8.1 所示。

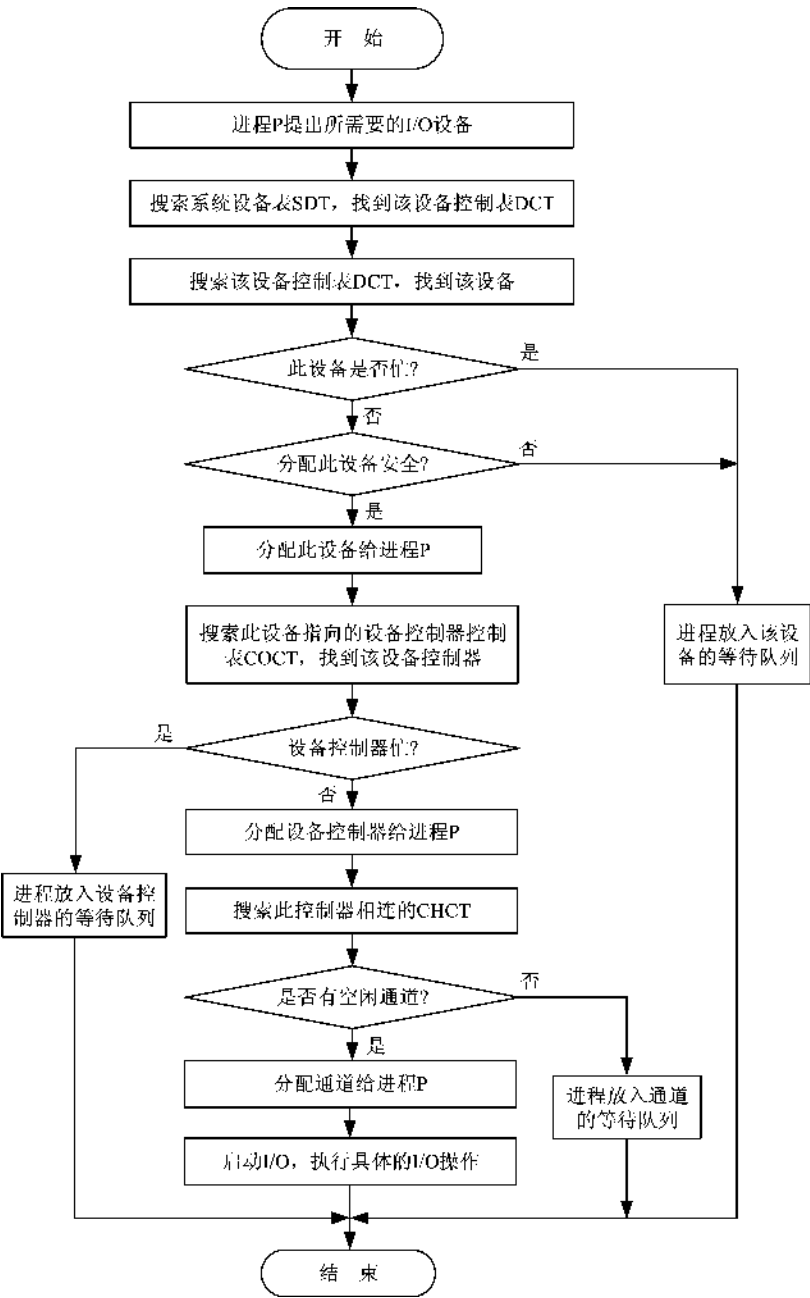


图 8.1 设备分配流程示意

图 1-1 所示的分配流程只是单通路的 I/O 系统，即每个 I/O 设备只有一条通路与内存连接。实际上，大型系统为了可靠性和灵活性，往往采用多通路的 I/O 系统。这样一个 I/O 设备可以连至几个控制器，每个控制器又连至几个通路，使得设备分配更为复杂。

当进程已获得了所需的设备、控制器和通道后，便可执行具体的 I/O 操作。对于设有通道的系统，I/O 启动 I/O 则就是调用相应的通道程序，由通道专用处理机来执行通道程序以完成 I/O 操作。

目前，大多数微型机或小型机并不设置通道。在这种情况下，I/O 启动 I/O 是调用相应的设备驱动程序和 I/O 处理程序，由 CPU 直接向 I/O 设备发出指令，控制设备来完成所需的 I/O 操作。

I/O 软件的层次结构

I/O 软件设计的基本思想是将设备管理软件组织成一种层次结构。每一层都有独立的功能，并定义好与相邻层的接口。底层软件对高层软件隐藏了硬件的具体特性，而高层软件为用户提供了清晰、方便的统一界面。

可把 I/O 软件构成分为如下四个层次，如图 1-2 所示。在不同的系统中上述各层的功能和接口是有差异的。

用户层 I/O 软件
与设备无关的 I/O 软件
设备驱动程序
中断处理程序
硬件

图 1-2 I/O 软件的层次结构

用户层 I/O 软件

虽然 I/O 软件的实现大部分在操作系统的核心中，但是用户程序一般直接使用 I/O 库函数进行输入输出，如 C 语言库中的函数 `write()`，`read()` 等。这些库函数在用户态运行，它们往往没有做太多的事情，而只转调操作系统 I/O 相关的系统调用。操作系统的系统调用也是以函数界面形式供用户程序调用，如 C 语言库中的 `fork()`，`exec()` 等。

这个层与设备的控制细节无关。它将所有的设备都看成逻辑资源，它为用户进程提供各类 I/O 函数，允许用户进程通过设备标识符和一些简单的函数接口来使用设备，如打开、关闭、读、写等函数。所以这个层次主要的任务是为相应的系统调用提供参数。

与设备无关的 I/O 软件

这一层及以下的功能一般在操作系统内核中实现。该层对上层提供了系统调用的接口，对下层通过设备驱动程序接口调用设备驱动程序。

与设备无关的 I/O 和设备驱动程序之间的精确界限在各个系统都不尽相同。对于一些与

设备无关方式完成的功能，在实际中由于考虑到执行效率等因素，也可由驱动程序完成。

图 1-1 给出了与设备无关软件所实现的典型功能。它的基本功能是执行适用于所有设备的通用功能。

设备驱动程序统一接口
设备命名
设备保护
提供与设备无关的逻辑块
缓冲
块设备上的存储分配
独占设备的分配和释放
错误报告

图 1-1 与设备无关软件的功能

设备命名

用户通过设备名指定申请使用的设备，使用设备要通过设备驱动程序，与设备无关的任务之一就是要将设备名映射到相应的驱动程序。在系统中，一个设备名对应一个特殊的设备文件，如 `/dev/lp` 唯一地确定了一个设备数据结构，其中包含了主设备号和次设备号。主设备号用于寻找对应的设备驱动程序，而次设备号提供了设备驱动程序的参数，用来确定要读写的具体设备。

设备保护

设备保护防止未授权用户对设备的存取。有些微机系统（如 `PC`）完全“开放”，没有保护措施，任何进程都可以干它想干的事情。而多数大型机系统中完全禁止用户进程存取设备。系统中采取灵活方式，利用特殊文件的存取权限控制进程的存取。系统管理员可以根据需要为每一个设备设置适当的存取权限。

提供与设备无关的逻辑块

不同的磁盘可以采用不同的扇区尺寸。与设备无关软件的一个任务就是向较高层次软件隐藏这些差异并提供统一的盘块大小。例如，可以将若干扇区合并成一个逻辑块，这样高层软件只与抽象设备打交道，不考虑物理扇区的尺寸而使用等长的逻辑块。

缓冲

如前所述，缓冲技术可解决数据传输过程中的瓶颈问题，提高并行程度，加快处理速度。对块设备和字符设备来说，通常都要使用缓冲区。

块分配及释放设备

一些设备，如打印机，在同一时刻只能由一个进程使用。这要求操作系统检查对该设备的使用请求，并根据设备的忙闲状况来决定是接受或拒绝此请求。一般是在设备申请时，通过调用系统调用打开相应的设备文件来进行申请。若设备不可用，则申请失败。关闭（`close`）独占设备的同时释放该设备。有些操作系统把设备分配和释放工作看成设备相关的工作，这时，系统调用处理程序要调用设备驱动程序对应的函数进行处理。这是因为设备使用方式不同使得设备分配申请处理不一样。如申请独占方式使用时，只有当设备未被独

占时才可以成功分配,而分时共享式使用的设备就随时可以分配成功。

8.1.2 错误报告

对错误的处理,基本上是由驱动程序实现的。多数错误是与设备紧密相关的,因此只有驱动程序知道应如何处理。经过处理如仍有错,则通知与设备无关软件。在这一层对错误的处理是与设备无关的。如果读用户文件时出错,只要把错误返回调用者就可以了。

8.1.3 设备驱动程序

所有与设备相关的代码都放在设备驱动程序中,由于设备驱动程序与设备密切相关,故应为每一类设备配置一个驱动程序、或为一类密切相关的设备配置一个驱动程序。

设备驱动程序的任务是接收来自上层的与设备无关软件的抽象请求,将这些请求转换成设备控制器可以接受的具体命令,再将这些命令发送给设备控制器,并监督这些命令是否正确执行。如果请求到来时设备驱动程序是空闲的,它立即开始执行这个请求;若设备驱动程序正在执行一个请求,则它将新到来的请求插入到等待队列中。设备驱动程序是操作系统中惟一知道设备控制器中设置了多少个寄存器,这些寄存器有何用途的程序。

8.1.4 中断处理程序

中断处理程序在系统软件层次结构的最底层。当进程进行 I/O 操作时,将其阻塞至 I/O 操作结束并发生中断。中断发生时,由中断处理程序执行相应的操作并解除相应进程的阻塞状态,使其能够继续执行。同时,中断处理程序应该从设备请求队列中获得下一个设备驱动请求并驱动设备。

8.2 磁盘管理

8.2.1 磁盘性能参数

从物理性能上看,磁盘是比较简单的,盘片由一些特殊材料组成,它的两个表面上覆盖着磁性材料,信息就记录在盘的表面上。

当使用磁盘时,驱动电机带动磁盘高速旋转,磁盘的表面定位一个读写磁头,系统的信息通过读写磁头可由磁盘读出或写入磁盘。磁盘寻址方式是多维的,即读写磁盘中的一个扇区通常应给出驱动器号、柱面号、磁头号和扇区号。为了减小移动臂移动所花费的时间,每个文件信息不是按照盘面上的磁道顺序存放信息,而是按照柱面顺序存放。即同一个柱面上各个磁道放满后,再放另一个柱面。由此可见,访问一个磁盘块,其访问时间由寻道时间、旋转延迟时间和读写传输时间三部分组成。

8.2.1.1 寻道时间

磁盘在读写信息之前,必须首先将磁头移到相应的柱面(磁道)。磁头这种定位柱面所花费的时间称为寻道时间。该时间是启动磁臂的时间与磁头移动到目标磁道所花费的时间之和,即

$$T_{\text{seek}} = T_{\text{start}} + T_{\text{move}}$$

其中, T_{start} 是一常数,与磁盘驱动器的速度有关,对于高速磁盘, T_{start} 很小;磁臂的启动时

间约为 $\frac{1}{2} \times \frac{1}{\text{转速}}$ 这样，寻道时间将随寻道距离的增加而增加，大体上是 线性

旋转延迟时间 (Rotational Latency)

这是读磁头定位于某一磁道的块号 (扇区) 所需要的时间。对于硬盘，其旋转速度为 $\frac{1}{\text{转速}}$ ，相当于每 $\frac{1}{\text{转速}}$ 转一周。因此，速度为 $\frac{1}{\text{转速}}$ 时，平均旋转延迟时间为 $\frac{1}{2} \times \frac{1}{\text{转速}}$ 。软盘的转速通常在 $\frac{1}{\text{转速}}$ 到 $\frac{1}{\text{转速}}$ 之间，因而平均延迟在 $\frac{1}{2} \times \frac{1}{\text{转速}}$ 到 $\frac{1}{2} \times \frac{1}{\text{转速}}$ 之间。

数据传输时间 (Transfer Time)

这是指把数据从磁盘读出或向磁盘写入数据所经历的时间。延迟的大小取决于磁盘的旋转速度：

延迟时间 $\propto \frac{1}{\text{转速}}$

其中 $\frac{1}{\text{转速}}$ 为磁盘每秒钟的转数， $\frac{1}{\text{转速}}$ 为要传送的字节数， $\frac{1}{\text{转速}}$ 为一条磁道上的字节数。

因此，可以将访问时间表示为：

延迟时间 $\propto \frac{1}{\text{转速}}$

考虑一个典型的磁盘，平均寻道时间为 $\frac{1}{\text{转速}}$ ，转速为 $\frac{1}{\text{转速}}$ ，每个磁道有 $\frac{1}{\text{转速}}$ 个扇区，每个扇区 $\frac{1}{\text{转速}}$ 个字节。假设读取一个包含 $\frac{1}{\text{转速}}$ 个扇区的文件，大小为 $\frac{1}{\text{转速}}$ ，现在估计传送需要的时间。

首先，假设文件尽可能紧凑地保存在磁盘上，也就是说，文件占据了 $\frac{1}{\text{转速}}$ 个相邻磁道中的所有扇区 ($\frac{1}{\text{转速}}$ 个磁道 $\frac{1}{\text{转速}}$ 个扇区 $\frac{1}{\text{转速}}$ 个字节)。那么，读第一个磁道的时间如下：

延迟时间 $\propto \frac{1}{\text{转速}}$

延迟时间 $\propto \frac{1}{\text{转速}}$

延迟时间 $\propto \frac{1}{\text{转速}}$

延迟时间 $\propto \frac{1}{\text{转速}}$

假设现在可以不需要寻道时间读其余的磁道。那么，最多只需要为随后的每个磁道处理旋转延迟。因此，后面的每个磁道可以在 $\frac{1}{\text{转速}}$ 内读入。所以，读取整个文件的时间为：

延迟时间 $\propto \frac{1}{\text{转速}}$

假设访问是随机分布在磁盘上。那么，对每个扇区，有：

延迟时间 $\propto \frac{1}{\text{转速}}$

延迟时间 $\propto \frac{1}{\text{转速}}$

延迟时间 $\propto \frac{1}{\text{转速}}$

延迟时间 $\propto \frac{1}{\text{转速}}$

此时，读取整个文件的时间：

延迟时间 $\propto \frac{1}{\text{转速}}$

显然，从磁盘读扇区的顺序对系统性能有很大的影响，分析可以提高磁盘性能的方法是非常有必要的。

磁盘调度策略

在前面的例子中，产生性能差异的原因可以追溯到寻道时间。如果扇区访问请求包括随

机选择磁道，磁盘的随机性能会非常低。为了提高磁盘的性能，需要减少花费在寻道上的时间。

考虑在多道程序环境中的一种典型情况，操作系统为每个随机设备维护一个请求队列。因此对一个磁盘，队列中可能有来自多个进程的许多随机请求（读写）。如果按随机顺序从队列中选择项目，那么磁道完全是被随机访问的，这种情况下的性能是最差的。所以，需要按照一定的策略来调度磁盘的访问。常见的磁盘调度策略有先来先服务、最短寻道时间优先及扫描调度等。

先来先服务

这是最简单的磁盘调度策略，它根据进程请求访问磁盘的时间顺序进行调度。这个策略具有公平的优点，因为每个请求都会得到处理，并且是按照接收到的请求顺序进行处理。在以下的例子中，假设磁盘有 184 个磁道，磁盘请求队列中是一些随机请求。被请求的磁道，按照接收顺序分别为 18、38、39、55、58、90、100、150、160、184。当前磁头在 90 磁道处。图 4-1 显示了磁头臂按先来先服务策略移动的轨迹，表 4-1 给出了相应的结果。

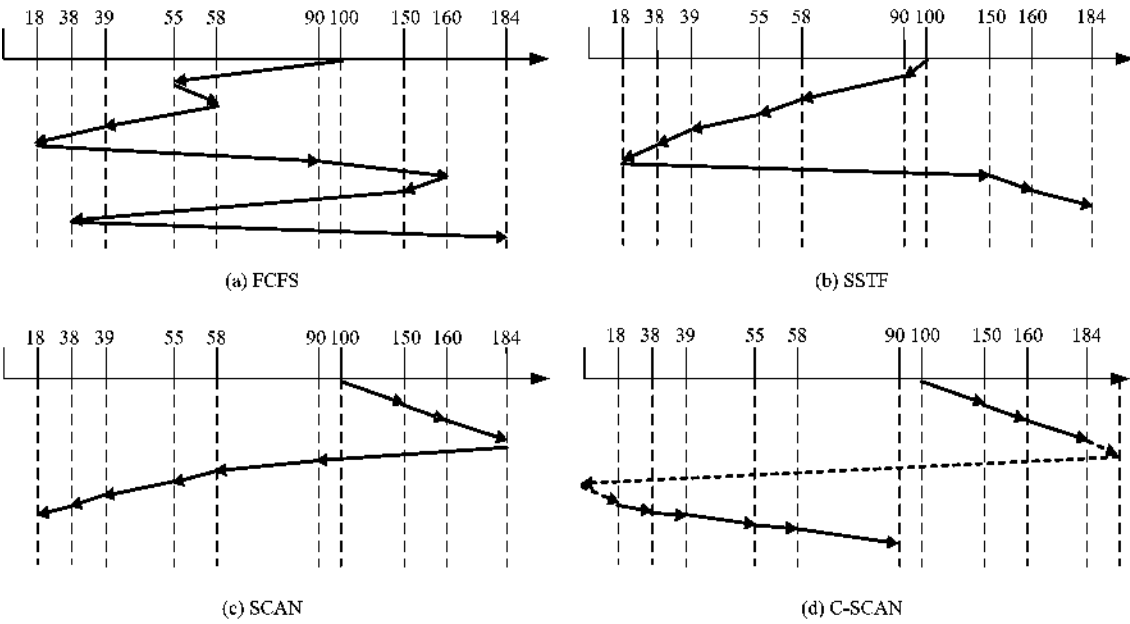


图 4-1 磁头臂移动轨迹图

对于先来先服务，如果只有一些进程需要访问，并且如果大多数请求都是访问簇聚的文件扇区，则有望达到更好的性能。但是，如果有多个进程竞争一个磁盘，这种策略在性能上往往接近随机调度。因此，需要考虑一些更复杂的调度策略。

最短寻道时间优先

最短寻道时间优先策略根据磁头的当前位置，选择请求队列中距离磁头最短的请求作为响应对象。因此，最短寻道时间策略总是选择导致最小寻道时间的请求。当然，总是选择最小寻道时间并不能保证平均寻道时间最小，但它能提供比先来先服务更好的性能。由于磁头臂可以沿两个方向移动，

因此可以使用一种随机选择算法解决距离相等的情况。图 愿愿源遭显示了磁头臂按 灾灾云策略移动的轨迹，表 愿愿员遭给出了相应的结果。

灾灾云策略的平均寻道时间小于 云灾云策略，但它不是最优策略。并且，灾灾云策略可能导致队列中某些请求长时间得不到服务而被饿死。在实际系统中，请求队列中随时可能增加新的请求。假设队列中存在访问磁道 愿和 愿源请求，如果磁头正在磁道 愿 而且在磁道 愿附近频繁地增加新的请求，那么 灾灾云调度策略使得磁头长时间在磁道 愿附近工作，而磁道 愿源的访问被迫长时间等待。从理论上讲，如果磁道 愿附近新的请求连续到达，将使磁道 愿源的访问无限期地延迟，即被饿死。

表 愿愿源磁盘调度算法的比较

(葬云灾灾云		(遭灾灾云		(糟灾灾灾		(凿悦愿灾灾灾	
磁道访问顺序	横跨的磁道数	磁道访问顺序	横跨的磁道数	磁道访问顺序	横跨的磁道数	磁道访问顺序	横跨的磁道数
缘	源	愿	愿	员愿	缘	员愿	缘
愿	猿	愿	猿	员愿	愿	员愿	愿
猿	愿	缘	猿	员源	愿源	员源	愿源
愿	愿	猿	愿	愿	愿	愿	愿
愿	愿	猿	员	愿	猿	猿	愿
员愿	猿	愿	愿	缘	猿	猿	员
员愿	愿	员愿	员愿	猿	愿	缘	愿
猿	员愿	员愿	愿	猿	员	愿	猿
员源	员愿	员源	愿源	愿	愿	愿	猿
平均寻道长度	缘猿	平均寻道长度	愿猿	平均寻道长度	愿猿	平均寻道长度	愿猿

猿灾灾灾策略

考虑到磁盘队列的动态性，为避免队列的某些请求被饿死，一种比较简单的方法是 灾灾灾策略，有时候也称“电梯”策略，因为这种方式与电梯在各楼层间的往返移动非常类似。

灾灾灾策略要求磁头臂仅仅沿一个方向移动，并在途中满足所有未完成的请求，直到它到达这个方向上的最后一个磁道，或者在这个方向上没有别的请求为止。然后倒转服务方向，沿相反方向扫描，同样按顺序完成所有请求。图 愿愿源糟显示了磁头臂按 灾灾灾策略移动的轨迹，表 愿愿员糟给出了相应的结果。可以看出，灾灾灾策略的行为和 灾灾云策略非常类似，实际上，如果在开始时，假设磁头臂沿着磁道号减少的方向移动，那么 灾灾云和 灾灾灾的调度方式是相同的。但是，灾灾灾策略对最近经过磁道的访问请求不立即响应，比如，如果磁头刚刚移动过后的一个磁道上的访问请求，只能等到磁头反方向移动到该磁道时才能得到处理。因此，它在局部性方面不如 灾灾云好。

源悦愿灾灾灾策略

假定对磁道的请求是均匀分布的，考虑对磁道的请求密度。当磁头到达一端并反向时，

落在磁头之后的请求相对较少,这是由于这些磁道刚刚被处理过;而磁盘另一端的请求密度相当高,而且这些请求等待的时间也较长。为了解决这种问题,引入了循环扫描法(即循环扫描策略),以提供比较均衡的等待时间。循环扫描策略与先来先服务策略类似,也是把扫描限定在一个方向。当沿某个方向访问到最后一个磁道时,磁头臂返回到磁头的另一端,并再次开始扫描。循环扫描策略将磁盘上的磁道排列视为一个圆,最后一个磁道与第一个磁道紧密相接,以此达到对磁道上请求的均衡服务。图 8-10 显示了磁头臂按循环扫描策略移动的轨迹,表 8-1 给出了相应的结果。

值得注意的是,先来先服务策略和循环扫描策略都将磁头由磁盘的一端移向另一端,但实际上没有一个策略是这样实现的。通常,磁头在向任何方向移动时都只移动到最远的一个请求磁道上。一旦在前进的方向上没有请求到达,磁头就反向移动,这种方式的扫描及循环扫描称为磁头寻道和循环磁头寻道策略。

磁头寻道策略

对于先来先服务和循环扫描,磁头臂可能会一段很长的时间停留在某处不动。例如,如果一个或多个进程对一个磁道有较高的访问频度时,它们可以通过重复地请求这个磁道垄断整个设备。高密度磁盘更容易受这种特性的影响。为避免这种“磁头臂的粘性”,磁盘请求队列被分成段,一次只有一段被完全处理。这种方法的两个例子是磁头寻道策略和磁头寻道策略。

磁头寻道策略把磁盘请求分成长度为 N 的子队列,每次用先来先服务策略处理一个子队列。在处理某一个队列时,新请求必须添加到其他某个队列中。如果在扫描的最后剩下的请求数小于 N ,则它们全都将在下一次扫描时处理。对于比较大的 N 值,磁头寻道策略的性能接近先来先服务;当 $N=1$ 时,退化为先来先服务。

磁头寻道是一种使用两个子队列的策略。当扫描开始时,所有请求都在一个队列中,而另一个队列为空。在扫描过程中,所有新到的请求都被放入另一个队列。因此,对新请求的服务延迟到处理完所有老请求以后。

磁盘调度策略很多,如何选择一个有效的算法呢?先来先服务是公认的、最具吸引力的策略,先来先服务和循环扫描对于磁盘负载较重的系统更为合适。然而,任何调度策略性能的优劣都是与进程对磁盘的请求数量和方法紧密相关的。当磁盘等待队列中的请求数量很少超过一个时,所有的策略都是等效的。在这种情况下,最好采用先来先服务策略。另外,文件的分配方式,目录及索引块的位置也会在很大程度上影响对磁盘的请求服务。

提高磁盘性能与容错性的冗余阵列

提高磁盘性能和容错性的机制就是采用独立磁盘的冗余阵列(RAID)。冗余阵列是一种将数据分布在多个磁盘上,从而提高磁盘上数据传输的速度和改善磁盘的容错能力。目前公认的 RAID 方案包括第 0~5 共 6 种,简称 RAID 0~RAID 5。

RAID 0 第 0 层

RAID 0 第 0 层使用磁盘阵列在块一级,以条带(Striping)形式将数据循环地分布在各磁盘上。一个条带可以是一个物理块、扇区或其他单位。它既不包含磁盘镜像,也不包含奇偶校验位的任何冗余信息。当两个或多个不同的 I/O 请求所涉及的数据块在不同的磁盘上时,可以实

现并行存取，因此具有较高的数据传输率。

圆层

它通过复制所有的数据来实现冗余。它将 圆层中的每个数据条带完全影射到两个不同的物理磁盘上，从而实现磁盘镜像。这样不仅可以提高存取数据的速度，而且提高了数据冗余能力。当然，它是以牺牲磁盘容量为代价的，主要用来保存系统软件和其他极其重要的文件。

猿层

在 猿层，采用以字节或字为单位的奇偶校验方式。对磁盘上的每个字节中的位，计算一个错误校验位，并且将它存储在多个奇偶校验磁盘中的相应位中。如果一个字节中有一位被破坏，该字节的奇偶校验码改变，使得与存储的奇偶校验位不匹配。反之，若存储的奇偶校验位被破坏，它将不匹配计算出的奇偶校验位。一个磁盘的错误，可以通过该字节的其他位和相关的奇偶校验位来恢复。

源层

源层与 猿层类似，只是不管数据磁盘阵列有多大，仅需一个冗余磁盘。它采用位交叉奇偶校验组织（月），系统只检查是否正确地读一个扇区，因此只需要一个奇偶校验位就可以进行检查和错误修正。其想法为：如果一个扇区损害，就计算该扇区的各奇偶校验位是 员还是 园。若这些位的计算结果与存储的奇偶校验相等，则丢掉的是 园，否则是 员。由此可见，它与 猿层实现相同的功能，但开销却大大降低。

缘层

源层采用块交叉奇偶校验组织（月），与 猿层一样，使用块级条带。对每个条带逐位计算奇偶校验，并存储在奇偶校验磁盘的相应条带上。如果一个磁盘受损，则用奇偶校验盘上的奇偶校验块恢复。这种模式非常适合并行存取。

远层

与 源层类似，只是 远层把奇偶校验条带分布在所有磁盘上，从而避免奇偶校验盘的 瓶颈。

远层

远层又叫 冗余模式。它采用 源层和 缘层使用的异或算法和独立数据校验算法。两种不同的冗余信息存储在不同磁盘的不同块中。每 源个数据位存储两个冗余位。这样即使有两个磁盘失效，也可重新恢复被破坏的数据，从而提供极高的数据可用性。

哉系统设备管理

概述

哉系统的设备管理包含在内核中，文件系统作为用户进行系统调用的接口。哉系统包括两类设备：块设备和字符设备。每个设备在文件系统目录中都占一个节点。设备作为特殊

文件，有一个文件名，也标志了文件类型和访问权限等，对设备的使用如同对文件的存取。

文件系统与设备驱动程序之间的接口是设备转换表，分为块设备转换表和字符设备转换表。设备转换表记录了每类设备驱动程序的入口地址，由系统启动时根据硬件配置建立，如图 8-1 所示。

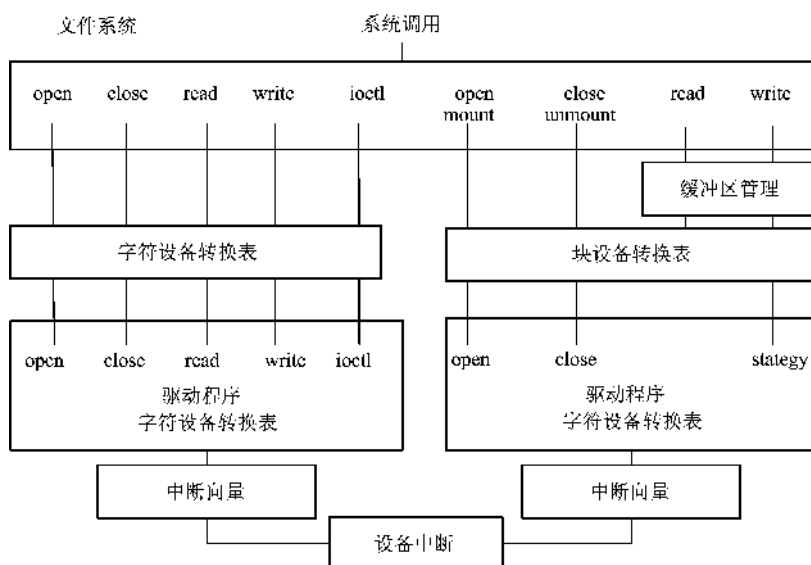


图 8-1 文件系统与设备驱动程序接口

8.1 块设备管理

8.1.1 块设备的主要数据结构

在块设备管理中，主要设置三种数据结构：缓冲存储区及缓存控制块、块设备表和块设备开关表

（1）缓冲存储区及缓存控制块

在块设备管理模块中，设置了若干缓冲存储区，以构成缓冲池。每个缓冲区的容量为 4096 字节 ~ 16384 字节。系统还为每个缓冲区设置了一个缓存控制块，用来记录缓冲区的使用情况，它的定义如下：

缓存控制块

{

缓冲标志信息 * 缓存标志

设备队列向前和向后指针 * 缓存标志

设备队列向前和向后指针 * 缓存标志

自由队列向前和向后指针 * 缓存标志

自由队列向前和向后指针 * 缓存标志

对应该缓冲区的设备名 * 缓存标志

对应该缓冲区的设备名 * 缓存标志

单总线映射描述符 * 缓存标志

单总线映射描述符 * 缓存标志

传输计数 * 缓存标志

传输计数 * 缓存标志

急露出

摇摇糟磁删成遭磁删

摇摇蛋城*遭增巢

摇摇泽磁删成遭磁删*遭磁删

摇摇泽磁删成遭磁删*遭磁删;

摇摇磁磁删成*遭磁磁删

摇摇}遭磁;

磁磁删成遭磁删

糟磁删遭磁删

急露巢蛋城遭磁删

磁巢城遭磁删

泽磁删成*遭磁删

}

辕 缓冲区在主存的地址 * 辕

辕 要清除的字 * 辕

辕 超级块的缓冲区的指针 * 辕

辕 节点表指针 * 辕

辕 文件索引的间接块指针 * 辕

辕 对应设备上的物理块号 * 辕

辕 磁删完成返回的错误码 * 辕

辕 出错后未传输完的字数 * 辕

辕 请求开始时间 * 辕

辕 请求 磁删的进程指针 * 辕

下面对其各字段进行说明：

①标志信息 遭磁删 与描述相应缓冲区的使用情况及 磁删方式有关，共 员园位。各位的含义如下：

磁磁删月卒磁磁磁磁磁磁磁磁磁磁 辕 第 园位为 园表示写，为 员表示读 * 辕

磁磁删月磁磁磁 员

磁磁删月磁磁磁 圆 辕 当 磁删完成时置位 * 辕

磁磁删月磁磁磁磁 园原 辕 当 磁删故障时置位 * 辕

磁磁删月磁磁磁 园园 辕 相应缓冲区正在传输信息时设置 * 辕

磁磁删月磁磁磁 园园 辕 隐含了物理 磁删使用单总线时设置 * 辕

磁磁删月磁磁磁 园园 辕 这个块已用做单总线映射时置位 * 辕

磁磁删月磁磁磁磁 园园园 辕 有进程等待使用缓冲区时置位 * 辕

磁磁删月磁磁磁 园园园 辕 延迟写时修改年龄标记为旧，以便释放后放到空闲队列头部 * 辕

磁磁删月磁磁磁磁 园园园 辕 异步实现 磁删时置位 * 辕

磁磁删月磁磁磁磁磁 园园园 辕 相应缓冲区移作它用时，才写盘时被设置延迟写标志 * 辕

磁磁删月磁磁磁 园园园 辕 正调用打开例程 * 辕

磁磁删月磁磁磁 园园园 辕 保留（传输出错时的标志）* 辕

②队列指针信息。遭磁删 遭磁删 葬磁删 葬磁删是两对队列指针。

③存储设备及其上数据块的位置信息。遭磁删是逻辑设备号，定义如下：

泽磁删 糟磁删磁磁删

糟磁删磁磁删

}

该结构的低字节 磁磁删是设备的编号，又称次设备号。高字节 磁磁删是设备类型号，又称主设备号。

遭磁删是该缓存块对应设备上的物理块号。由 遭磁删和 遭磁删使一个缓冲区与某个特

定设备上的一个字符块建立起对应关系。

④ 存储区位置信息 用来存放缓冲区的首地址。

⑤ 请求及其执行结果信息。为要传输的字节数或字计数；为请求出错时，返回的错误信息；为出错时，尚未传输完的剩余字数；为请求传输的开始时间。

⑥ * 为当前正在利用该缓冲区进行物理请求的进程 结构指针。

块设备表

为了管理方便，操作系统为每个块设备控制器设置块设备表 定义如下：

块设备表

```
{
    与 的 相同 *
    设备忙闲标志 *
    请求出错次数计数器 *
    * 指向这个设备的第一个缓冲区 *
    * 指向这个设备的最后一个缓冲区 *
    * 指向这个设备请求队列的第一个缓冲区 *
    * 指向这个设备请求队列的最后一个缓冲区 *
    设备名 *
    * 指向块设备错误记录块 *
    设备寄存器个数 *
    设备控制状态寄存器的地址 *
    配置结构寄存器地址 *
    * 指向部件统计块 *
    输入输出启动时间 *
    由设备驱动保留的 和 空间 *
    ；
    ；
}
```

块设备开关表

对于每类设备，根据其不同的物理特性，系统为其设置一套管理程序。包括打开、关闭、启动、中断处理等。为了便于管理，系统为每类设备设置一个数据结构，存放这些程序的入口地址。这些设备的数据结构组合起来，叫做设备开关表。其结构定义如下：

设备开关表

```
{
    打开子程序入口地址 *
    关闭子程序入口地址 *
    启动子程序入口地址 *
    中断处理子程序入口地址 *
    ...
};
```

磁盘和磁带在块设备开关表中占有前两项 `buf[0]` 和 `buf[1]`。当程序执行相应的系统调用时，系统转入相应的程序入口地址去执行。块设备的系统调用 `read()` 和 `write()` 通过调用打开和关闭例程来实现。`read()` 和 `write()` 调用通过高速缓冲管理实现。

空闲缓冲队列的管理

在块设备系统中，主要有以下几种队列：

①空闲缓冲队列

`bfree` 作为所有空闲缓冲区组成的空闲队列的头部变量，其中的 `bfree` 和 `bfree` 分别为指向空闲缓冲队列的队首和队尾的指针。

为了统一管理需要，除了空闲缓冲队列外，系统还设置一个空闲设备队列。其设备名为 `buf[0]`，用来记录那些与特定设备无关的空闲缓冲区。空闲设备队列头部变量也是 `bfree`，该队列的队首和队尾指针为 `bfree` 和 `bfree`。

`bfree` 用来记录空闲缓冲队列中的空闲缓冲区个数。`bfree` 中仅 `bfree` 标志有效。当进程申请空缓冲而空闲队列中无缓冲区可用时，设置此标志，睡眠等待。

系统初始化时，对缓冲池做初始化处理。所有缓冲区按序号从 `buf[0]` 到 `buf[NBUF-1]` 排在空闲队列中，按序号 `buf[0]` 到 `buf[NBUF-1]` 排在 `buf[0]` 队列中。`bfree` 中除两个连接两指针（空闲队列为 `bfree` 和 `bfree`，空闲设备队列为 `bfree` 和 `bfree`），`bfree` 和 `bfree` 初始化为 `0` 被赋值外，其他各项均为 `0`。如图 10-10 所示。

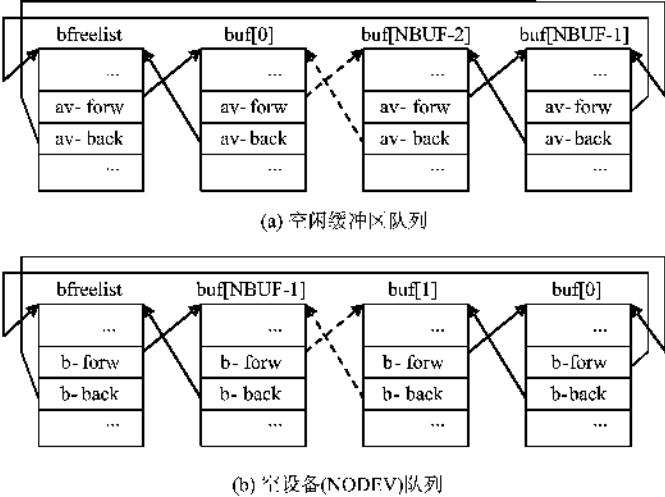


图 10-10 空闲缓冲队列和空闲设备队列的初始状态

②设备缓冲队列

为了便于管理，当缓冲区与某设备上的一个物理块相关时，将所有与同一设备相关的缓冲区排成队列，叫设备缓冲队列。其队列头部的结构定义如下：

设备缓冲队列

```

    缓冲队列长度++;
    缓冲队列首指针 = 缓冲队列尾指针;
} 缓冲队列;

```

其中，缓冲队列中仅用设备号标志有效。当进程希望使用这个队列而有进程在使用时，设置此标志，睡眠等待。缓冲队列首指针和缓冲队列尾指针分别是设备缓冲队列的队首和队尾指针。共设置 256 个设备队列，系统初始化时，每个设备队列均为空。

为了加速对缓冲区的查找，系统按缓冲区所在的设备号和块号计算的散列值不同，组成 256 个散列队列。每个散列队列是一个双向链，如图 8-1 所示。

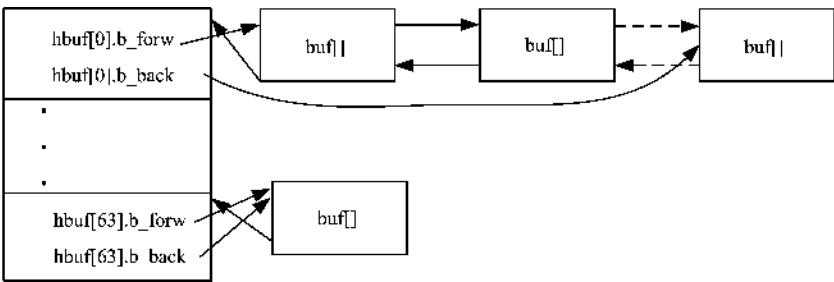


图 8-1 设备缓冲队列

操作系统采用下列公式计算各缓冲区所在的散列队列：

$$(\text{设备号} \times \text{块号}) \text{ 取 256 的 余 数}$$

其中 设备号 和 块号 表示该缓冲区对应的设备号和物理块号，设备号屏蔽码是一个屏蔽码，其值为 256 (即 0x100)。此式表示将设备号加上块号取 256 的模，结果作为 缓冲队列 数组的下标，确定设备缓冲队列的头部。这样，同一个块设备的各个缓冲区可以分布在不同队列中。不同设备的缓冲区均匀分布，使每个队列中只有 2 个缓冲区，加快了缓冲区的查找速度。

③ 设备 缓冲请求队列

当进程读写磁盘时，先申请一个缓冲区，填写传输所用的必要参数后，将该缓冲区排到设备请求队列中等待传输。每个设备对应一个 缓冲请求队列，且采用单向链连在一起。缓冲请求队列头部结构为 缓冲请求头 与磁盘设备表结构相同。

缓冲请求头与磁盘设备表结构

(1) 缓冲请求头管理算法

为了提高使用效率，系统按某种算法选择被重新分配的缓冲存储区。操作系统采取了缓冲淘汰算法 先进先出 (最近最少使用，缓冲淘汰算法)。在把一缓冲区分配给一个磁盘块后，内核就不让别的磁盘块再使用它，直到其他所有缓冲区最近都被使用过。

(2) 操作系统缓冲存储区的分配

当进程需要对某设备上的字符块进行处理时，先要为其分配一个主存缓冲区。实现分配主存缓冲区功能的函数是 分配主存缓冲区，这里参数 设备号 和 块号 分别是字符块所在设备号及物理块号，返回值是加了锁的缓冲控制块指针。其实现过程如下：

- ① 根据设备号和块号计算散列值。以该散列值作为设备队列头部的下标，在设备缓冲队列中搜索对应的缓冲区，查找是否有该设备号、块号相同的缓冲区。

设备物理设备中断处理程序

首先由设备驱动程序等待操作结束,然后调用设备驱动程序完成设备的中断处理。

设备驱动程序块设备读写技术

块设备读写技术主要由基本字符块的输入和预读操作两种方式。提供了基本读盘块的程序 `read()`,其中的参数代表主、次设备号。字符块预读程序 `readahead()`,其中的三个参数为块设备号、当前读的块号、预读块号,系统用同步方式读 `read()` 字符块号,以异步方式读 `readahead()` 字符块号。字符块输出程序 `write()`,其中的参数指向它所控制的需要写到块设备的缓存。

设备驱动程序块设备的程序结构

块设备程序结构分为缓冲管理层、设备驱动层和块设备读写管理层。

缓存管理层处于块设备的最底层,包括初始化程序、获取缓存和释放缓存的程序。设备驱动层处于块设备的中间层,包括从队列中请求输入输出的设备请求块、启动程序、块设备中断处理程序及同步操作、等待设备的中断处理程序。块读写管理层处于块设备的最上层,包括读、写、预读、延迟写、交换程序等。

操作系统中还有另外一种设备——字符设备,限于篇幅,就不再作介绍。

本章小结

设备管理是操作系统的重要组成部分,而且是最复杂的一部分。

可以从不同的角度对外部设备进行分类,比如可以从数据传输的角度分为块设备和字符设备,可以从资源分配的角度分为独占设备、共享设备和虚拟设备等。

设备控制器是一台设备的电子部分,在操作系统和外部设备进行数据交换的过程中发挥着重要的作用。

不同规模的计算机系统,其系统结构也有差异,通常分为总线型和通道型两种。外部设备和系统之间的数据传送控制方式主要有 3 种,分别是程序直接控制方式、中断控制方式、通道方式和通道方式。中断及其处理是计算机系统中非常重要的内容,本章详细介绍了中断的基本概念及其处理过程。

缓冲是得到广泛采用的平滑数据速率的方法。引入缓冲的目的是为了匹配设备和系统的处理速度,利用缓冲可以增强系统的处理能力和提高资源的利用率。本章详细介绍了有关缓冲一些基本原理。

根据设备的物理特性和管理上的方便、有效,通常采用的设备分配技术有 3 种:独占、共享和虚拟,其中虚拟设备系统就是典型的虚拟设备系统,并介绍了设备分配的原则和算法。

设备软件分为四层:中断处理程序、设备驱动程序、与设备无关的设备软件以及用户空间的设备软件。

磁盘系统是大多数计算机上的主要设备。可用的调度算法有 3 种:先来先服务、最短响应时间和优先级调度等。

最后,作为实例,介绍了操作系统中块设备管理的一些内容。

习题

1. 如何将独占型输入设备改造成可共享使用的虚设备？

2. 在设备管理中，何谓设备独立性？如何实现设备独立性？

3. 何谓虚拟设备？请说明 操作系统是如何实现虚拟设备的。

4. 何为设备虚拟？实现设备虚拟时所依赖的关键技术是什么？

5. 请图示实现 设备虚拟技术时的系统构成图，说明需要设置什么系统进程参与管理操作。设备虚拟技术的实质是什么？

6. 什么是中断？简单叙述中断的处理过程。

7. 打印机和磁盘在计算机系统中都是共享资源，当多个作业共享时有什么不同？

8. 轮转方式和中断方式有什么不同？

9. 磁盘是用来存放文件和数据的，因此可以说磁盘仅仅是用作文件系统的设备，这句话对吗？为什么？

10. 什么是缓冲？为什么要引入缓冲？

11. 系统中有哪些类型的设备？

12. 假设 输入设备和输出设备 能并行执行，且输入设备 和输出设备 的启动受 控制指令的控制。另外，输出设备 的启动还受输出缓冲是否装满输出数据的限制。只有装满输出数据，输出设备才能被启动。试描述中断处理方式下的 运作过程。

13. 假设管理缓冲区的 个队列分别为空白缓冲队列、输入缓冲队列、以及输出缓冲队列。试描述 过程。 和 分别用来把缓冲区 插入 队列和从 队列中取出缓冲区。试描述进程从任一队列中得到一个缓冲区的过程。 和 释放一个缓冲区 进入缓冲队列的过程。

14. 如磁盘的每个磁道分成九段，先有一个文件共有 9 个记录，每个记录的大小与段的长度相当，若磁盘转速为 3600 转/分，每读出一段后需要 10ms 的处理时间。若忽略其他辅助时间，试问：

(1) 如顺序存放这些记录时，读出该文件需多少时间？

(2) 采用合适的磁盘调度算法，画出各记录的存放位置，计算该文件的读出时间，并与 (1) 的结果进行比较。

15. 磁盘系统调度中，采用 调度扫描算法为任务队列 服务。试计算服务结束时，磁头共移动了几个磁道。假设磁头总在第 10 道至 100 道之间移动；开始服务时，磁头刚从 10 道移动到 100 道。

16. 磁盘请求的柱面以 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320 330 340 350 360 370 380 390 400 410 420 430 440 450 460 470 480 490 500 510 520 530 540 550 560 570 580 590 600 610 620 630 640 650 660 670 680 690 700 710 720 730 740 750 760 770 780 790 800 810 820 830 840 850 860 870 880 890 900 910 920 930 940 950 960 970 980 990 1000 1010 1020 1030 1040 1050 1060 1070 1080 1090 1100 1110 1120 1130 1140 1150 1160 1170 1180 1190 1200 1210 1220 1230 1240 1250 1260 1270 1280 1290 1300 1310 1320 1330 1340 1350 1360 1370 1380 1390 1400 1410 1420 1430 1440 1450 1460 1470 1480 1490 1500 1510 1520 1530 1540 1550 1560 1570 1580 1590 1600 1610 1620 1630 1640 1650 1660 1670 1680 1690 1700 1710 1720 1730 1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010 2020 2030 2040 2050 2060 2070 2080 2090 2100 2110 2120 2130 2140 2150 2160 2170 2180 2190 2200 2210 2220 2230 2240 2250 2260 2270 2280 2290 2300 2310 2320 2330 2340 2350 2360 2370 2380 2390 2400 2410 2420 2430 2440 2450 2460 2470 2480 2490 2500 2510 2520 2530 2540 2550 2560 2570 2580 2590 2600 2610 2620 2630 2640 2650 2660 2670 2680 2690 2700 2710 2720 2730 2740 2750 2760 2770 2780 2790 2800 2810 2820 2830 2840 2850 2860 2870 2880 2890 2900 2910 2920 2930 2940 2950 2960 2970 2980 2990 3000 3010 3020 3030 3040 3050 3060 3070 3080 3090 3100 3110 3120 3130 3140 3150 3160 3170 3180 3190 3200 3210 3220 3230 3240 3250 3260 3270 3280 3290 3300 3310 3320 3330 3340 3350 3360 3370 3380 3390 3400 3410 3420 3430 3440 3450 3460 3470 3480 3490 3500 3510 3520 3530 3540 3550 3560 3570 3580 3590 3600 3610 3620 3630 3640 3650 3660 3670 3680 3690 3700 3710 3720 3730 3740 3750 3760 3770 3780 3790 3800 3810 3820 3830 3840 3850 3860 3870 3880 3890 3900 3910 3920 3930 3940 3950 3960 3970 3980 3990 4000 4010 4020 4030 4040 4050 4060 4070 4080 4090 4100 4110 4120 4130 4140 4150 4160 4170 4180 4190 4200 4210 4220 4230 4240 4250 4260 4270 4280 4290 4300 4310 4320 4330 4340 4350 4360 4370 4380 4390 4400 4410 4420 4430 4440 4450 4460 4470 4480 4490 4500 4510 4520 4530 4540 4550 4560 4570 4580 4590 4600 4610 4620 4630 4640 4650 4660 4670 4680 4690 4700 4710 4720 4730 4740 4750 4760 4770 4780 4790 4800 4810 4820 4830 4840 4850 4860 4870 4880 4890 4900 4910 4920 4930 4940 4950 4960 4970 4980 4990 5000 5010 5020 5030 5040 5050 5060 5070 5080 5090 5100 5110 5120 5130 5140 5150 5160 5170 5180 5190 5200 5210 5220 5230 5240 5250 5260 5270 5280 5290 5300 5310 5320 5330 5340 5350 5360 5370 5380 5390 5400 5410 5420 5430 5440 5450 5460 5470 5480 5490 5500 5510 5520 5530 5540 5550 5560 5570 5580 5590 5600 5610 5620 5630 5640 5650 5660 5670 5680 5690 5700 5710 5720 5730 5740 5750 5760 5770 5780 5790 5800 5810 5820 5830 5840 5850 5860 5870 5880 5890 5900 5910 5920 5930 5940 5950 5960 5970 5980 5990 6000 6010 6020 6030 6040 6050 6060 6070 6080 6090 6100 6110 6120 6130 6140 6150 6160 6170 6180 6190 6200 6210 6220 6230 6240 6250 6260 6270 6280 6290 6300 6310 6320 6330 6340 6350 6360 6370 6380 6390 6400 6410 6420 6430 6440 6450 6460 6470 6480 6490 6500 6510 6520 6530 6540 6550 6560 6570 6580 6590 6600 6610 6620 6630 6640 6650 6660 6670 6680 6690 6700 6710 6720 6730 6740 6750 6760 6770 6780 6790 6800 6810 6820 6830 6840 6850 6860 6870 6880 6890 6900 6910 6920 6930 6940 6950 6960 6970 6980 6990 7000 7010 7020 7030 7040 7050 7060 7070 7080 7090 7100 7110 7120 7130 7140 7150 7160 7170 7180 7190 7200 7210 7220 7230 7240 7250 7260 7270 7280 7290 7300 7310 7320 7330 7340 7350 7360 7370 7380 7390 7400 7410 7420 7430 7440 7450 7460 7470 7480 7490 7500 7510 7520 7530 7540 7550 7560 7570 7580 7590 7600 7610 7620 7630 7640 7650 7660 7670 7680 7690 7700 7710 7720 7730 7740 7750 7760 7770 7780 7790 7800 7810 7820 7830 7840 7850 7860 7870 7880 7890 7900 7910 7920 7930 7940 7950 7960 7970 7980 7990 8000 8010 8020 8030 8040 8050 8060 8070 8080 8090 8100 8110 8120 8130 8140 8150 8160 8170 8180 8190 8200 8210 8220 8230 8240 8250 8260 8270 8280 8290 8300 8310 8320 8330 8340 8350 8360 8370 8380 8390 8400 8410 8420 8430 8440 8450 8460 8470 8480 8490 8500 8510 8520 8530 8540 8550 8560 8570 8580 8590 8600 8610 8620 8630 8640 8650 8660 8670 8680 8690 8700 8710 8720 8730 8740 8750 8760 8770 8780 8790 8800 8810 8820 8830 8840 8850 8860 8870 8880 8890 8900 8910 8920 8930 8940 8950 8960 8970 8980 8990 9000 9010 9020 9030 9040 9050 9060 9070 9080 9090 9100 9110 9120 9130 9140 9150 9160 9170 9180 9190 9200 9210 9220 9230 9240 9250 9260 9270 9280 9290 9300 9310 9320 9330 9340 9350 9360 9370 9380 9390 9400 9410 9420 9430 9440 9450 9460 9470 9480 9490 9500 9510 9520 9530 9540 9550 9560 9570 9580 9590 9600 9610 9620 9630 9640 9650 9660 9670 9680 9690 9700 9710 9720 9730 9740 9750 9760 9770 9780 9790 9800 9810 9820 9830 9840 9850 9860 9870 9880 9890 9900 9910 9920 9930 9940 9950 9960 9970 9980 9990 10000 10010 10020 10030 10040 10050 10060 10070 10080 10090 10100 10110 10120 10130 10140 10150 10160 10170 10180 10190 10200 10210 10220 10230 10240 10250 10260 10270 10280 10290 10300 10310 10320 10330 10340 10350 10360 10370 10380 10390 10400 10410 10420 10430 10440 10450 10460 10470 10480 10490 10500 10510 10520 10530 10540 10550 10560 10570 10580 10590 10600 10610 10620 10630 10640 10650 10660 10670 10680 10690 10700 10710 10720 10730 10740 10750 10760 10770 10780 10790 10800 10810 10820 10830 10840 10850 10860 10870 10880 10890 10900 10910 10920 10930 10940 10950 10960 10970 10980 10990 11000 11010 11020 11030 11040 11050 11060 11070 11080 11090 11100 11110 11120 11130 11140 11150 11160 11170 11180 11190 11200 11210 11220 11230 11240 11250 11260 11270 11280 11290 11300 11310 11320 11330 11340 11350 11360 11370 11380 11390 11400 11410 11420 11430 11440 11450 11460 11470 11480 11490 11500 11510 11520 11530 11540 11550 11560 11570 11580 11590 11600 11610 11620 11630 11640 11650 11660 11670 11680 11690 11700 11710 11720 11730 11740 11750 11760 11770 11780 11790 11800 11810 11820 11830 11840 11850 11860 11870 11880 11890 11900 11910 11920 11930 11940 11950 11960 11970 11980 11990 12000 12010 12020 12030 12040 12050 12060 12070 12080 12090 12100 12110 12120 12130 12140 12150 12160 12170 12180 12190 12200 12210 12220 12230 12240 12250 12260 12270 12280 12290 12300 12310 12320 12330 12340 12350 12360 12370 12380 12390 12400 12410 12420 12430 12440 12450 12460 12470 12480 12490 12500 12510 12520 12530 12540 12550 12560 12570 12580 12590 12600 12610 12620 12630 12640 12650 12660 12670 12680 12690 12700 12710 12720 12730 12740 12750 12760 12770 12780 12790 12800 12810 12820 12830 12840 12850 12860 12870 12880 12890 12900 12910 12920 12930 12940 12950 12960 12970 12980 12990 13000 13010 13020 13030 13040 13050 13060 13070 13080 13090 13100 13110 13120 13130 13140 13150 13160 13170 13180 13190 13200 13210 13220 13230 13240 13250 13260 13270 13280 13290 13300 13310 13320 13330 13340 13350 13360 13370 13380 13390 13400 13410 13420 13430 13440 13450 13460 13470 13480 13490 13500 13510 13520 13530 13540 13550 13560 13570 13580 13590 13600 13610 13620 13630 13640 13650 13660 13670 13680 13690 13700 13710 13720 13730 13740 13750 13760 13770 13780 13790 13800 13810 13820 13830 13840 13850 13860 13870 13880 13890 13900 13910 13920 13930 13940 13950 13960 13970 13980 13990 14000 14010 14020 14030 14040 14050 14060 14070 14080 14090 14100 14110 14120 14130 14140 14150 14160 14170 14180 14190 14200 14210 14220 14230 14240 14250 14260 14270 14280 14290 14300 14310 14320 14330 14340 14350 14360 14370 14380 14390 14400 14410 14420 14430 14440 14450 14460 14470 14480 14490 14500 14510 14520 14530 14540 14550 14560 14570 14580 14590 14600 14610 14620 14630 14640 14650 14660 14670 14680 14690 14700 14710 14720 14730 14740 14750 14760 14770 14780 14790 14800 14810 14820 14830 14840 14850 14860 14870 14880 14890 14900 14910 14920 14930 14940 14950 14960 14970 14980 14990 15000 15010 15020 15030 15040 15050 15060 15070 15080 15090 15100 15110 15120 15130 15140 15150 15160 15170 15180 15190 15200 15210 15220 15230 15240 15250 15260 15270 15280 15290 15300 15310 15320 15330 15340 15350 15360 15370 15380 15390 15400 15410 15420 15430 15440 15450 15460 15470 15480 15490 15500 15510 15520 15530 15540 15550 15560 15570 15580 15590 15600 15610 15620 15630 15640 15650 15660 15670 15680 15690 15700 15710 15720 15730 15740 15750 15760 15770 15780 15790 15800 15810 15820 15830 15840 15850 15860 15870 15880 15890 15900 15910 15920 15930 15940 15950 15960 15970 15980 15990 16000 16010 16020 16030 16040 16050 16060 16070 16080 16090 16100 16110 16120 16130 16140 16150 16160 16170 16180 16190 16200 16210 16220 16230 16240 16250 16260 16270 16280 16290 16300 16310 16320 16330 16340 16350 16360 16370 16380 16390 16400 16410 16420 16430 16440 16450 16460 16470 16480 16490 16500 16510 16520 16530 16540 16550 16560 16570 16580 16590 16600 16610 16620 16630 16640 16650 16660 16670 16680 16690 16700 16710 16720 16730 16740 16750 16760 16770 16780 16790 16800 16810 16820 16830 16840 16850 16860 16870 16880 16890 16900 16910 16920 16930 16940 16950 16960 16970 16980 16990 17000 17010 17020 17030 17040 17050 17060 17070 17080 17090 17100 17110 17120 17130 17140 17150 17160 17170 17180 17190 17200 17210 17220 17230 17240 17250 17260 17270 17280 17290 17300 17310 17320 17330 17340 17350 17360 17370 17380 17390 17400 17410 17420 17430 17440 17450 17460 17470 17480 17490 17500 17510 17520 17530 17540 17550 17560 17570 17580 17590 17600 17610 17620 17630 17640 17650 17660 17670 17680 17690 17700 17710 17720 17730 17740 17750 17760 17770 17780 17790 17800 17810 17820 17830 17840 17850 17860 17870 17880 17890 17900 17910 17920 17930 17940 17950 17960 17970 17980 17990 18000 18010 18020 18030 18040 18050 18060 18070 18080 18090 18100 18110 18120 18130 18140 18150 18160 18170 18180 18190 18200 18210 18220 18230 18240 18250 18260 18270 18280 18290 18300 18310 18320 18330 18340 18350 18360 18370 18380 18390 18400 18410 18420 18430 18440 18450 18460 18470 18480 18490 18500 18510 18520 18530 18540 18550 18560 18570 18580 18590 18600 18610 18620 18630 18640 18650 18660 18670 18680 18690 18700 18710 18720 18730 18740 18750 18760 18770 18780 18790 18800 18810 18820 18830 18840 18850 18860 18870 18880 18890 18900 18910 18920 18930 18940 18950 18960 18970 18980 18990 19000 19010 19020 19030 19040 19050 19060 19070 19080 19090 19100 19110 19120 19130 19140 19150 19160 19170 19180 19190 19200 19210 19220 19230 19240 19250 19260 19270 19280 19290 19300 19310 19320 19330 19340 19350 19360 19370 19380 19390 19400 19410 19420 19430 19440 19450 19460 19470 19480 19490 19500 19510 19520 19530 19540 19550 19560 19570 19580 19590 19600 19610 19620 19630 19640 19650 19660 19670 19680 19690 19700 19710 19720 19730 19740 19750 19760 19770 19780 19790 19800 19810 19820 19830 19840 19850 19860 19870 19880 19890 19900 19910 19920 19930 19940 19950 19960 19970 19980 19990 20000 20010 20020 20030 20040 20050 20060 20070 20080 20090 20100 20110 20120 20130 20140 20150 20160 20170 20180 20190 20200 20210 20220 20230 20240 20250 20260 20270 20280 20290 20300 20310 20320 20330 20340 20350 20360 20370 20380 20390 20400 20410 2

第 8 章 文件系统

计算机系统本质上可视为一个可以存储并处理大量的各式各样信息的信息加工系统。由于处理的信息量太大,内存容量有限,且不能长期保存,因此多数情况下它们以文件的形式存放在外存中,需要时可随时调入内存。

要实现用户提出的“按名存取”,操作系统必须解决文件如何在外存中存放,如何按照文件的名称检索到这个文件,如何对文件的内容进行更新,如何保证文件的共享和保密等问题。于是在操作系统中设计了文件管理功能,即构成一个文件系统,负责管理外存上的文件,并把对文件的存取、共享和保护等手段提供给用户。

本章着重讲述以下几方面的内容:

(1) 目录结构

(2) 用户组织文件的方式(逻辑结构)与存储文件时的组织方式(物理结构)

(3) 对文件存储空间——磁盘的管理

(4) 文件的共享与保护。

(5) 文件的系统调用

(6) 文件系统的基本模型

(7) 文件系统的主要结构和算法

8.1 目录结构

操作系统将大量文件存放于辅助存储设备中。所谓文件是指由创建者所定义并具有完整逻辑意义的一组相关信息的集合。为了能对这些文件实现有效的管理,必须对它们加以妥善管理,这主要是通过文件目录实现的。文件目录也是一种数据结构,用于表示系统中的文件及其物理地址,供检索时使用。因此对目录管理的要求是:操作系统实现对文件信息的“按名存取”,力求文件查找简便,提高对目录的检索速度,方便灵活地存取信息,允许文件重名,便于文件共享和保密。

8.1.1 一级目录结构

一级目录结构是最简单的目录结构,所有的文件都登记在同一个文件目录中,即在整个文件系统中只建立一张目录表,如图 8-1 所示,每个文件占据其中的一个目录项,目录项中包含文件名、文件扩展名、文件长度、文件类型、文件物理地址以及其他文件属性。

建立一个新文件时,系统首先检查存在的文件目录,确定该文件名是否重名,若不重名,就在文件目录中增加一个目录项,并把文件控制块的全部信息保存在该项中。访问一个文件时,系统首先根据文件名去查找文件目录,确定该文件是否存在,若存在,则执行相应的操作。删除一个文件时,先从目录中找到该文件的目录项,回收该文件所占用的存储空间,然

后清除该目录项。

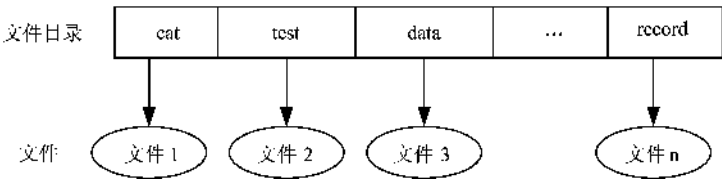


图 1-1-1 一级目录结构

一级目录通常按卷（可理解为一盘磁带，一个逻辑磁盘）构造，即把一卷中的全部文件形成一级目录表，保存在该卷的固定区域，使用时先将目录表读到主存。

当某卷的空间较大，文件数目太多时，目录结构中的目录项随之增加。这不仅给文件的检索带来困难，更严重的是多个用户的文件同时保留会带来“重名”问题。

一级目录结构管理简单，实现容易，能实现“按名存取”。但当系统中的文件数很多时，则查找时间长，且易发生重名问题，不便于实现文件共享。所以，一级目录结构一般只适应于单用户系统。

二级目录结构

为了解决一级目录结构存在的缺点，可以为每个用户建立一个独立的文件目录，于是就形成了二级目录结构。

如图 1-1-2 所示，在二级目录结构中，第一级为主文件目录，主文件目录以用户名为索引，对每个用户都设置一个指向用户文件目录的指针。第二级为用户文件目录，用户文件目录为本用户的每一个文件设置一个目录项。

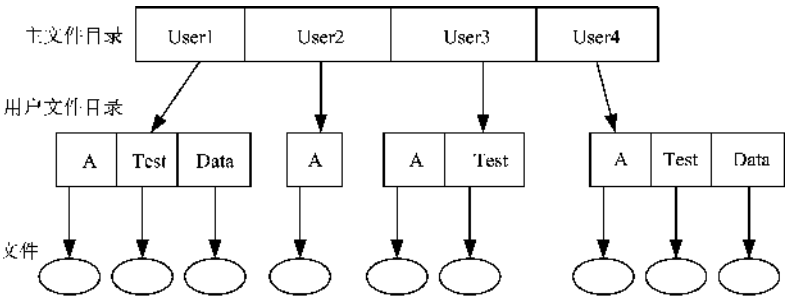


图 1-1-2 二级目录结构

当一个新的用户作业进入系统时，系统为该用户设置一个用户文件目录，且将其登记到主文件目录中。若用户不再需要自己的文件目录，可以请求系统管理员将它撤销。当用户有了文件后，可以根据自己的需要创建新文件。而此时，系统只需检查该用户的文件目录，判定是否重名。若重名，则重新命名文件；若不重名，则在文件目录中建立一个新目录项，将新文件名

及其属性填入目录项中。当用户要访问某个文件时，系统先从 **根目录** 中找出该用户的 **用户目录**，然后在 **用户目录** 中查找指定的文件。当用户删除一个文件，系统也需找到该用户 **用户目录**，再从中找出指定文件的目录项，回收该文件所占用的存储空间，并将该目录项删除。

因此，在二级目录结构中，每个用户都拥有自己的目录，按名查找一个文件的时间将会减少。另外，不同的用户也可以给文件取相同的名字，文件重名已经不再成为问题。

然而，若一个用户拥有的文件数越多，则在他的目录中查找一个文件所花费的时间就越长。另外，在二级目录结构中，用户无法对自己的文件进行再分类安排。所以，这种目录结构还是难以使用户感到满意。为此，引入了树形目录结构。

树形目录结构

若允许用户在自己的文件目录中根据不同类型的文件再建立子目录，则可将二级目录结构推广成多级目录结构。多级目录结构像一棵倒置的有根树，故称为树形目录结构。树形目录结构即是目录的层次结构。在图 2-1 中，用户 **用户1** 的子目录就有 3 层之多（注意，图中只是用字母表示文件或目录的所有者，并没有给它们分别取名字）。

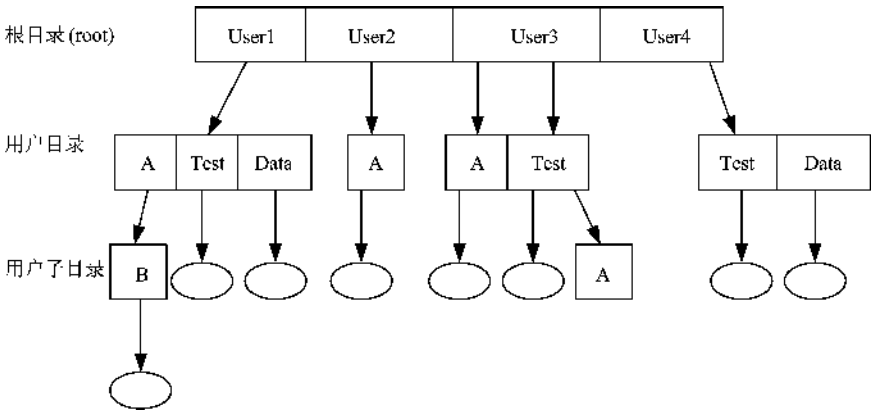


图 2-1 树形目录结构

在这一棵倒置的树中，第 1 层为根目录，第 2 层为用户目录，再往下是用户的子目录。另外，每一层目录中，既可以有子目录的目录项，也可以有具体文件的目录项。利用这种目录结构，用户可以根据需要，组织自己的目录层次。

利用树形结构组织文件系统时，为了能够明确地指定文件，不仅文件要有文件名，目录和子目录也都要有名字。从根目录出发到具体文件所经过的各层名字的组合，就构成了文件的“绝对路径名”。注意，文件的绝对路径名必须从根目录出发，且是惟一的，路径名中的每一个名字之间用分隔符隔开。在 **UNIX** 中，路径各部分之间是用“/”分隔；在 **MS-DOS** 中，路径各部分之间是用“\”分隔；在名为 **OS/2** 的操作系统中，路径各部分之间是用“.”分隔。不管采用哪种分隔符，凡是以分隔符打头的路径名必定是绝对路径名。

绝对路径名较长，使用不方便，一般采用文件的相对路径名。此时，用户可以指定一个目录作为当前目录（也称工作目录），从当前目录往下的文件的路径名，这时称为文件的相对路径名。因此，一个文件的相对路径名与当前所处的位置有关，它不是惟一的。若文件在当

前目录中，则存取文件时只要指出文件名就行，文件系统将在当前目录中寻找该文件。若文件不在当前目录中，但在当前目录的下级目录中，则可用相对路径名指定文件，文件系统就从当前目录开始沿着指定的路径查找该文件。例如：

绝对路径名 绝对路径名

相对路径名 相对路径名

都指向同一个文件，文件系统用相对路径名和绝对路径名均可找到名为 月的文件。显然，使用相对路径名可以减少查找文件所花费的时间。

树形目录结构的特点如下：

解决了重名问题

允许在不同的子目录中使用相同的名字命名文件或下级子目录。

层次清楚

由于采用分支结构，不同性质、不同用户的文件可以构成不同的子树，便于管理。

提高检索文件的速度

利用当前目录和相对路径不仅方便用户，而且系统从当前目录开始检索文件，缩短了检索路径，提高了检索速度。

能进行存取权限的控制

在子目录中可规定存取权限，检索文件时核对存取权限可以避免一个用户未经授权就存取另一个用户的文件，保证了用户文件的私有性，实现了对文件的保护和保密。

无环图目录结构

树形目录结构便于实现文件分类，但不便于实现文件共享。因为用户一般只使用其子目录下的所有文件，若要共享其他用户文件，则需绕到另外一个用户子目录上去访问。此时需提供提供一个共享文件的完全路径名，由系统从根目录开始检索，或用户将其“当前目录”指向另一用户的子目录上，以实现共享访问。

组织合理的目录结构对实现文件共享非常重要，因此引入一种无环图目录结构或文件，如图 所示。这种结构允许若干目录共同描述或共同指向被共享的子目录或文件。

无环图目录结构比一般简单的树结构更为灵活，但也更复杂。在无环图目录结构中，同一文件可能存在多条完全路径名，结果导致不同的文件名可能指向同一文件，这有点像程序设计中的别名问题。若试图遍历整个文件系统（以找出一个文件，积累所有文件的统计资料，或复制所有文件到辅助存储器中），则可能重复遍历图中某些节点（共享节点）。

怎样删除无环图目录结构中的共享节点呢？一种可能就是每当用户要删除一个文件时就将其去掉。但这样可能会留下指向现已不存在文件的一个悬挂指针。更严重的是，如果留下的文件指针包含实际磁盘地址，

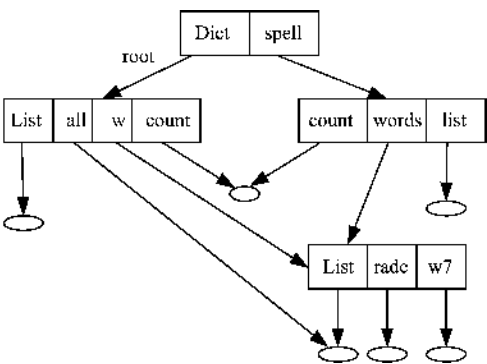


图 无环图目录结构

并且此空间又接着被其他文件使用，这些悬挂指针就可能指向其他文件。

一种可行的解决方法是，为图中每个节点设置一个访问计数器（或称共享计数器）。每当图中增加了一条对某个节点的共享链时，该节点的访问计数器加 1；每当需要删除某个节点时，该节点的访问计数器减 1。如果访问计数器为 0，则删除该节点，否则只删除共享链，保留原节点。

通用图目录结构

无环图目录结构虽有利于共享，但必须保证图中不存在环路。为此人们引入一种一般的通用目录结构，如图 9-15 所示。这种结构也是在树形目录结构的基础上增加共享链，允许图中出现环路。

通用图目录结构欲增加共享链很容易，但删除链时可能在目录结构中出现一些“废料”节点。当环路存在时，即使不再访问文件或目录，其访问数也不一定为零，这种现象是由于目录结构中存在自引用。因此，系统应定期对文件存储空间进行“废料”回收。回收的方法可以是，先遍历目录结构，并将遍历到的节点做标记。然后进行第二次遍历以将未加标记的节点删除，回收相应的空间。

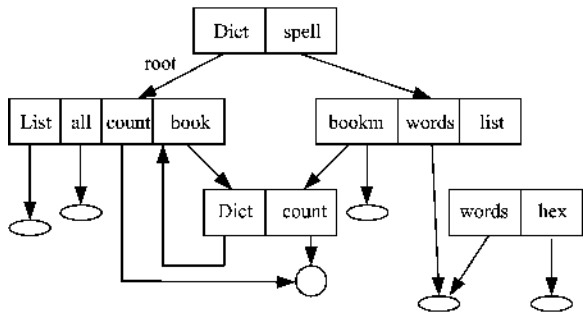


图 9-15 通用图目录结构

文件结构和存取方法

文件的物理结构

文件结构是指文件的构造方式。用户从使用的角度出发去组织文件，这种从用户观点出发所观察到的文件组织形式称为文件的“逻辑结构”；而系统从存储的角度出发去组织文件，即文件在外存上的存储组织方式称为文件的“物理结构”。

文件的物理结构对用户来说是不必关心的，但对文件系统来说却是至关重要，因为它直接影响存储空间的使用和检索文件信息的速度。用户按逻辑结构使用文件，文件系统按物理结构管理文件。因此，当用户请求读写文件时，文件系统必须实现文件的逻辑结构与相应的物理结构两者之间的转换，即进行逻辑结构与物理结构之间的映射。

文件的逻辑结构

用户按自己对信息的使用要求组织文件，由于这种文件独立于物理环境而构造。因此，把用户概念中的文件称为文件的逻辑结构，或称逻辑文件。一个文件的逻辑结构，就是该文件在用户面前呈现的结构形式。

文件逻辑结构的类型

文件的逻辑结构可分为两种：流式文件和记录式文件。

① 流式文件

流式文件是指用户对文件内的信息不再划分可独立的单位，整个文件是有序相关信息的集合，其基本单位是字节。流式文件以字符为操作对象，故流式文件管理简单，但查找文件中的基本信息单位比较困难。因此，适用于进行字符流的正文处理，如源程序文件和目标代码文件等。在 哉操作系统中，所有的文件都被看作流式文件。

② 记录式文件

记录式文件是指用户对文件内的信息按逻辑上独立的含义再划分信息单位，每个单位成为一个逻辑记录（简称记录）。于是，一个逻辑文件由若干个逻辑记录组成，称为记录式文件。在这种文件中，用户为每个记录顺序编号，称为“记录号”。记录号一般从 园开始，因此有记录 园 记录 员 记录 圆 记录 灶 出现在用户文件中的记录称为“逻辑记录”。每个记录由若干个数据项组成。图 怨给出了一个具体文件的逻辑结构形式，它的每一个记录包含：“学号”、“姓名”、“班级”和“各科成绩”（其中又分“计算机网络”、“数据结构”、“操作系统”等课程）等数据项。

逻辑记录是记录式文件中可以独立存取的最小信息单位。当用户请求文件系统读出一个逻辑记录后，用户可以对逻辑记录中的各个数据项进行处理。为了能准确快速地存取逻辑记录，必须要有一个数据项能够惟一地标识记录，以便对记录加以区分。文件中的这种数据项被称为主关键字或主键，如图 怨中的“学号”就是该文件的主关键字。要查找文件中的某个记录时，只要按主关键字去搜索。记录中的其他项被称为次关键字（或次键）。利用次键去查找记录，可以对文件中的记录进行再分类。如用“操作系统 越”的条件去搜索图 怨，则会得到一个记录，即王浩的操作系统成绩是 怨分。

记录号	学号	姓名	班级	各科成绩			
				计算机网络	数据结构	操作系统	...
园	园园园园	张宏	网通 园园园	愿	愿	苑	...
员	园园园园	黎明	网通 园园园	苑	愿	怨	...
圆	园园园园	王浩	电商 园园园	愿	怨	愿	...
...	...	...	...	...	...	...	...

图 怨记录式文件

圆 文件组织

文件的组织形式非常多，下面主要介绍 缘种基本组织。实际中使用的大多数结构有时是这几类之一，有时是它们的组合。

① 堆

堆是最简单的文件组织形式，如图 怨所示。数据按它们到达的顺序被排列，每个记录由一串数据组成。记录可以有不同域，或者域相似但顺序不同。每个域都包括域名和值。每个域的长度由定界符隐式地指定，或明确地包含在一个子域中，或者是该域类型的默认长度。由于堆文件没有结构，因而对记录的访问是通过穷举搜索的方式。

当数据在处理前排列并存储时，或者当数据难以组织时，会用到堆文件。当保存的数据结构和大小不同时，堆文件空间使用情况很好，能较好地用于穷举搜索，且易于修改。但除了这些受限制的使用，堆文件对大多数应用都是不适应的。

② 顺序文件

顺序文件是最常用的文件组织形式。在这类文件中，每个记录都使用一种固定的格式。所有记录都具有相同的长度，并且有相同数目、长度固定的域按特定的顺序组成，如图 3-10 所示。由于每个域的长度和位置已知，因此只需要保存各个域的值，每个域的域名和长度是该文件结构的属性。

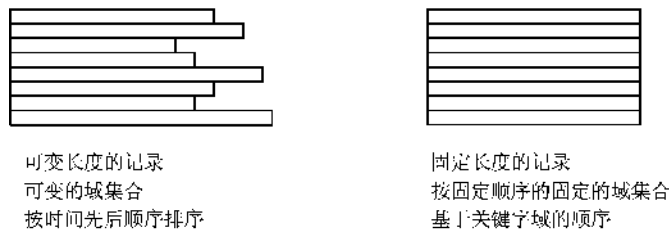


图 3-10 堆文件和顺序文件的组织

一个特殊的域，通常是每个记录的第一个域，称作关键字域。关键字域唯一地标识这个记录。不同记录的关键字值是不同的。此外，记录按关键字顺序存储，文本关键字按字母顺序，数字关键字按数字顺序。

顺序文件适应于批量存取。顺序文件在批量存取时，它的存取效率是所有逻辑文件中最高的；另外，只有顺序文件才能存储在磁带上，并且有效地工作。

对于涉及查询或更新记录的交互式应用，顺序文件表现出来的性能较差，尤其是当文件较大时，情况更为严重。另外，顺序文件增加或删除一个记录比较困难。

③ 索引文件

索引文件是为文件建立一张索引表，对主文件中的每个记录，在索引表中设有一个相应表项，用于记录该记录的长度及指向该记录的记录指针（指向该记录在逻辑地址空间的首址），如图 3-11 所示。由于索引表是按记录排序的，因此，索引表本身是一个顺序文件。

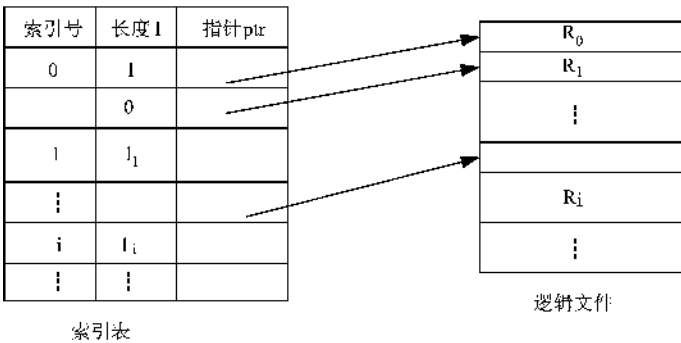


图 3-11 索引文件的组织

在对索引文件进行检索时，可根据用户提供的关键字，利用折半查找法去检索索引表，从中找到相应的表项；然后利用该表项中给出的指针值，访问所需的记录。当在主文件中增

加一个新记录时，索引文件必须全部更新。

由于索引文件有较快的检索速度，故它主要用于对信息的及时性要求比较严格且很少会对所有数据进行处理的应用程序中，如航空公司订票系统和商品库存控制系统。另外，索引文件除了有主文件外，还需配置一张索引表，而且每个记录都要有一个索引项，因此提高了存储费用。

④ 索引顺序文件

索引顺序文件是一种常见的逻辑文件形式，它有效地克服了顺序文件的缺点，而且付出的代价不算太大。它是顺序文件和索引文件相结合的产物。它将顺序文件中的所有记录分为若干个组，并为顺序文件建立一张索引表，在索引表中为每组中的第一个记录建立一个索引项，其中含有该记录的键值和指向该记录的指针，如图 9-5 所示。

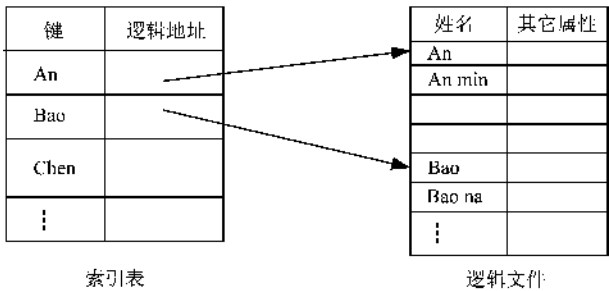


图 9-5 索引顺序文件的组织

在对索引顺序文件进行检索时，先利用用户所提供的关键字以及某种查找算法，去检索索引表，找到该记录所在记录组中第一个记录的表项，从中得到该记录组第一个记录在主文件中的位置，然后，利用顺序查找法去查找主文件，从中找到所要求的记录。

⑤ 直接文件（或散列文件）

直接文件（或散列文件）有开发直接访问磁盘中任何一个地址已知块的能力。同顺序文件和索引顺序文件一样，直接文件使用基于关键字的散列。直接文件常用在要求快速访问，且记录的长度是固定的情况，通常一次只访问一条记录，如目录、调度和名字列表。

9.2 存储设备与文件的物理结构

由逻辑地址到物理地址的映射与文件的存储方式（即文件的物理结构）密切相关。另外，文件系统采用哪种存取方法和逻辑结构，实际上与物理存储介质有关。下面介绍文件的物理结构和常用的文件存储设备。

（1）文件存储设备

目前常用的存储设备主要有磁带、磁盘、光盘等。其中磁盘又分为硬盘和软盘。存储设备的特性决定文件的存取设备和方法。因此，下面主要介绍以磁带为代表的顺序存储设备和以磁盘为代表的直接存储设备的特性及有关存取方法。

① 顺序存取设备

磁带是一种适合顺序存取的存储设备，也是最典型的顺序存取设备。它只有在前面的物理块被存取访问过后，才能存取后续物理块的内容。为了在存取一个物理块时，让磁带机提

前加速和不停止在下一个物理块的位置上，则必须在磁带的相邻物理块之间设计一个间隙将它们隔开，如图 9-1-10 所示。

磁带设备的存取速度（或信息传输率）与信息密度（字符数/英寸）、磁带带速（英寸/秒）和块间间隙有关。若带速高，信息密度大，且块间间隙小，则磁带存取速度和数据传输率高，反之亦然。

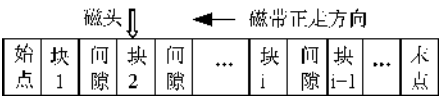


图 9-1-10 磁带存储结构

另外，磁带上的块不是用地址标识，而是由它在磁带上的物理位置来识别。若想按随机方式或按键存取方式存取磁带上的某一块的信息，则必须让磁头先定位到磁带的始端，然后才能正确定位到指定块的位置。这样，花费在定位上的时间就非常多，影响系统效率，故磁带上的文件很少采用随机存取方式。然而，磁带存取设备具有容量大，顺序存取方式时存取速度高等优点。所以，磁带存取设备仍获得了较为广泛的应用。

② 直接存取设备

磁盘是一种可按指定的块地址进行信息存取的设备，也是最典型的直接存取设备。磁盘上的每一块都有确定的位置和一个惟一的地址，地址用“柱面号、磁头号、扇区号”三个参数表示。磁盘设备能根据给定的地址带动读写磁头到达指定柱面，再让指定的磁头存取指定扇区上的信息。因此，磁盘设备能随时读写任何一块的信息，具有随机存取的功能。于是，对存储在磁盘上的文件，既可采用顺序存取方式，又可采用随机存取方式。

（四）文件的物理结构

文件在外存组织结构称为文件的物理结构。事实上，由于文件的物理结构决定了文件信息在存储设备上的存储位置，因此，文件信息的逻辑块号（逻辑地址）到物理块号（物理地址）的变换也是由文件的物理结构决定的。常用的文件物理结构有三种，即文件的顺序结构、链接结构和索引结构（或连续文件、串联文件和索引文件）。

① 顺序结构——连续文件

若逻辑文件的信息在外存中连续存储，则称文件的物理结构为顺序结构，或连续文件。图 9-1-11 给出了连续文件结构的图形说明。在图 9-1-11 中，一个逻辑块号为 0 的文件的物理块号依次为 14、15、16、17、...

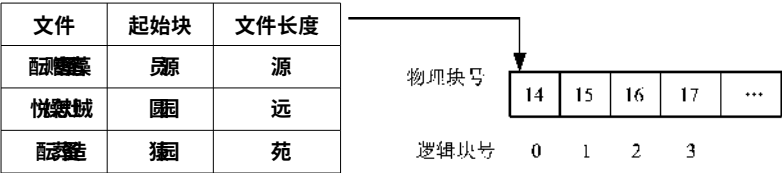


图 9-1-11 连续文件结构

这种分配方式保证了逻辑文件中的记录顺序与存储器中文件占用盘块的顺序的一致性。为了使系统能找到文件存放的地址，只需在目录项的“文件物理地址”字段中，记录该文件第一个记录所在的盘块号和文件长度（以盘块为单位）。图 9-1-12 中假定了记录与盘块的大小

相同。

同内存的动态分区分配一样，随着文件建立时空间的分配和文件删除时空间的回收，将使磁盘空间被分割成许多小块，这些较小的连续区难以用来存储文件，因此就称为外存的碎片。同样也可以用紧凑的方法，将盘上所有的文件紧靠在一起，把所有的碎片拼接成一大片连续的存储空间。

连续文件结构的特点是若已知文件在文件存储设备上的起址和文件长度，则能很快地进行存取。因此它的主要优点：一是顺序访问容易。访问一个占有连续空间的文件，非常容易。系统可从目录中找到该顺序文件所在的第一个盘块号，于是开始顺序地、逐个盘块地往下读。同时它也支持直接存取。二是顺序访问速度快。因为连续文件是连续存放在相邻的物理块中，它所占用的盘块可能是位于一条或几条相邻的磁道上，此时磁头移动的距离最少，因此对连续文件的访问速度是几种方式中最高的。

由于连续文件要求存放在连续的物理块中，因此，它的主要缺点：一是必须已知文件的最大长度，否则操作系统无法确定要为其开辟多少磁盘空间；二是与存储管理一样，会造成磁盘碎片，因为有一些小的磁盘块连续区满足不了用户作业的存储需求，则不能分配出去；三是若一个顺序文件需要扩展，则一定要有与其相邻的空闲磁盘块，然而与其相邻的磁盘块很可能被其他文件占用，从而顺序文件的扩展受到影响。因此，连续文件结构不宜用来存放用户文件、数据库文件等经常被修改的文件。

② 链接结构——串联文件

克服连续文件缺点的办法之一是采用串联文件结构。串联文件结构用非连续的物理块来存放文件信息。这些非连续的物理块之间没有顺序关系，然而每一块都包含一个指针，指向与它链接的下一块所在的位置，最后一块的指针为“原”，表示文件结束，如图 9.1.1 给出了串联文件的物理结构。在图 9.1.1 中，由首块指针指出 1 号文件是从第 14 块开始存放；由第 14 块中的指针“20”表明信息存放的下一块是第 20 块；由第 20 块中的指针“8”表明信息存放的下一块是第 8 块；最后，第 8 块中的指针是“原”，表示文件存放结束。于是，从首块指针出发，顺着指针，就能够找到该文件的所有记录。这个由第 14、20、8 块组成的文件，就是用户文件 1 号的物理结构，且是链接结构，它就是文件 1 号的串联文件。采用链接结构来存储文件，最大的好处是能够利用每一个存储块，不会因为磁盘碎片而浪费存储空间。然而指针要占用字节，因此每个磁盘块存储数据的字节数不再是 1024 的幂，从而降低了系统的运行效率和减少了存储空间。

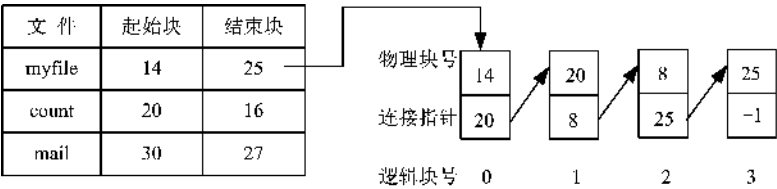


图 9.1.1 串联文件结构

由于串联文件结构只能按队列中的串联指针顺序搜索，因此，串联文件结构的搜索效率较低。另外，在存取文件时，若某个指针丢失或被破坏，则错误的指针可能指向其他文件而导致混乱。串联文件一般只适应于逻辑上连续的文件，且存取方法应该是顺序存取。否则，

为读取某个信息块而造成磁头大幅度的移动将花去较多的时间。因此，串联文件结构不适宜随机存取。

③ 索引结构——索引文件

若把逻辑上连续的用户文件信息散存于外存的各物理块中，则系统应为每个文件建立一张索引表，表中按逻辑记录存放的物理块顺序记录这些物理块号，则此时所涉及的物理块，就是这个用户文件的物理结构。由于这些物理块是不连续的，逻辑文件信息的连续性是通过索引表中记录的物理块的块号反映出来，因此把这个文件的物理结构称为索引结构，或索引文件，如图 8-1 给出了索引文件的物理结构。在图 8-1 中，索引表记录了依次分配给它的块的顺序与实际物理块的块号，也就是最先分配给它的第 0 块，实际是第 14 块；接着分配给它的第 1 块，实际是第 20 块，如此等等。在此，索引表的作用有点像页式存储管理中的页表，通过逻辑记录号，可以知道它应该位于第几块，由它所在的逻辑块，去查索引表，就知道它的物理块。这个由第 14、20、8、25 块组成的文件，就是文件 myfile 的物理结构，且是索引结构，它也是文件 myfile 的索引文件。

文件的索引结构实际上就是把链接结构中的指针取出来集中存放在一起，这样它既能够完全利用每一个存储块的最大存储量，又保持物理块为 1 的幂，从而克服了链接结构的缺点。

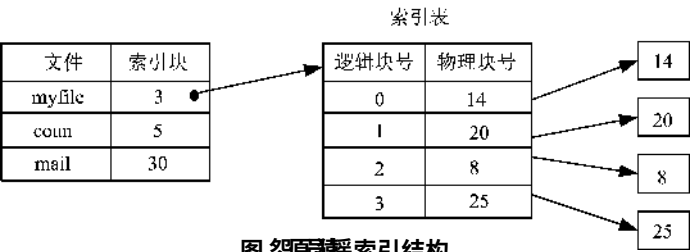


图 8-1 索引结构

索引文件支持直接访问。当要读文件的第 i 个块时，可以方便地直接从索引表中找到第 i 个盘块的盘块号。另外，索引文件也不会产生外部碎片。当文件较大时，索引文件无疑要优于链接文件。

索引文件的主要问题是：可能要花费较多的外存空间。每当建立一个文件时，便须为之分配一个索引块，将分配给该文件的所有盘块号记录于其中。而一般情况是中、小型文件居多，甚至不少文件只需 1~10 个盘块，此时若采用链接文件，只需设置 1~10 个指针。若采用索引文件，则同样仍需为之分配一个索引块。因此，对于小文件采用索引文件时，其索引块的利用率将是极低的。

当文件很大时，文件索引表比较大。若索引表的大小超过了一个物理块，则必须像处理其他文件的存放那样决定索引表的物理存放方式。但这不利于索引表的动态增加。索引表也可按串联方式存放，这却增加了存放索引表的时间开销。一种较好的解决方法是采用多级索引，即索引表所指的物理块中存放的不是文件信息，而是装有这些信息的物理块地址。于是，若一个物理块可装下 n 个物理块地址，则经过二级索引，可寻址的文件长度将变为 n^2 块。若文件长度大于 n^2 块，则可用三级索引、四级索引等方式，其原理如图 8-2 所示。

不过，大多数文件不需要进行多级索引。因为这些文件所占用的物理块数的块号可以放在一个物理块内。若对这些文件采用多级索引，则显然会降低文件的存取速度。因此，在实际系统中，总是把索引表的头几项设计成直接寻址方式，即这几项所指的物理块中存放的是

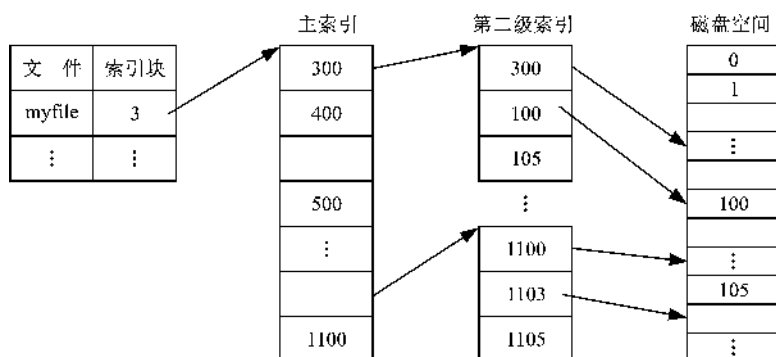


图 3-1-1 怨原源多级索引结构

文件信息，而索引表的后几项设计成多级索引，即间接寻址方式。在文件较小时，就可利用直接寻址方式找到物理块号而节省存取时间。

索引结构不仅适应于顺序存取，也适应于随机存取。但是它的缺点是：

- ① 增加了存储空间开销。

② 在存取文件时需要至少访问存储器二次以上。一次是访问索引表，另一次根据索引表提供的物理块号访问文件信息。由于文件在存储设备的访问速度较慢，因此，若把索引表存放在存储设备上，则将大大降低文件的存取速度。一种改进的方法是，当对某个文件进行操作之前，系统预先把索引表放入内存。因此，文件的存取就可直接在内存通过索引表确定物理地址块号，从而使得访问磁盘的动作只需要一次。

怨園園揺存取方法

存取文件信息的方法有多种，怎样选择合适的存取方法是设计中的一个重要问题。下面讨论几种常用的存取方法。

员爰顺序存取法

按照文件记录的排列次序一个接一个地存取,即为了存取第 i 个记录,必须先通过记录员到记录第 i 号。这种存取方法称为文件的顺序存取,它是基于文件的磁带模式的。

由于磁带机的物理特性，文件只能采用顺序结构在其上存放，也只能采用顺序存取的方式对文件进行访问。对于磁盘，文件保存可以采用顺序结构、链接结构和索引结构。

圆媛随机存取法

随机存取法是基于文件的磁盘模式的。为实现随机存取，文件被看作是编了号的块或记录的序列。可以以任何次序存取文件中的记录，无须涉及它前面的记录。这种存取方式对很多应用程序是必须具有的，比如数据库系统。如果乘客打电话预订某个航班的机票，订票程序必须能根据用户给出的航班，直接存取该航班的记录，而不必先读出在它前面的各个航班记录。

对磁带机上的文件，不宜采用随机存取的方式进行访问。对磁盘上的文件，如果该文件使用的是链接结构（也就是串联文件），那么也不宜采用随机存取的方式进行访问。图 9-1-1 给出了存储设备、存储结构以及存取方式之间的关系。

存储设备	磁摇摇摇摇盘			磁摇带
存储结构	连续文件	串联文件	索引文件	连续文件
存取方式	顺序, 随机	顺摇序	顺序, 随机	顺摇序

图 1-1-1 存储结构、存储设备及存取方式的关系

猿援其他存取方法

另外一些方法是建立在随机存取方法的基础上，一般包含文件的索引结构，索引结构含有指向不同块的指针。为寻找一个文件的记录，首先查询索引，接着使用索引指针直接存取所需记录。

当有大量文件时，索引文件本身可能太大，而不便于存放到内存。一种解决办法是创建一个索引文件的索引（称为辅索引），辅索引文件指向主索引文件，而主索引文件指向实际的数据项。

怨聲文件存储空间管理

在外存中，一个文件是由许多块组成的，文件系统负责给文件分配块。首先，外存中必须有足够的空间分配给文件；其次，必须知道可以分配哪些空间。很显然，这两个任务是相关的。即文件分配采用的方法可能会影响自由空间的管理方法。另外，文件结构和分配策略也是相互影响的。

怨癡焉揺文件存储空间分配

文件分配涉及以下几个问题：一是当创建一个新文件时，是否一次性地分配它所希望的最大空间；二是在分配文件时，分区的大小应该是多大；三是使用哪种文件结构。下面依次分析这几个问题。

资源预分配与动态分配

预分配策略要求在发出创建新文件的请求时就申明文件的最大空间。如程序编译、产生摘要数据文件或从另一个系统中通过通信网络传送文件，一般都可以可靠地估计文件的最大空间。然而对许多应用程序，不能可靠地估计文件的最大空间，就很难实现预分配策略。通常，用户和应用程序会多估计一些文件的大小，以避免分配的空间不够用，但这显然浪费外存空间。因此，使用动态分配策略就会好一些，动态分配策略只有在文件需要时才给文件分配空间。

圆暖分区大小

分区大小有两种：一种是可变的大连续分区，一般一个分区可由多个磁盘块组成，且每个分区的磁盘块数不同。它可以提供较好的性能，大小可变避免了浪费，并且文件分配表比较小，然而这将导致空间很难再利用；另一种是固定大小的分区，所有分区所含磁盘块数相等，小的固定分区提供了更大的灵活性，然而为了分配，它们将需要较大的表或更复杂的结构。

每一种分区方式都适用于预分配和动态分配。对于可变的、大规模连续分区，一个文件被预分配给一组连续的块，这就消除了对文件分配表的需求，它所需要的仅仅是指向第一块

的指针和分配的块数目。而对于固定大小分区形式，一次性地分配所需要的所有块，该文件的文件分配表将保持固定大小。

对于可变大小的分区，可以使用首次适应分配法、最佳适应分配法和最近适应分配法。很难评价哪种策略最好，因为它还和文件的类型、文件访问的权限、多道程序的程度、系统中的其他性能因素、磁盘缓存、磁盘调度等有关。另外，可变大小的分区存在空闲空间的碎片问题。

文件的分配方法

文件分配通常使用三种方法：连续、链式和索引，三种方法的区别如图 9-11 所示。

	连续	链式	索引	
是否预分配	需要	可能	可能	
分区大小固定还是可变	可变	固定块	固定块	可变
分区大小	大	小	小	中等
分配频率	一次	低到高	高	低
分配需要的时间	中等	长	短	中等
文件分配表的大小	一个入口	一个入口	大	中等

图 9-11 文件分配方法的比较

存储空间的管理

不仅分配给文件的空间需要管理，没有分配给任何文件的空间（空闲空间）也必须管理。为实现前面描述的分配方法，必须知道磁盘中的哪些块是可用的。因此，除了文件分配表以外，还需要一个磁盘分配表。下面介绍几种空闲空间的管理技术。

空闲表法

系统为每个磁盘建立一张空闲区表，表中的每一个表目记录磁盘空间中的一个连续空闲盘区的信息，比如该空闲盘区的起始空闲块号、连续的空闲块个数等信息。再将所有空闲区按其起始盘块号递增的次序排列，如图 9-12 所示，称此表为“空闲区表”。

任何时刻，空闲区表都记录着当前磁盘空间内可以使用的所有空闲块信息。空闲表法属于连续分配方式，空闲区的分配与内存的动态分配类似，同样是采用首次适应算法、最佳适应算法、最坏适应算法等分配方法。当创建一个新文件时，根据文件的长度查找该表，直到找到第一个其大小能满足要求的空闲区，再将该空闲区分配给用户（进程），同时修改空闲区表。系统在对用户所释放的存储空间进行回收时，如删除一个文件，系统也采取类似于内存回收的方法，即考虑回收区是否与空闲表中插入点的前区和后区相邻接，对相邻接者应予以合并。若无邻接，则在空闲区表中，增加一个新表项，并填写该文件原来占用的连续存储空间信息（如被释放的块号、长度以及第一块块号）。

显然，内存管理的有关空闲连续区分配和释放算法，只要稍加修改就可用于磁盘空闲区的分配与回收。在内存分配上，虽然很少采用连续分配方式，然而在外存的管理中，由于它具有较高的分配速度，可减少访问磁盘的频率，因此，它在诸多分配方式中仍占有一席之地。例如，对换方式的对换空间，一般采用连续分配方式。对于文件系统，在文件较小（员- 源个盘块）时，则采用连续分配方式，为文件分配相邻接的几个盘块；当文件较大时，仍采用离散分配方式。

空闲表法适用于连续文件结构的文件存储区的分配与回收。

空闲链表法

空闲链表法是把所有空闲区连接在一起构成空闲链，分配空间时，从链中取出空闲块；归还空间时，把归还块加入到链中。根据构成链元素的不同，分为空闲块链和空闲区链。

（员）空闲块链。它是以盘块为单位构成链。即在磁盘的每一个空闲块中设置一个指针，指向另一个磁盘空闲块，从而所有的空闲块形成一个链表，称为“空闲块链”。

若用户创建文件而申请分配存储空间时，系统就从链首开始一块一块地摘下进行分配；如果释放存储块，则把释放的块从链尾插入。当然，在申请时必须修改链首指针；无论是申请还是释放，必须调整相邻空闲块中的指针。

（圆）空闲区链。它是以空闲盘区（每个盘区可包含若干个盘块）为单位构成链。即在磁盘的每一个盘区中设置一个指针，指向另一个空闲盘区，另外每个盘区还要有指明本盘区大小（盘块数）的信息，从而把所有的空闲盘区形成一个链表，称为“空闲区链”。分配盘区的方法与内存的动态分区分配类似，通常采用首次适应法，这时需要调整指针和长度。在回收盘区时，同样也要将回收区与相邻接的空闲盘区合并。

位示图法

由于磁盘被分块后，每一块的大小相同，块数固定，于是可以采用位示图法来指示磁盘空间的使用情况。为所要管理的磁盘设置一张位示图，位示图的大小由磁盘的总块数决定。位示图中的每个二进制位与一个磁盘块对应，某位值若为“员”，表示所对应的块已经被使用；若为“园”，表示所对应的块是空闲块。如有一个磁盘，共有 员园园个柱面（编号为 园- 怨，每个柱面有 愿个磁道（编号为 园- 苑，注意，这也就是磁头编号），每个磁道分成 愿个扇区（编号为 园~ 苑，那么，整个磁盘空间磁盘块的总数为：愿伊愿伊员园园= 愿园园园（块）。若用字长为 愿位的字来构造位示图，那么共需要 员园园个字，如图 怨所示。

	园位	员位	圆位	...	远位	近位
第 园字	园园	园园	园园	...	园园	园园
第 员字	园园	园园	园园	...	园园	园园
第 圆字	园园	园园	园园	...	园园	园园
...	...	...	...	...	...	...
第 怨字	园园	园园	园园	...	园园	园园

图 怨 位示图

当请求分配存储空间时,系统顺序扫描位示图,从中找出一个或一组值为“0”的二进制位,转换成所对应的块并进行分配,然后修改位示图,使其对应的位置为1。回收存储空间时,将回收盘块的盘块号转换成位示图中的字号与位号,再修改位示图,使其对应的位置为0。因此,在申请存储空间时,就有一个“已知字号、位号,计算对应块号(即柱面号、磁头号、扇区号)”的问题。在回收盘块时,就有一个“已知柱面号、磁头号和扇区号,计算对应字号和位号”的问题。

“相对块号”是指从0开始,按柱面和盘面(即磁头)的顺序对磁盘块进行统一编号。即第0柱面第0盘面上的块号是0,接着,第0柱面第1盘面上的块号是1。由于第0柱面上共有10个磁盘块,故编号为0~9。第1柱面上磁盘块的编号为10~19。整个磁盘块的编号是0~199。这就是该磁盘块的“相对块号”。因此,位示图中第100字的第10位对应的相对块号为:相对块号=100×10+10=1010。

在申请磁盘空间时,根据查到的值为“0”的那位的字号和位号,可以计算出这一位所对应的相对块号,然后再求出具体的柱面号、磁头号和扇区号,如图10-1所示,假设字号=100,相对块号=1010,位号=10,则“字号=100,位号=10”就可计算出“柱面号、磁头号和扇区号”。

柱面号=100/10=10

磁头号=1010/10=101

扇区号=1010%10=10

于是,文件信息就可按确切的地址存放到找到的磁盘块上。

在归还磁盘块时,根据释放块的“柱面号、磁头号和扇区号”,先计算出该块的相对块号,然后再求出它在位示图中的字号和位号。

相对块号=柱面号×10+磁头号×10+扇区号

字号=相对块号/100

位号=相对块号%100

这种方法的主要优点是很容易从位示图中找到一个或一组相邻接的空闲盘块。如在文件分配时,要找10个相邻接的空闲盘块,则只需在位示图中找出10个其值连续为“0”的位即可。另外,由于位示图很小,占用空间少,于是可将它保存在内存中。因此,在每次进行盘区分配时,无需首先把盘区分配表读入内存,从而节省了许多磁盘的启动操作。

成组链接法

空闲表法和空闲链表法,都不适应于大型文件系统,因为这会使空闲表或空闲链表太长。若把空闲块分成若干组,把指向一组中各空闲块的指针集中在一起,这样既可方便查找,又可减少为修改指针而启动磁盘的次数。这种改进的方法被称为“成组链接”法。操作系统采用的是成组链接法,在10-2节里会详细介绍它的实现过程。

文件的共享和保护

在现代计算机系统中,必须提供文件共享手段,即系统允许多个用户(进程)共享同一份文件。因此,在系统中只需保留该共享文件的一份副本。若系统不能提供文件共享功能,则凡是需要该文件的用户,都必须各自备有此文件的副本,显然这会造成对存储空间的极大浪费。随着计算机技术的发展,文件共享的范围也在不断扩大,从单机系统中的共享,扩展为

多机系统的共享,进而又扩展为计算机网络范围的共享,甚至实现全球的文件共享。

8.1 文件共享

文件共享是指系统允许多个用户(进程)共同使用同一个文件。文件共享既可避免系统复制文件的工作,又可节省文件占用的存储空间等。在允许文件共享的系统中,必须对共享文件进行管理,共享文件的访问有两种:

8.1.1 不允许同时访问

任何时刻只允许一个用户(进程)访问共享文件,即不允许两个或两个以上的用户同时打开一个文件。当一个用户打开共享文件后,必须等这个文件使用结束并关闭后,才允许其他用户(进程)打开该文件。

8.1.2 允许同时访问

若允许多个用户(进程)访问一个共享文件,则文件管理系统必须强加一些规范。一种方法是当用户修改文件时,允许它锁定整个文件;另一种方法是在修改时锁定单个记录。因此,在设计共享访问能力时必须解决互斥问题和死锁问题。

早在 20 世纪 70 年代前,已经出现了不少实现文件共享的方法,如绕道法、连访法,以及利用基本文件实现文件共享的方法;现代的一些文件共享方法是在早期这些方法的基础上发展起来的。下面介绍常用的文件共享方法。

8.1.3 基于索引节点的文件共享

在树形结构的目录中,当有两个(或多个)用户要共享一个子目录或文件时,必须将共享文件或子目录链接到两个(或多个)用户的目录中,才能方便地找到该文件,如图 8-1 所示。此时该文件的目录结构已不再是树形结构,而是个有向非循环图。

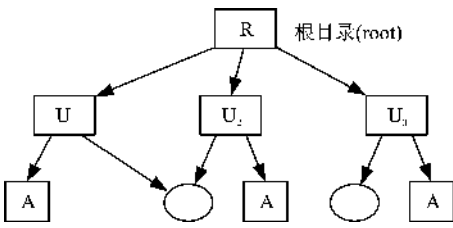


图 8-1 包含共享文件的文件系统

怎样建立 用户目录与共享文件之间的链接呢?若在文件目录中包含了文件的物理地址,即文件所在盘块的盘块号,则在链接时,必须将文件的物理地址拷贝到 用户目录中去。然而,若以后 用户或 用户还要继续向该文件中添加新内容,也必然要相应地再增加新的盘块。而这些新增加的盘块,也只会在执行了操作的目录中。可见,这种变化对其他用户而言,是不可见的,因而新增加的这部分内容已不能被共享。

为了解决这个问题,可以引用索引结点,即文件的物理地址及其他的文件属性等信息,不放在目录项中,而放在索引结点中。在文件目录中只设置文件名及指向相应索引结点的指针,如图 8-2 所示。此时,由任何用户对文件进行增加或修改所引起的相应结点内容的改变,都是其他用户可见的,从而也就能提供给其他用户共享。

索引结点中还设有一个链接计数 链接域 表示链接到本索引结点上的用户目录项的个数。当 链接域 为 2 时,表示有两个用户目录项链接本文件上(即两个用户共享此文件)。

当用户 用户创建一个新文件时,他便是该文件的所有者,此时 用户是 文件所有者。当有用户 用户要共享该文件时,在用户 用户的目录项中增加一目录项,并设置一指针指向该文件的索引结点,此时。文件主仍然是 用户,链接域 为 2。若用户 用户不再需要此文件,能否将它删除呢?回答是

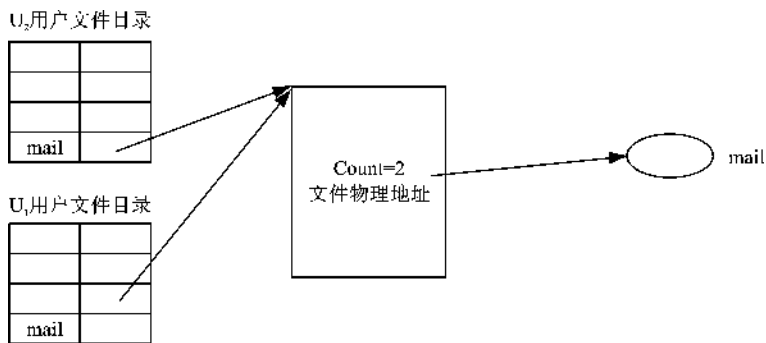


图 9-1 基于索引结点的共享方式

否定的。因为若删除了该文件，将一定删除该文件的索引结点，于是 U_1 的指针悬空，而 U_2 在此文件上进行的操作将半途而废。

利用符号链实现文件共享

为了使 U_1 能共享 U_2 的一个文件 F ，可以由系统创建一个 F 类型的新文件，也命名为 F 并将 F 写入 U_1 的目录中，以实现 U_1 的目录与文件 F 的链接。在新文件中只包含被链接文件 F 的路径名。这种链接方式称为符号链接。新文件中的路径名，被看作是符号链，当 U_1 要访问被链接的文件 F 且正要读 F 类型文件时，将被操作系统截获，操作系统根据新文件中的路径名去读该文件，由此实现了用户 U_1 对文件 F 的共享。

利用符号链方式实现文件共享时，只有文件主才拥有指向其索引结点的指针，而其他共享该文件的用户，则只有该文件的路径名，并不拥有指向其索引结点的指针。因此，不会发生共享文件删除后留下悬空指针的情况。当文件主删除该共享文件后，若其他用户试图通过符号链去访问一个已被删除的共享文件时，会因系统找不到该文件而使访问失败，于是再将符号链删除，此时不会产生任何影响。

符号链的共享方式有一个很大的优点，即它能够用于链接（通过计算级网络）世界上任何地方的机器中的文件，只需提供该文件所在机器的网络地址及在其中的文件路径名即可。

符号链的共享方式也有它的缺点：当其他用户去读共享文件时，系统根据给定的文件路径名，逐个地去查找目录，直至找到该文件的索引结点。因此，在每次访问共享文件时，可能要多次地读盘。因此，不仅每次访问文件的开销甚大，而且增加了启动磁盘的频率。另外，为用户创建的符号链，实际上是一个文件，虽然文件非常简单，却要为其配置一个索引结点，这也要耗费一定的磁盘空间。

绕道法

绕道法又叫绕弯路法，是早期的 $MS-DOS$ 等操作系统中所采用的一种共享文件的方法。绕道法要求每个用户处在当前目录下工作，用户对所有文件的访问都是相对于当前目录进行的。用户文件的路径名是由当前目录到信息文件通路上所有各级目录的目录名加上该信息文件的符号名组成。使用绕道法进行文件共享时，用户从当前目录出发向上返回到所要共享文件所在路径的交叉点，再顺序下访到共享文件。绕道法需要用户指定所要共享文件的逻辑位置或到达被共享文件的路径，绕道法的原理如图 9-2 所示。

为了访问一个不在当前目录下的共享文件，绕道法需要花费很多时间去绕弯路访问多级目录，因而搜索效率很低。

源访问法

为了提高共享其他目录中文件的速度，一种共享的方法是在相应目录表之间进行链接，即将一个目录中的表目直接指向被共享文件所在的目录，这样被链接的目录以及其子目录所包含的文件都成为了共享的对象。在图 8-1 所示的树形目录结构中，如要访问新增一个符号名为“源”的目录表目，该表目要共享“源”中的文件，则采用连访法的做法是，将符号名为“源”的目录表目中的“源”指针直接指向“源”子目录中的共享文件名的目录表目。采用这种方法，在表目中必须增加一个“连访”属性，指明其中的物理地址是指向文件还是指向共享文件的目录表目。另外，还要有一个“链接计数”表目，以指明是否还有用户需要共享文件。仅当再无用户需要共享文件时，才可以将此共享文件撤销。

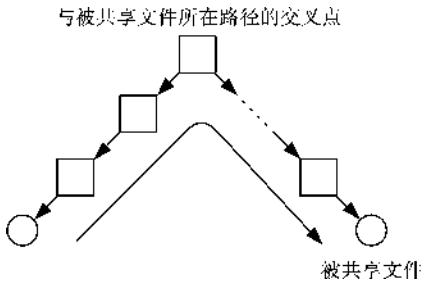


图 8-1 源访问法

基本文件目录表

实现文件共享的一种有效方法是采用基本文件目录表（Basic File Directory Table），记录文件的说明信息。基本文件目录中所有文件被系统赋予惟一的标识符。同时系统又为每个用户建立一个符号文件目录（Symbolic File Directory），记录由用户给出的文件符号名和系统赋给文件说明信息的惟一标识符之间的对应关系。这样文件目录的内容由两个部分组成：基本文件目录和符号文件目录，整个目录结构如图 8-2 所示。基本文件目录中的“主”项固定赋予基本文件目录、空闲文件目录和主目录。

在这种方式下，由于文件的符号名和文件说明信息是各自不同的两部分。因此，当一个用户要共享另一个文件时，只需要在他的符号文件目录中增加一个表目，表目的内容是用户为被共享文件取的符号名和系统赋予此文件的惟一标识。用户就可以用自己的文件名使用该文件了。

如图 8-2 所示，用户“源”和“源”共享基本文件目录中标识符为“源”的文件，对用户“源”而言，该文件为“源”；对于用户“源”而言，该文件为“源”。用户“源”使用文件“源”时，系统首先通过基本文件目录找到主目录，在主目录中查找到“源”的符号文件目录中“源”文件的标识符为“源”，于是到基本文件目录中读出标识符为“源”文件信息。同理当用户“源”使用文件“源”时，系统可以查找到“源”的符号文件目录得到文件“源”的标识符也为“源”。可见，用户“源”和用户“源”从基本文件目录中读出的是同一个文件，实现了文件的共享。

文件的保护

“文件保护”是指防止文件被破坏，包括三个方面，一是防止未经授权的用户使用文件，二是防止文件主自己错误地使用文件而给文件带来伤害，三是物理破坏。为此，文件系统经常采用建立副本和定时转储的方法来保护文件。对于共享文件而造成的破坏，可以采用存取控制矩阵、存取控制表、口令和密码等方法，来达到保护文件不受侵犯的目的。

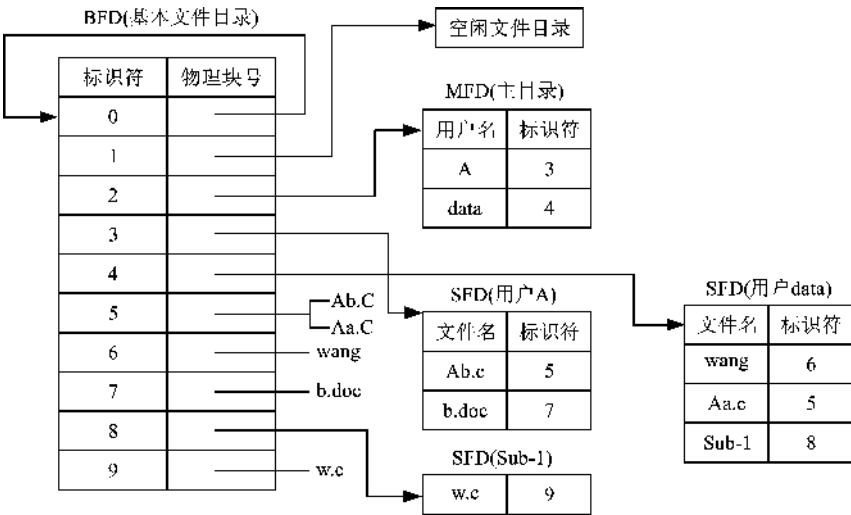


图 2-1 采用基本文件目录和符号文件目录的多级目录结构

存取控制矩阵

所谓“存取控制矩阵”，即是整个系统维持一个二维矩阵，二维矩阵的一维列出系统中的所有文件名，另一维列出系统中所有的用户名，对应的矩阵元素则是用户对文件的存取权限，包括读、写、执行和删除等。如图 2-2 所示。

如图中的用户 wang 对文件 Ab.c 只能读，对文件 b.doc 可以读、写和执行等等。对应的矩阵元素为空白时，表示用户无权对此文件进行任何访问。文件系统接收到来自用户对某个文件的操作请求后，根据用户名和文件名，查存取控制矩阵，检验命令的合法性。如果请求的命令与矩阵中的限定不匹配，系统则拒绝执行。只有在命令请求与存取控制权限的要求相匹配时，才能去完成具体的文件存取工作。存取控制矩阵的道理虽然简单，但若系统中的用户数和文件数都很大，则该矩阵里的空项将非常多。保存这样一个大而空的矩阵，将是对磁盘存储空间的一种浪费，而且为使用文件而对矩阵进行扫描的时间开销也非常大。因此，在实现时一般采取某些辅助措施以减少时间和空间的开销。

权限 \ 文件名	用户	wang	li	zhang	zhu
		读	写	读	读
wang		读	写	读	读
li		读	写	读	写
zhang		写	写	写	读
zhu		读	写	读	读
zhang		读	写	读	读

图 2-2 存取控制矩阵

存取控制表

如上所述，若只按存取控制矩阵的文件存储，且只存储非空元素，就形成了所谓的“存取控制表”。存取控制表是以文件为单位，将用户按某种关系划分为若干组，同时规定每组的存取权限而构成。每一个文件都有一张存取控制表，如图 8-1 所示。在实现时，该表存放在文件说明中，也就是文件的有关表目中。当文件被打开时，存取控制表也相应地被复制到内存活动文件中。由此，存取控制验证能高效进行。

文件 用户	存取权限
文件	读、写、执行
用户组	读
月组	写
进程	读、写、执行
其他	无

图 8-1 存取控制表

口令

前面是系统对文件提供的保护机制，口令则是一种验证手段，也是最广泛采用的一种认证形式。当用户发出对某个文件的使用请求后，系统会要求他给出口令。若输入的口令与原来的口令不一致，该用户将被系统拒绝。

采用口令的方式保护文件，容易理解，也容易实现。但由于口令简单，经常被内行人破译，以至于达不到保护的效果。另外，口令也容易遗忘、记错，给文件的使用带来不必要的麻烦。

密码

密码方式是为了防止文件泄密以及控制存取访问的另一种方法。它是把文件信息转换成密码形式保存，在读出文件时再对其进行译码解密。密码的编码方式只限文件主及允许使用文件的用户知道，因此非授权用户就无法访问文件信息。

文件的加密和解密都需要用户提供一个密钥。加密程序根据这一密钥对用户文件进行密码变换，然后将其写入存储设备。在读取文件时，只有用户给定的密钥与加密时的密钥一致时，加密程序才能对加密文件进行解密，将其还原为源文件，加密处理过程如图 8-2 所示。

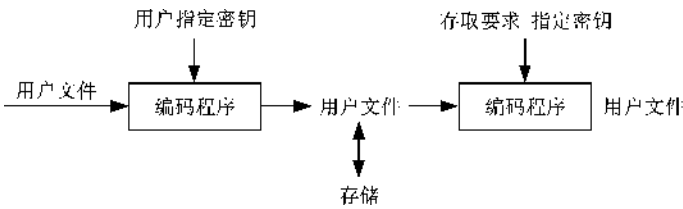


图 8-2 加密解密过程

加密方式具有保密性强的优点，因与口令不同，进行编码解码的密钥没有存放在系统中，而是用户掌握。但采用密码的方法却增加了文件重新编码和解密的工作，使系统的开销增大。

密码技术发展迅速，保密性很强，实现时也很节省存储空间，但需花费较长的译码时间。目前，也有由应用程序提供加密保护的实现方式。

文件备份

为了增强系统存储文件的可靠性，一般简单的方法是给重要文件以多个副本。有两种形成文件备份的方法，一种是批量备份，即定期地进行文件备份拷贝；另一种是同步备份，即

在生成或写文件的同时备份该文件。

(录 批量备份

批量备份又称为批量转储，是将一批文件复制到后备存储器中。后备存储器可以采用磁带，也可用磁盘或光盘。一般常用两种转储方法：一种是全量转储，即把文件存储器中的全部文件定期复制到后备存储器中。当系统出现故障，文件遭到破坏时，便可把最后一次转储内容从后备存储器复制回系统以恢复正常进行。这种方法浪费机时，在转储过程中需停止使用任何文件。另一种是增量转储，即每隔一段时间便把上次转储以来改变过的文件和新文件用后备存储器转储。关键性的文件还可再次转储。这能克服全量转储的缺点，但后备存储器上的转储信息不紧凑。文件转储方法不但可以保护用户文件，而且当系统出现故障时，可用该方法修复系统。

(圆 同步备份

上述两种文件转储方法不具备实时性，即在上一备份时刻后生成或写过的文件不可能被恢复。另外，在操作上过于麻烦。为了防止计算机故障或磁盘介质的损坏而引起文件破坏，现在主要采用同步备份的方法。主要有两种方法，一种是磁盘镜像，如图 怨原圆所示，系统拥有一个完全相同的镜像盘，在对磁盘写操作的同时，对称地写其镜像盘，确保一个存储介质损坏时，另一个介质数据能用。磁盘镜像功能虽可有效地解决在一台磁盘机故障时的数据保护问题，但若控制这两台磁盘驱动器的磁盘控制器发生故障，或主机到磁盘控制器之间的通道发生了故障，此时将使这两台磁盘机同时失效，磁盘镜像功能也起不到数据保护的作用。另一种是磁盘双工，它是指有两台机器进行文件写操作时完全对称地工作，保证一台机器出错时，另一台机器还可以接着往下工作，如图 怨原圆所示。双机动态文件备份加上双份盘就可防止来自处理机和存储介质两方面损坏对文件系统引起的破坏。

磁盘镜像和磁盘双工，在当前计算机系统工作中经常使用，是一种行之有效的数据保护手段。

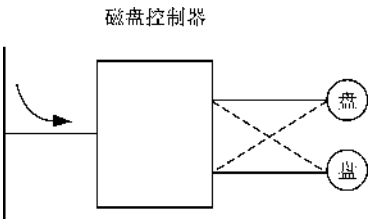


图 怨原圆 磁盘镜像

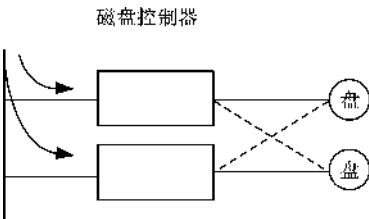


图 怨原圆 磁盘双工

文件的系统调用

一般的文件系统通过系统调用方式或命令方式为用户提供了下列几类服务功能：

- ① 设置和修改用户对文件的存取权限；
- ② 建立、改变和删除目录；
- ③ 实现文件共享、设置访问路径；
- ④ 创建、打开、读写、关闭及撤销文件。

实现上述服务的命令数目及其功能根据操作系统环境的不同而异。有关对文件操作的命令都基于操作系统提供的系统调用。这些系统调用包括创建文件 `creat`、读文件 `read`、关闭文件 `close` 以及撤销文件 `unlink` 等。一般的操作系统均能提供如图 8-1 所示的服务功能。其中最基本的文件操作主要是创建文件、删除文件、打开文件、关闭文件、读文件和写文件等。

名 称	功 能
<code>creat</code>	创建一个新的文件到系统目录
<code>unlink</code>	从系统目录中撤销一个文件
<code>rename</code>	在系统目录中改变文件的名字
<code>chmod</code>	设置文件属性
<code>mount</code>	在用户和文件（或设备）之间关闭一个逻辑通路
<code>mkdir</code>	在用户和文件（或设备）之间建立一个逻辑通路
<code>write</code>	写到一个文件（或设备）上
<code>read</code>	从一个文件（或设备）读入数据信息
<code>ioctl</code>	确定在一个给定设备上可利用的磁盘区域的大小
<code>link</code>	从一个文件到其他文件之间创建一个连接
<code>unlink</code>	撤销到文件的逻辑连接
<code>chown</code>	改变文件的 所有者或

图 8-1 常用文件操作功能

- ① 创建文件。创建文件时需要两个操作。系统首先要为新文件寻找存储空间，然后在文件系统的目录中建立一个新的目录项，目录项中记录新文件的文件名、文件在外存的地址以及其他文件说明信息。
- ② 读文件。在读一个文件时，系统调用需给出文件名和该文件下一块在内存的地址。接收命令后，系统在目录中查找相关的目录条目，找到指定文件的目录项。目录也需要一个指向下一被读文件块的指针，若读出此块，该指针应作相应的修改。
- ③ 写文件。写一个文件时，需要在系统调用中给出文件名及待写信息。给出一文件名，系统就在目录中查找该文件位置，目录条目保留一指针，它指向文件的当前末端处。使用此指针，计算出下一块的地址，写入信息。写入之后，指针需作修改。
- 一般，在读文件时，虽然可以同时使用两个指针：读指针和写指针，然而多数系统都只有一个，即文件当前位置指针，读操作都使用此指针。这样能够节省目录条目空间，降低系统复杂性。
- ④ 删除文件。删除文件时，系统先从文件系统的目录中找到要删除文件的目录项，释放该文件的空间并删去相应的条目。
- ⑤ 打开文件。为了避免用户在每次访问文件时从外存中查找文件目录，系统提供了打开文件操作。该操作的功能是将待访问文件的目录信息读入内存活动文件表中，建立用户和文件的联系。一旦文件被打开一次后就可以多次使用，直到文件被关闭。

⑥ 关闭文件。关闭文件是撤销主存中有关该文件的目录信息，只有文件的建立者或打开者才有权关闭文件。首先，检查读入主存的文件目录或索引表是否被修改过，若该文件被修改，则系统会先执行写操作，将其写入外存，然后在“已打开文件表”中清除该文件。一个文件关闭之后，若要再次访问该文件，则必须重新打开。

图例 文件系统的层次模型

在以上各节中，主要从系统和用户两方面的角度，讨论了文件系统的基本概念和功能，下面介绍文件系统的一般层次模型，其中子模块的层次关系如图 图例 所示。

自从 图例 于 图例 年提出用层次结构的方法设计操作系统后，图例 年 图例 首先把这一思想引入文件系统，将实现文件系统各种功能的一组软件分成 图例 层，下层为上层提供服务，上层调用下层的功能。这样，上下层之间彼此无需了解对方的内部结构和实现方法，而只关心二者的接口。于是，一个看上去十分复杂的系统由于层的划分而变得易于设计、易于理解和易于实现。而且，当系统出现错误时，也容易进行查错和调整。因此，层次化设计方法也使得系统的管理和维护更加容易。

在层次化设计中，层次的划分是一个十分复杂的问题。若层次划分太少，则每一层的内容仍然十分复杂，分层的意义不明显。若层次太多，则各层之间传递的参数会急剧增加，而且每一层的处理需要一定的系统开销，从而影响系统效率。因此，层次的划分要根据实际需要仔细考虑。

图例 用户接口层

用户接口层由若干个程序模块组成，每一模块均对应一条系统调用。用户发出系统调用时，其控制权即转入相应的模块。该层的主要功能如下：

- ① 对用户发出的系统调用参数进行语法检查及合法性检查。
- ② 把系统调用码及参数转换成内部调用格式。
- ③ 补充用户缺省提供的参数，并完成相应的初始化。
- ④ 调用下层程序，并负责与用户通信。

图例 符号文件系统层

该层完成第一层所提供的功能，并把第一层所提供的参数——用户文件名转换成系统内部的惟一标识符 图例 。该层的主要工作是搜索文件目录，找到相应文件名的表目以找到 图例 ，然后将 图例 作为参数传给第三层。

图例 基本文件系统层

该层根据第二层的调用参数 图例 找到文件的说明信息，包括存取控制表、文件逻辑结构、物理结构以及第一个物理块地址等。

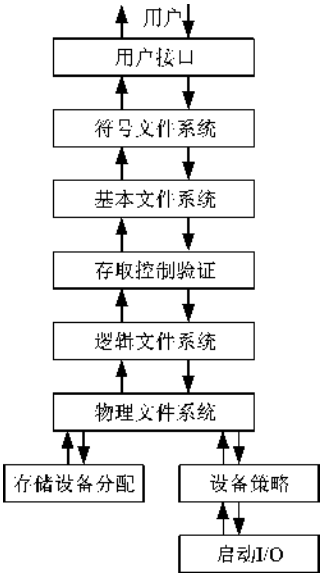


图 图例 文件系统的层次模型

源存取控制验证层

文件的保护主要由该层完成，它把用户的访问要求与控制块中指示的访问控制权限进行比较，以确定访问的合法性。若不合法，则向上层返回出错信息，表示请求失败；若合法，则将控制权传递给逻辑文件系统。

逻辑文件系统层

该层的主要功能是，根据文件的逻辑结构将用户欲读写的逻辑记录节转换成文件逻辑结构内的“相对块号”和“块内相对地址”，进行实际的文件读写。对于流式文件来说，只要把用户指定的逻辑地址按块长换算成相对块号就可以了。然而，对于记录式文件来说，由于用户有时指定的是键名（或记录名），因此需首先由键名（或记录名）搜索到相应的记录并得到对应的逻辑地址之后，再将其转换为相对块号。

物理文件系统层

该层的主要功能是，把逻辑记录所在的相对块号转换成实际的物理地址。文件的物理结构不同，确定物理块号的方法也不同。另外，它也负责与下层进行通信。若本次是写操作，且尚未给写入的记录分配外存空间，则调用外存分配模块分配物理块，然后再调用下层的设备管理程序模块，进行实际的写操作。对于读操作，则直接调用设备管理程序模块，进行实际的读操作。

存储设备分配模块和设备策略模块层

文件存储设备分配模块实现对空闲存储块的管理，包括分配、释放和组织。

设备策略模块主要是把物理块号转换成相应文件存储设备所要求的地址格式，如磁盘的柱面号、磁道号、盘区号等。然后根据具体的操作要求及必要的参数，准备启动输入输出设备的命令。

启动层

由设备管理程序执行具体的读写文件操作。

需要指出的是，上述文件系统的基本模型的分层方法不是惟一的，对各层的功能亦可调整。但模型的结构思想都是相同的，只要掌握了上述模型的层次结构思想，就容易解析和理解其他文件系统。

文件系统

文件系统概述

操作系统中的文件系统是其设计实现技术中最重要的也是最出色的模块之一，与其他操作系统的文件系统比较，它具有结构简单、管理规范、使用灵活等鲜明的特点。

文件系统的结构特点

文件的文件系统是按层次结构建立的。文件系统内部使用一个多级目录结构管理构件，控制着多级多层次的文件体系，文件系统从目录开始向下延伸组成了文件系统的倒置树形的目录关系体系，如图 8-1 所示。这棵树的根是根目录，树上的每个节点都是一个子目录，而树的叶节点则是信息文件。每个用户都可建立自己的文件系统，并把它安装到文件系统上，从而形成一棵更大的树；用户也可完整地拆卸已安装的文件系统。因此，整个

文件系统显得十分灵活、方便。

操作系统内核把所有文件都看作是字节流，任何内部逻辑结构都是专用的。然而，操作系统不区分文件的物理结构。操作系统的文件的物理结构为混合索引式文件结构，即包含多种索引文件结构，如单级索引文件结构、两级索引文件结构和多级索引文件结构。这种物理结构既可提高对它的查询速度，又能节省存放文件地址所需的空

间，还可以使文件动态地增长或减少。操作系统除一般的文件（如文字、程序等信息文件）外，为管理和调度方便，操作系统将系统中的有关设备也定义成“文件”，并用一般文件的调度和管理方法解决系统的硬设备调度问题。这样操作系统内核程序设计更加简单、更加精练，并且使用户感觉对设备的访问就如同对普通文件访问一样简单方便。

文件的分类

按照文件的内容，操作系统把文件分成三类：

（1）普通文件。这是通常意义下的磁盘文件，文件中包括的信息是由用户、应用程序或系统实用程序输入的。它们都是无结构、无记录概念的字符流式文件，文件的长度可动态增减。在管理中，系统将为文件设立各种属性信息，因为操作系统系统中文件的管理是通过对文件属性信息的管理完成的。操作系统系统中普通文件的基本属性包括：文件名、文件的索引号、字节数、最后访问时间、文件存取权限、属主名、属组名等。

（2）目录文件。操作系统系统中的目录文件是一种特殊的文件类型，在应用上与其他操作系统中的目录定义类似，而在文件管理中仍作为一种文件来对待。操作系统目录文件包含若干个目录项（至少应有两个目录项，即当前目录项和父目录项）。每个目录项包含文件名、文件描述信息、文件链接关系等，目录中的所有信息构成了目录文件的内容。组成操作系统文件目录项的主要内容有：目录项文件名、索引节点号。

操作系统的目录结构与树形和层次结构的目录结构相似，是分层次的树状结构，如图例所示。它们的不同之处是，操作系统的目录结构中只有一个根节点，而树形和层次结构的目录树结构中可有多个根节点，一般是每个盘区有一个根节点，如树形、层次结构。

目录文件实质上是具有特殊的写保护特权的普通文件，只有文件系统才能够对它进行写操作，但是所有用户程序都允许对它进行读访问。

另外，操作系统的目录树是通过“安装”（安装操作形成的，也可以用“卸载”（卸载操作卸载某些目录。因此文件系统是一个具有灵活管理机制的文件系统，可以用

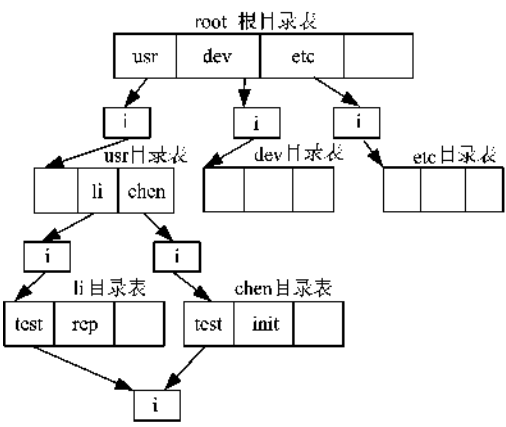


图 例 操作系统文件结构

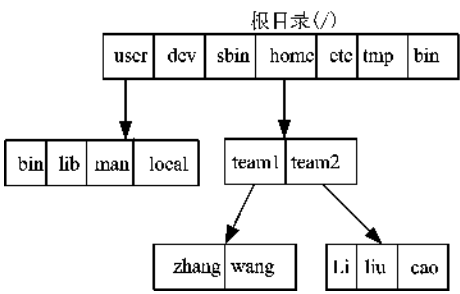


图 例 操作系统的目录结构

命令对用户需要的文件、文件资源和文件目录合理地进行安装和卸载,最大限度地保证用户使用方便,目录管理简单。

(3) 特殊文件。在 FAT 中,把块存储设备(如磁盘)和字符设备(如键盘、打印机)都视为文件。不过由于它们只有文件目录和索引节点,并不占用实际的物理存储块,因此,称它们为设备文件。为了检查和处理方便, FAT 总是把所有的特殊文件放在名为“ dev ”的目录下。

FAT 通过引入特殊文件,缩小设备管理和调度与一般性文件管理和调度之间的差异,通过对用户隐藏各种硬设备具体物理特性的方法,将操作系统中对文件管理和对设备管理的程序统一进行设计。这样使用户感觉对设备的访问就如同对普通文件访问一样简单方便。

8.1.3 文件系统的物理结构

FAT 文件的物理结构采用的是混合索引文件结构,这种索引结构是通过每个文件节点中的数组 block_list 来形成文件存储索引表的。该数组总共有 n 个元素,每个元素为一个索引项。前 m 个索引项直接指向文件数据存放的磁盘块号,后 $n-m$ 个索引项分别构成一次间接索引、二次间接索引和三次间接索引的多级索引结构。在查找文件时,便可采用直接或间接的寻址方式获得指定文件的盘块。

8.1.3.1 直接寻址

当一个文件的长度在 $m \sim n$ 个磁盘块之间时,就称为小型文件。这时,用文件节点中数组 block_list 的前 m 个地址项,直接指向文件数据存放的磁盘块号,这种方式就称为直接寻址方式,如图 8-10 所示。因此,在 FAT 中,小型文件是通过节点中的数组 block_list 构成一级索引表而获得文件在磁盘上的存储位置的。

地址映射的方法是:首先将文件的逻辑块号转换为索引节点中地址项的下标,再从该下表所指的地址项,即可获得物理块号。例如,某进程要访问偏移量为 1000 处的数据,设块长为 512 ,先将 1000 转换成文件逻辑块号 2 ,块内偏移量 448 。由于逻辑块号小于 m ,因此,可直接从相应的地址项 $\text{block_list}[2]$ 中找到物理块号为 10 的块,如图 8-10 所示,该块中的第 448 字节处即为文件的第 1000 字节。

8.1.3.2 一次间接寻址

当一个文件的长度在 $m \sim n$ 个磁盘块内时,就成为一个 FAT 的中型文件。这时除了用 $\text{block_list}[0] \sim \text{block_list}[m-1]$ 外,还要用到 $\text{block_list}[m]$,如图 8-11 所示。在图 8-11 中, $\text{block_list}[0] \sim \text{block_list}[m-1]$ 仍然直接指向文件数据存放的 m 个磁盘块号,然后又利用 $\text{block_list}[m]$ 指向一个磁盘块。这个块并不存放文件的数据,而是利用它形成一级的索引,这就称为一次间接寻址方式。在 FAT 中,设一个磁盘块大小为 512 ,设一个磁盘块号占用 4 个字节,因此一个磁盘块可以存放 128 个磁盘块号。于是,通过 $\text{block_list}[m]$ 提供的索引,一个文件就可以达到 $128 \times m$ 个磁盘存储块这么大,寻址范围可达 $128 \times m$ 。

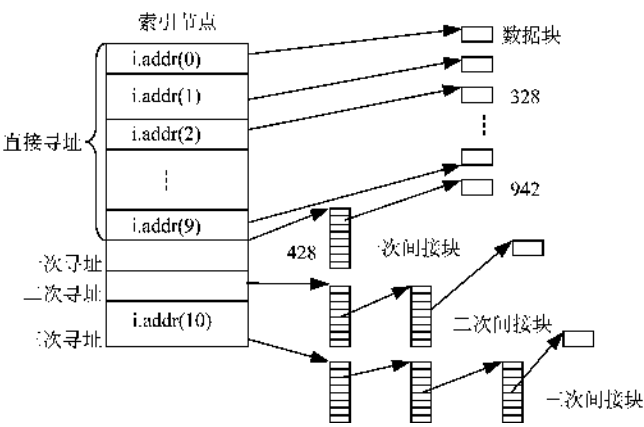
地址映射的方法是:首先从 $\text{block_list}[m]$ 中得到一次间址盘块号,读间址块;然后,计算一次间址中文件的逻辑块号,即将文件的逻辑块号减 m 。根据一次间址中的逻辑块号得到间址块号中地址项的下标,再从相应下标的地址项中得到物理块号。例如,某进程要访问偏移量为 1500 处的数据,设块长为 512 ,先将 1500 转换成文件逻辑块号 3 ,块内偏移量 448 。由于逻辑块号大于 m 小于 $128 \times m$,故应属于一次间址寻址。从 $\text{block_list}[m]$ 中得到一次间址盘块号为 10 ,读该

盘块；计算间址中的文件逻辑块号为 圆 则可得到对应的盘块号为 怨圆 因此，该块中的第 苑圆 字节处便是要访问的文件的第 员 字节，如图 怨圆 所示。

援多次间接寻址

当一个文件的长度在 圆~ 远 磁盘块内时，就成为一个 哉 的大型文件。这时除了用到 蚤 外，还要用到 蚤，还要用到 蚤，如图 怨圆 所示。在图 怨圆 中，蚤 直接指向文

件数据存放的 员 个磁盘块号；利用 蚤 指向一个磁盘块，由它提供 圆 个磁盘块的索引，从而使文件总共达到 圆 个磁盘存储块这么大；又利用 蚤 指向一个磁盘块，由它指向 圆 个磁盘块，每个都是一个索引。这就称为二次间接寻址。若还利用 蚤 指向一个磁盘块，由它指向 圆 个磁盘块，每个都是一个索引，每个索引再指向 圆 个磁盘块，每个又是一个索引，这样的方式称为三次间接寻址。二次间接寻址可以拥有 远 个磁盘块（即 员 垣 圆 垣 圆，若每块为 员，则寻址范围可达 远）。三次间接寻址可以拥有 员 个磁盘块（即 员 垣 圆 垣 圆 垣 圆，若每块为 员，则寻址范围可大于 员）。多次间址的转换方法和一次间址相似，但要多次循环，请读者思考。



摇摇图 怨圆 文件的多次索引存储结构

怨圆 哉 文件系统的存储结构及其关系

员 哉 文件系统的存储结构

哉 系统把文件信息存储在磁盘或磁带上。一般来说，哉 中的一个逻辑卷就是一个文件系统。在某个物理存储介质中可以包含有一个或若干个文件系统（即逻辑卷）。这些文件系统可以被动态装卸。一个文件系统中包含的模块是固定的，其中除了文件和目录的数据信息外还包括一些控制和管理信息。

文件系统由每块 缘 字节的任意值所构成的逻辑块序列组成。在同一个文件系统中，这些逻辑块的大小完全相同。块长的选取将直接影响设备与主存之间的数据传输速率和内存的存储能力。块长越大，内存与设备之间的数据传输越容易，但影响内存的有效存储能力。

一个 哉 的文件系统通常由引导块、超级块、索引节点表和数据区 源 部分组成，如图 怨圆 所示。

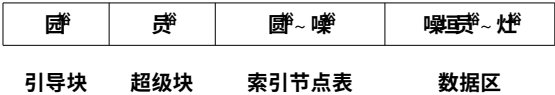


图 怨圆 哉 文件系统物理结构

引导块处于文件系统的开始部分,存放操作系统引导信息或系统的启动代码。在操作系统管理的多个文件系统中,只有具有引导功能的文件系统的引导块才装有启动代码,其余文件系统的此块为空。

超级块存放的是文件系统中和目录在磁盘上的静态分布描述信息,超级块对文件系统的维护很重要,包含的主要内容有:文件系统的状态、索引节点信息和存储块信息。

索引节点表存放的是文件系统中包含的所有文件的描述信息和数据结构,其中除了包括描述文件特性的基本数据项外,还包含每个文件在存储介质上的存储信息。

数据区存放本文件系统中包括的文件数据及目录文件数据等。若文件系统中有空闲区时,则还包含有空闲块的信息。

图 9-1 文件系统的静态数据结构

(员) 超级块的数据结构

文件的超级块中包含了文件系统中和目录的描述信息及文件的大小和结构,是文件系统管理中的重要数据结构。它的数据结构描述如下:

```

// 超级块数据结构描述
struct super_block {
    int s_inodes_count; // 索引节点数
    int s_blocks_count; // 块数目
    int s_bmap_blocks; // 保留块数目
    int s_free_blocks; // 空闲块数目
    int s_free_inodes; // 空闲索引节点数
    int s_first_data_block; // 第一个数据块
    int s_block_size; // 块大小
    int s_mount_time; // 装载时间
    int s_max_time; // 寿时间
    ...
};

```

在文件系统的管理中,通常将超级块数据结构存放在文件系统的多个地方(例如每个逻辑分区的块组中都包含有超级块的信息)。但只有图 9-1 模块组中的超级块才被读进内存,其他的仅作备份用。

(圆) 磁盘索引节点

一般的操作系统(如 UNIX 系统)都是将目录文件设计成文件的索引信息表,表中包括许多文件特征信息(如文件名、文件长度、文件修改日期等),内容非常繁杂。而 哉系统则构造一个简单的目录文件,在目录文件的基础上增加一项数据结构即索引节点(索引节点),用索引节点补充目录文件中的管理信息。哉系统将文件的大部分说明信息都放在索引节点结构中,使 哉系统形成了 哉系统文件管理的重要数据结构。磁盘索引节点实际上是描述文件的抽象数据结构,它的数据结构描述如下:

```

// 磁盘索引节点结构描述
struct inode {
    int i_mode; // 指出文件的属性:文件类型和存取权限。
    int i_uid; // 文件主标识。
    int i_gid; // 文件同组用户标识。
    ...
};

```

```
摇摇总目录项记录该文件有多少条路径。  
摇摇块目录项记录文件大小(以字节计数)。  
摇摇块目录项记录文件最近被访问的时间。  
摇摇块目录项记录文件被建立的时间。  
摇摇块目录项记录文件最近被修改的时间。  
摇摇块目录项记录文件被删除的时间。  
摇摇块目录项记录文件的块数。  
摇摇…  
}
```

磁盘中的文件在磁盘上是按索引结构组织的,每个文件都有一张索引表记录文件占用的磁盘空间块号。磁盘每个文件的索引表存放在文件的索引节点中,索引节点是该文件的索引表(见图 9.1.1)。磁盘索引节点存放在磁盘的索引节点区,当建立一个新文件时,就要为该文件分配一个索引节点号和存放索引节点的磁盘空间,将索引节点初始化后写到磁盘上,在文件目录中登记文件名和索引节点号。

文件系统的动态数据结构

为了使文件系统的访问灵活、便捷,仅有文件系统的静态描述还不够,因为静态结构无法表征文件系统管理过程中的动态特征,因此必须增加动态运行中的数据结构,并用动态数据结构描述动态运行中的相关信息。

(一) 内存超级块

内存超级块信息主要是针对文件系统的超级块信息而设定的。文件系统的超级块信息存放的是文件系统的资源管理信息。为了保证文件系统的超级块信息的正确性、完整性和管理的便利性,系统运行时在内存中给每个被加载的文件系统的超级块划出一个特定区域,这个区域就成为该文件系统的内存超级块。于是,当文件系统工作时,对文件的管理操作就可以在内存超级块中进行。而当一个文件系统用完后被卸载时,系统再将内存超级块中的内容复制到文件系统的超级块中,以保证被访问过的文件系统与磁盘中保存的内容一致。

(二) 活动索引节点表

当系统打开文件并对其进行搜索、读写等操作时,需为相应的文件分配一个内存索引节点。以便把对应磁盘索引节点信息复制到内存,内存中这部分信息保存着当前所有活动文件的索引节点,构成活动索引节点表,从而使得对文件的访问更加便利、更加快捷。

(三) 用户打开文件表

系统内部为了记录并控制用户或进程打开文件个数及使用共享文件的情况,在内存中建立了一个用户打开文件表的数据结构。它是进程的私有数据项,是进程数据结构中的一部分内容,系统使用它可以进行文件访问控制和文件共享管理。

(四) 系统打开文件表

在内存中保存着一个公用数据结构,它就是系统打开文件表。它记载着整个系统中被打开文件的基本情况,其中包括指明打开同一文件的不同进程、不同进程使用的不同打开路径以及这些不同进程和不同打开路径所对应的读写指针,等等。

(五) 安装表

安装表记录系统中各个文件卷的安装情况,如安装状态、文件卷的索引节点、超级块的

指针，等等。

文件的动态数据结构与静态数据结构相互关联，形成动态管理记录，构成文件系统的实际管理体系结构，如图 3-1 所示。

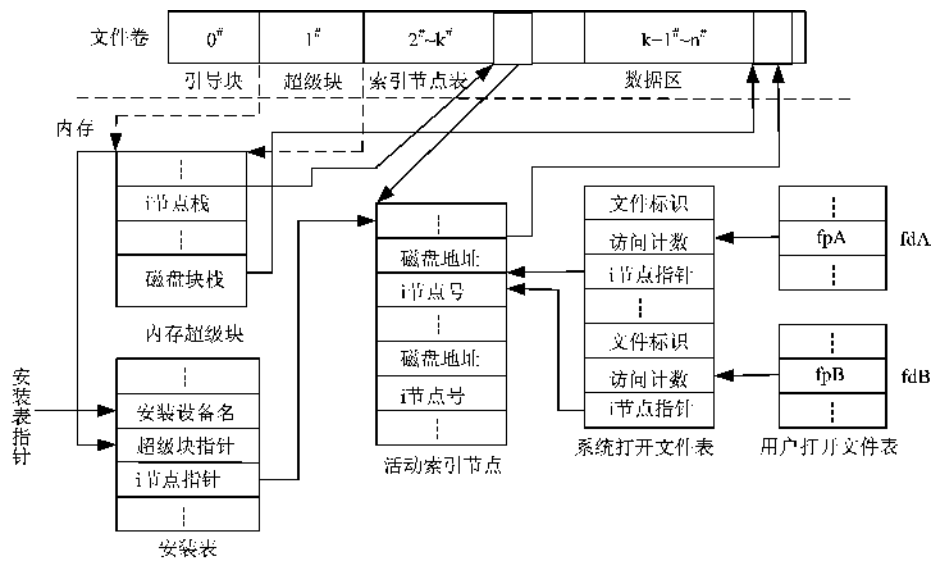


图 3-1 文件系统中动态数据结构与静态数据结构关系图

在图 3-1 中，用户进程通过用户打开文件表中的文件描述符找到系统打开的文件表的入口地址，再由系统打开文件表中对应项找到相关索引节点的入口指针，从而得到操作该文件所需的索引节点控制信息，实现对磁盘文件数据区的操作。另外图中，用户进程各打开了一个文件，由文件描述符找到了系统打开该文件的对应项，再通过它们的索引节点指针寻找到对应的索引节点（同一个内存索引节点），从而实现文件共享。

空闲磁盘空间的管理

空闲盘块的组织

在磁盘上，总是把文件安排在一般数据存储区。因此，对文件存储空间的管理，即是对磁盘上一般数据存储区的管理。前面曾提及“成组链接”法，并说操作系统就是采用这种方法来管理磁盘上的空闲块的。下面详细介绍它的实现过程。

系统把每 16 个空闲块作为一组，每一组的第一个空闲块中登记下一组空闲块的块号和空闲块数，余下不足 16 块的那部分空闲块的块号及块数登记在一个文件资源表中，登记最后一组块号的那个空闲块其中第 16 个单元填“0”，表示该块中指出的块号是最后一组的块号，空闲块链到此结束，因此最末一组只有 15 个盘块，如图 3-2 所示。

空闲盘块的分配与回收

在采用“成组链接”法后，系统可根据申请者的要求进行空闲块的分配，并在释放文件时回收空闲块。下面来讨论成组链接法的分配和回收过程。

首先，系统在开始启动时把文件资源表复制到内存，于是把存放第一组空闲块块号与总

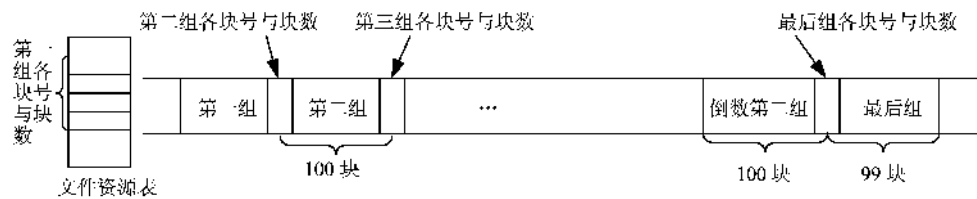


图 8-1 磁盘块成组链的组织

块数的堆栈进入内存，因此空闲块的分配与回收可在内存进行。

当系统为用户分配文件所需的盘块时，首先检查空闲盘块号栈是否上锁，若为上锁，便从栈顶取出一空闲盘块号，将与之对应的盘块分配给用户，然后栈顶指针减 1。若该盘块号为栈底，即为当前栈中最后一个可分配的块号时，系统启动设备管理程序，将该块中存放的下一组的块号与总块数读入内存后再将该块分配给申请者。然后，把栈中的空闲盘块数减 1 并返回。如果分配一直进行，直到把最后一组空闲块全部分配出去，此时，提出请求的进程只能阻塞等待。

当系统回收空闲盘块时，将回收盘块的盘块号记入空闲盘块号栈的顶部，并执行空闲盘块数加 1 操作。若空闲盘块数数目已达 100 时，表示栈已满，便将栈中的 100 个盘块号，记入新回收的盘块中，再将其盘块号作为新栈底。

文件操作

前面介绍了文件系统的基本数据结构、磁盘块和节点的管理方法、逻辑文件块到物理块的变换算法以及节点的搜索方法等。现从用户使用文件系统的角度来介绍文件系统的操作。

文件系统中对文件的操作通过系统调用实现，它是用户程序获取操作系统服务的惟一方式。操作系统中提供丰富的系统调用，文件操作类的系统调用主要有：创建、删除、打开、关闭、读、写等。

在用户进程使用系统调用时，首先，执行该系统调用，于是系统产生一条 trap 指令的信息，触发启动中断和异常总控程序，使系统由用户态进入核心态。在 trap 指令启动中断和异常总控程序的同时，系统调用的各参数写入系统调用结构中对应部分。然后，由中断和异常总控程序，系统进入执行与系统调用有关的文件系统内部过程和缓冲区管理过程等，它的执行过程如图 8-1 所示。

文件系统设置一张“系统调用处理程序入口表”。每个表目的内容包含有：系统调用分类号、系统调用名称、系统调用所带的参数个数和系统调用处理程序的入口地址。当用户程序请求系统调用时，系统处理子程序根据 trap 指令中的

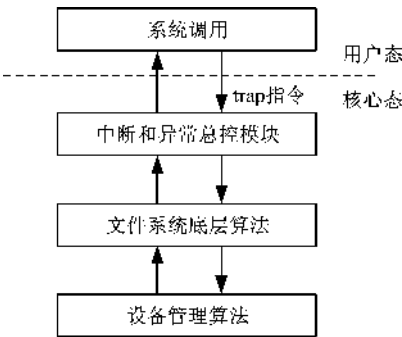


图 8-1 文件系统的系统调用执行过程

trap 处理子程序根据 trap 指令中的

系统调用分类号查系统调用处理程序入口表，获得该系统调用所带的参数个数和相应的处理程序入口地址，把参数传送到系统工作区后，按相应的处理程序入口地址转向系统调用处理程序执行。

下面，我们以系统调用 `open` 为例，进一步说明系统调用的过程。

打开文件

用户在第一次访问任何文件之前，都必须先调用 `open` 来打开指定的文件，然后才能对该文件执行读、写等操作。当不再需要使用此文件时，还需调用 `close` 将该文件关闭。

在调用了系统调用 `open` 之后，用户得到返回的文件描述符，且系统已将所要读文件的磁盘节点复制到内存节点中，并建立了用户打开文件表、系统打开文件表与内存节点之间的联系。例如，在系统调用 `open` 打开文件 `/usr/users/want` 后的数据结构之间的关系如图 8-1 所示。这里假定 `/usr/users/want` 已在同一个进程中被打开了 4 次，且每次都设有不同的读写权限。

`open` 系统调用的语法格式为：

```
int open(const char *pathname, int flags, mode_t mode);
mode_t * mode;
int fd;
```

其中 `pathname` 是文件的路径名；`flags` 是读写标志，它指出对文件的访问方法；`mode` 给出了文件的访问许可权，即规定了文件主、同组用户和其他用户的访问权限。`open` 系统调用成功，将返回文件描述符，否则返回 -1。

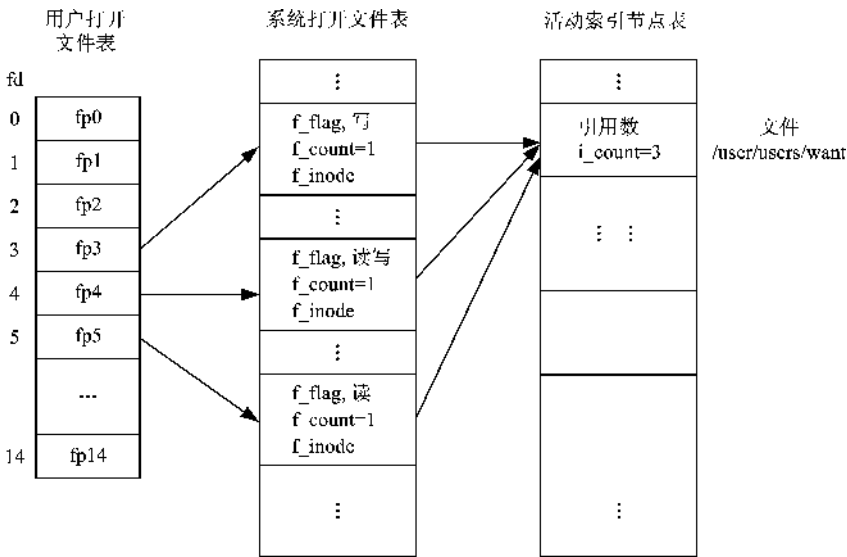


图 8-1 打开文件的数据结构之间的关系

打开文件系统调用的过程如下：

① 检索目录。核心程序先调用检索目录过程 `lookup` 从根目录或从当前目录开始，沿目录树查找指定目录的索引节点。若未找到指定文件或该文件不允许存取，则系统调用失败返

回，否则进入下一步。

② 分配内存索引节点。若该文件已被其他用户打开，此时只需对在第一步中所找到的索引节点执行其引用计数加 1 的操作；否则应为被打开文件分配一内存索引节点，并调用磁盘读过程将磁盘索引节点的内容拷贝到内存索引节点中，再将其引用计数置为 1。

③ 分配文件表项。为已打开的文件分配一文件表项，并使该表项指向内存索引节点。一般将读指针置为 0，表示从头开始读该文件，并设置读标志位为 0。

④ 分配用户文件描述符表项。从用户文件描述符表中获取一空表项，使其指针指向文件表项，并返回用户文件描述符。

读文件

对文件的各种操作中，读是最基本的操作，使用最频繁。文件读操作涉及到许多方面，如文件读权限、文件读顺序控制及读效率等等。

文件打开后，才可使用系统调用 `read` 和 `write` 对文件进行相应的读、写操作。这两个系统调用都需要用户提供三个参数：文件描述符 `fd`、缓冲区首地址 `buf`（对读而言是信息传送的目标地址，对写而言则是信息传送的源地址）；用户要求传送的字节数 `count`。

系统调用 `read` 的功能是从文件描述符 `fd` 所指的文件中读取 `count` 个字节的数据并将它们送入指针 `buf` 所指的缓冲区中。如果读成功，将返回实际读取的字节数。若再读时，文件指针已到达文件的末端，则即使未读够所指定的字节数 `count`，也立即停止传送，故实际读取的字节数可能小于 `count`。`read` 的语法格式如下：

```
int read(int fd, void *buf, rcount);
```

```
int write(int fd, const void *buf, wcount);
```

```
ssize_t readv(int fd, const struct iovec *iov, int iovcnt);
```

```
ssize_t writev(int fd, const struct iovec *iov, int iovcnt);
```

当执行 `read` 时，核心根据文件描述符 `fd` 获得指向文件表项的指针。接着检查本次读要求是否与文件存取权限一致。若两者相符，便将用户要求的 `count` 和 `buf` 直置入用户区中的用户地址和字节计数域中，并设置 `flags` 方式为读。然后再从文件表项找到文件对应的内存索引节点，并给索引结点上锁。对于一般文件，把文件表中的读指针赋给用户区中的字节偏移量。

读操作是一个循环过程。系统首先根据用户的要求把文件信息从外存读入系统缓冲区，再将系统缓冲区中的信息传送到指定用户区，同时修改用户区中的文件字节偏移量、用户地址和字节数域，为后续读做准备。读循环一直继续到本次读操作完成。最后，给索引节点解锁，修改文件表中的字节偏移量，返回实际读出的字节数。

本章小结

文件是用户所定义的一组相关信息的集合，具有文件名等属性。文件系统是一组系统软件，为使用文件的用户和应用程序提供服务，包括文件访问、目录维护和访问控制。

文件系统为用户提供了按名存取的功能。文件系统从存储介质的特性、用户的存取方法以及合理有效地存储和检索文件的角度来组织文件。用户是从使用文件的角度来组织文件，用户组织的逻辑文件有两种类型：流式文件和记录式文件。用户使用文件时可采用顺序存取

或随机存取的方法。系统在组织物理文件时根据存储介质的特性和用户选择的存取方法来决定存储结构。文件系统确定的物理文件主要有三种类型：顺序文件、链接文件和索引文件。对顺序存取的文件在磁盘上可组织成顺序文件、链接文件或索引文件，但在磁带上则只能组织成顺序文件；对随机存取的文件能在磁盘上组织成索引文件。

文件系统采用目录结构把文件的物理结构转换成逻辑结构提供给用户。文件目录是文件的名址录，通常组织成一级表、二级目录、树形目录结构。为了便于共享，也将目录组织成无环图结构。

文件存储空间的管理常见的有空闲文件目录、空闲链块、位示图三种方法。

文件系统通常提供的文件基本系统调用有：建立文件、打开文件、读文件、写文件、关闭文件和删除文件。

对文件的存取控制是和文件共享、保护保密紧密相关的。存取控制可采用存取控制矩阵、存取控制表、口令和密码的方法进行存取验证，以确定用户权限。

文件系统是指一个负责存取和管理文件信息的机构。一个采用层次结构的文件系统通常可分为：用户接口、符号文件系统、基本文件系统、存取控制验证、逻辑文件系统、物理文件系统、存储设备分配模块、设备策略模块和启动模块等层。

文件系统采用树形目录结构，系统采用文件名和文件说明信息分开放置的方法。用户进程通过用户打开文件表和系统打开文件表与文件系统联接，通过指定的路径名搜索到所要操作文件所对应的活动索引节点。

习题

1. 文件系统的模型可分为三层，试说明其每一层所包含的基本内容。

2. 试说明用户可以对文件施加的主要操作有哪些。

3. 何谓逻辑文件？何谓物理文件？

4. 如何提高对变长记录顺序文件的检索速度？

5. 试说明对索引文件和索引顺序文件的检索方法。

6. 试从检索速度和存储费用两方面来比较两级索引文件和索引顺序文件。

7. 试说明顺序文件的结构及其优点。

8. 什么是索引文件？为什么要引入多级索引？

9. 试说明混合索引分配方式。

10. 采用单级目录能否满足对目录管理的主要要求？为什么？

11. 目前广泛采用的目录结构形式是哪种？它有什么优点？

12. 试说明在树形目录结构中线性检索法的检索过程，并给出相应的流程图。

13. 有一计算机系统利用图 9.15 所示的位示图来管理空闲盘块。盘块的大小为 1024B，现要为某文件分配两个盘块，试说明盘块的具体分配过程。

	员	圆	猿	源	缘	远	苑	愿	怨	圆	员	圆	猿	源	缘	远
员	员	员	员	员	员	员	员	员	员	员	员	员	员	员	员	员
圆	员	员	员	员	员	员	员	员	员	员	员	员	员	员	员	员
猿	员	园	员	员	员	员	员	员	员	员	员	员	员	员	员	员
源	员	员	员	员	员	员	园	员	员	员	员	员	员	员	员	员
缘	园	园	园	园	园	园	园	园	园	园	园	园	园	园	园	园

图 怨愿某计算机系统的位示图

- 员援对空闲磁盘空间的管理常采用哪几种分配方式？在 哉葬操作系统中又是采用何种分配方式？
- 员援基于索引结点的文件共享方式有何优点？
- 员援基于符号链的文件共享方式有何优点？
- 员援当系统中有重复文件时，如何保证它们的一致性？

第 10 章 网络安全和保护

计算机安全是一个非常复杂的系统工程，包括的内容非常广泛，它既包括物理方面的，又包括逻辑方面的。在一个安全的计算机系统中，软件系统是非常重要的。在软件系统中最重要的是操作系统，所以操作系统的安全性是计算机系统安全性的基础。本章在介绍计算机信息安全基础知识的基础上，分析了操作系统安全产生的原因，讨论了操作系统的保护和安全管理机制，并对 3.5 版操作系统的管理做了简单的介绍。

10.1 操作系统安全概述

10.1.1 操作系统安全性概述

在信息化的进程中，信息已成为社会发展的重要资源，信息安全也成为 21 世纪国际竞争的重要战场。

影响计算机安全性的原因有很多，主要有以下几点：

首先，系统软件（包括操作系统软件）正变得越来越复杂。软件设计者在设计时无法预料程序运行时的各种系统状态，更无法精确预测在不同系统状态下会发生什么结果。软件复杂性和庞大的程序代码产生的软件错误很容易形成安全漏洞（漏洞）。

其次，计算机安全性问题是计算机及其技术飞速发展的必然产物。随着互联网需求的日益增长，将来自系统外部的服务请求完全隔离是不可能的；另一方面，系统的安全漏洞和系统的加密措施已经不再像以前那样仅仅被少数专业人士知道和少数的部门控制，随着互联网的普及，很多黑客站点发布相关的信息，并提供各种工具和技术以利用这些漏洞和破解加密体系，以进行系统攻击。

最后，随着计算机网络的迅速发展，除了信息的存储和处理外，存在大量数据传送操作如客户机访问服务器，一台计算机要传送数据给另一台计算机，这一过程中对安全的威胁极大，于是就需要有网络安全和数据信息的保护，防止入侵者恶意破坏。另外，计算机病毒也会通过网络扩散，破坏系统安全，需要采取措施预防、发现、解除它。

计算机系统在安全性上受到严重的威胁主要体现在以下几个方面：

（1）切断。系统的资源被破坏或变得不可用或不能用。这是对可用性的威胁，如硬切断通信线路或使文件管理失效。

（2）截取。未经授权的用户、程序或计算机系统获得了对某资源的访问。这是对机密性的威胁，如在网络中窃取数据及非法拷贝文件和程序。

（3）篡改。未经授权的用户不仅获得了对某资源的访问，而且进行篡改。这是对完整性的攻击，如修改数据文件中的值，修改网络中正在传送的消息内容。

（4）伪造。未经授权的用户将伪造的对象插入到系统中。这是对合法性的威胁，如非法

用户把伪造的消息加到网络中或向当前文件加入记录。

一般意义上讲,所谓一个计算机系统是安全的,是指该系统能够通过特定的安全功能控制外部对系统信息的访问。为了保证计算机系统的安全,计算机系统和网络通信提供的服务应满足四项安全要求:

(员 机密性 (机密性)。要求计算机系统中的信息只能由被授权者进行规定范围内的访问。这种访问可读或可视,如打印、显示以及其他形式,也包括简单显示一个对象的存在。

(圆 完整性 (完整性)。要求计算机系统中的信息只能被授权用户修改,修改操作包括写、改写、改变状态、删除和创建等。

(猿 可用性 (可用性)。防止非法独占资源,每当合法用户需要时,总能访问到合适的计算机系统资源,为其提供所需的服务。

(源 真实性 (真实性)。要求计算机系统能证实用户的身份,防止非法用户侵入系统,以及确认数据来源的真实性。

美国国防部 员愿年提出的“计算机可信系统评价准则”中把安全级别分为 苑个:

阅:最低安全性;

悦:自主存取控制;

悦:较完善的自主存取控制、广泛的审计;

月:强制存取控制,安全标识;

月:良好的安全体系结构、形式化安全模型、抗渗透能力;

月:全面的访问控制(安全内核)、可信恢复、高抗渗透能力;

粤:形式化认证、非形式化代码、一致性证明。

目前流行的几个操作系统的安全性分别为:诺尔:阅级;宰登蝶经:悦级和 杂登蝶经:悦级;韵云:月级;韵云:月级;韵云:月级。

软件系统中最重要的是操作系统,由于它的特殊地位,上述计算机安全性问题大部分要求操作系统来保证,所以,操作系统的安全性是计算机系统安全性的基础。

操作系统安全的概念

软件系统中最重要的是操作系统,如果没有操作系统安全机制的支持,就不可能保障其存取控制的安全可信性。在网络环境中,网络的安全可信性依赖于各主机系统的安全可信性,没有操作系统的安全性,就不会有主机系统和网络系统的安全性。因此,操作系统的安全是计算机及其网络系统的第一道安全屏障,是计算机及网络系统安全运行的基础,也是攻击者的首选目标。因此,操作系统的安全性已成为计算机系统、网络系统和建立在此基础上的各种应用系统安全性的最重要的基础。

操作系统安全包括了对系统重要资源(存储器、文件系统等)的保护和控制,即只有经过授权的用户和代表该用户进行的进程才能对计算机系统的信息进行访问。因此在使用操作系统安全这个概念时,通常具有如下两层含义:一是指操作系统在设计时,提供的权限访问控制、信息加密性保护、完整性鉴定等安全机制所实现的安全;二是指操作系统在使用过程中,通过系统配置,确保操作系统尽量避免由于实现时的缺陷和具体应用环境因素而产生的不安全因素。只有通过这两个方面的同时努力,才能最大可能地保证系统的安全。

一个操作系统的安全性可以从如下几个方面加以考虑：

(1) 物理上分离。要求进程使用不同的物理实体，例如将不同的打印机设置成具有不同的安全级别。

(2) 时间上分离。具有不同安全要求的进程在不同的时间运行。

(3) 逻辑上分离。系统通过限制程序的存取，使得程序不能存取其允许范围外的实体，从而使用户感觉自己的操作是在没有其他进程的情况下独立进行。

(4) 密码上分离。进程以一种其他进程不可知的方式隐藏数据及计算。

4.1 数据保密

操作系统的安全性是操作系统设计时首要解决的问题。安全性是指不因人为疏漏或蓄谋作案而导致的信息资源被泄漏、篡改或破坏，目标是反泄密。这一目标可通过加密技术的来实现的。

4.1.1 数据加密技术

数据加密技术基本思想就是隐藏、伪装信息，使未经授权者不能得到消息的真正含义。伪装信息的方法就是进行一组可逆的数学变换。伪装（变换）之前的信息是原始信息，称为明文（ P ）；伪装之后的信息，看起来是一串无意义的乱码，称为密文（ C ）。把明文伪装成密文的过程称为加密（ E ），该过程使用的数学变换就是加密算法。把密文还原成明文的过程称为解密（ D ），该过程使用的数学变换，通常是加密时数学变换的逆变换。加密与解密通常需要参数控制，我们把该参数称为密钥，有时也称为密码。加密时使用的叫加密密钥，解密时使用的叫解密密钥。加密密钥与解密密钥可能相同，也有可能不同。

因此，建立在密码系统上的加密体制是由明文空间、密文空间、密钥空间、加密算法与解密算法五个部分组成。明文、密文、密钥空间表示全体明文、全体密文、全体密钥的集合；加密算法与解密算法是一些公式、法则或程序，规定了明文与密文之间的数学变换规则。

下面用字母分别表示这个概念，密钥 K ， E 表示加密密钥， D 表示解密密钥，设明文 P ，密文 C ，加密算法 E ，解密算法 D 。

把明文加密为密文： $C = E(P, K)$

把密文解密为明文： $P = D(C, K)$

4.1.2 密码体制的分析

在多数情况下，信息加密是保证信息机密性的惟一方法。据不完全统计，到目前为止，已经公开发表的各种加密算法多达数百种。

加密和解密是一对逆过程。如果按照通信双方加密和解密密钥是否相同来分类，可以将这些加密算法分为对称加密和公钥密码算法。

4.1.2.1 对称加密

对称加密也称单密钥加密。在对称加密算法中，收信方和发信方使用相同的密钥，即加密密钥和解密密钥是相同或等价的。在 20 世纪 40 年代后期非对称加密出现以前，只有这一种加密技术。至今它仍然是使用最为广泛的加密方法。比较著名的对称加密算法有：美国的

及其各种变形，比如 凯撒密码、 维吉尼亚密码和 密码的前身 密码。欧洲的 密码（ 密码）；日本的 密码。以及以代换密码和转轮密码为代表的古典密码等。在众多的对称加密算法中影响最大的是 密码。

对称加密的方法如图 2-1-2 所示。

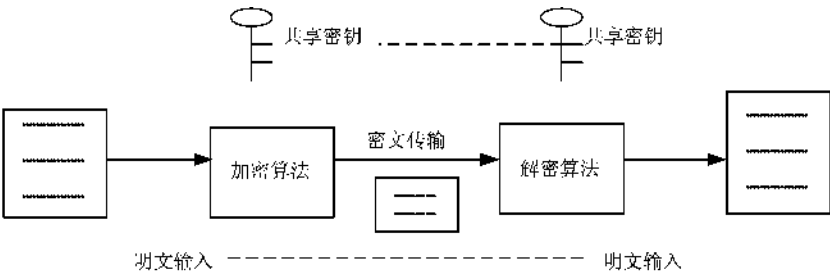


图 2-1-2 对称加密体系

为了安全地使用对称加密方法，有以下两点要求：

- （1）需要一个强壮的加密算法。要求即使有人知道了该算法，并且已经得到了一个或多个密文，也无法解密出密文或计算出密钥。
- （2）密钥的管理是非常重要的。信息的发送者和接收者必须以安全的方式获得并保存共享密钥。一旦密钥被别人截获，便失去了安全性。

非对称加密

非对称加密也称为公钥加密算法。在公钥加密算法中明文、加密算法、密文和解密算法的定义与对称加密一样，主要的区别在于密钥。在公钥加密体系中，需要一对密钥，一个被称为公钥，一个被称为私钥。应注意的是，公钥是对其他用户公开的，而私钥只有用户自己知道。这两个密钥一个用于加密，另一个用于解密。根据应用背景的不同，加密算法既可能是使用公钥，也可能是使用私钥；解密算法也一样。

公钥加密的方法如图 2-1-3 所示。

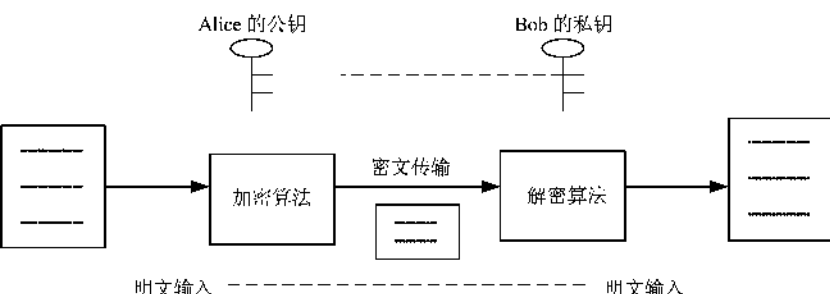


图 2-1-3 公钥加密体系

这里举一个 Alice 和 Bob 通信的例子。如果 Alice 想给 Bob 发送一个秘密的信息，并且他

知道月鑒所公开出来的公钥,就使用这个公钥对信息进行加密,产生密文后发给月鑒。当月鑒收到密文之后,利用自己的私钥对它进行解密得到明文信息。由于这一私钥只有月鑒自己知道,所以其他人是无法读到这个信息的。

自从 1976 年公钥密码的思想提出以来,国际上已经提出了许多种公钥密码体制,这些加密方法的安全性都是基于复杂的数学难题。对于某种数学难题,如果利用通用的算法计算出密钥的时间越长,那么基于这一数学难题的公钥密码体制就被认为越安全。当前比较流行的主要有两类:一类是基于大整数因子分解问题的,其中最典型的代表是 砸粤(三位科学家 砸鄢、增城、杂群到 粤望提出的首字母组合);另一类是基于离散对数问题的,比如 耘颢葬葬公钥密码和影响比较大的椭圆曲线公钥密码。当前比较著名的公钥密码算法有:砸粤、背包密码、酝耘葬葬密码、零知识证明的算法、椭圆曲线、耘颢葬葬算法等等。最有影响的公钥密码算法是 砸粤,它能抵抗到目前为止已知的所有密码攻击。

猿援对称加密算法和非对称加密算法的比较

对称加密算法的优点是有很强的保密强度,且经受住时间的检验和攻击,但其密钥必须通过安全的途径传送。因此,其密钥管理成为系统安全的重要因素。公钥密码的优点是可以适应网络的开放性要求,且密钥管理问题较为简单,尤其可方便地实现数字签名和验证。但其算法复杂,加密数据的速率较低。尽管如此,随着现代电子技术和密码技术的发展,公钥密码算法将是一种很有前途的网络安全加密体制。目前国际上公认的对称密码系统的密钥长度应达到 员颢比特。

人们在实际应用中通常将对称加密算法和公钥密码结合在一起使用,比如利用 陨砒或者 陨颢来加密信息,而采用 砸粤来传递会话密钥。

员颢援保护

多道程序设计技术的出现使操作系统的用户之间可以共享计算机软件和硬件资源,因此要求操作系统对共享资源提供保护。操作系统可能提供的保护层次主要有:

- ① 无保护。当敏感的过程是在独立的时间内运行时,这种方法是合适的。
- ② 隔离。这种方法意味着每个进程并不感觉到其他进程的存在。
- ③ 资源或通信行为。每个进程具有自己的地址空间、文件和其他对象。进程之间通过资源通信机制共享资源。
- ④ 共享或不共享。通过相关的机制宣布某个资源为公有的或私有的。公有的资源任何进程可以访问该对象;而私有的资源,只有所有者的进程才能访问该对象。
- ⑤ 通过访问控制的共享。操作系统检查特定用户对特定对象的访问的许可。操作系统就像位于用户和对象之间的护卫,保证只能发生已授权的访问。
- ⑥ 限制对象的使用。这种形式的保护不仅限制了对对象的访问,而且还限制存取后对对象的使用方式。

一个特定的操作系统可能对不同的对象、用户或应用程序提供不同程度的保护。下面介绍操作系统对一些对象实行保护的机制。

存储保护

对于一个安全操作系统，存储保护是一个最基本的要求，如果操作系统不提供内存存储器的保护，一个进程将能够不经意地写到另一个进程的存储空间，则后一个进程就可能不会正确地执行。存储保护负责保证系统各个任务之间互不干扰。这主要是指保护用户在存储器中的数据。保护单元为存储器中的最小数据范围，可为字、字块、页面或段。保护单元越小，则存储保护精度越高。对于代表单个用户，单任务的系统，存储保护机制应该防止用户程序对操作系统的影响。在允许多道程序并发运行的多任务操作系统中，还进一步要求存储保护机制对进程的存储区域实行互相隔离。

操作的存储保护与存储管理紧密相连。各个进程的存储空间的分隔可以很容易地通过虚存的方法来实现分段、分页，或两者的结合，提供了管理主存的有效方法。如果进程完全隔离，则操作系统必须保证每个段或页只能由所属的进程来控制 and 存取，这可以简单地通过要求在页表或段表中没有相同表项来实现。

因此操作系统的存储管理功能的页式、段式和段页式管理在一定程度上满足了操作系统内存存储器保护的需要。

访问控制

在数据处理系统中访问控制所采取的方法有两类：与用户有关的和与数据有关的。

面向用户的访问控制

用户登录是用户访问控制的最普遍的技术。它需要一个用户标识符（**用户名**）和一个口令。如果用户的 **用户名** 和口令和系统存储的一致，那么系统将允许该用户登录。通常，这种 **用户名/口令** 系统是用户访问控制的一种很差的不可靠的用户控制方法。主要原因有用户可能会忘记、无意或有意地泄漏其口令。所以，**用户名/口令** 文件容易遭到攻击。

在分布式环境中用户的访问控制可以是集中式，也可以是分散式的。使用集中式方法时，网络提供登录服务，用来确定允许哪些用户使用网络以及用户可以与哪些人连接。分散式用户访问控制将网络当作一个透明的通信链路，正常的登录过程由目的主机来执行。当然，在网络上传送口令的安全性也是必须解决的。

通过用户访问控制过程，系统对用户进行验证。有一个权限表与每个用户相关，指明用户被许可的合法操作和文件访问，操作系统就基于用户权限表进行访问控制。

面向数据的访问控制

面向数据的访问控制是用户成功登录以后，操作系统对数据提供的保护。用户进入计算机系统后有权访问一台或一组计算机及应用信息，但是并不是所有的数据用户都可访问。例如，允许管理员查询公司职员名单，但是只有特定的人员才能修改工资信息，这个问题需要更多级细节层次。尽管操作系统可能授权给一个用户存取文件或使用一个应用，但在这之后不再有进一步的安全性检查；但数据库管理系统必须针对每个独立访问企图做出决定，该决定将不仅取决于用户身份，而且取决于被访问的特定部分的数据，甚至取决于已公开给用户的信息。

文件或数据库管理系统采用的访问控制的一个通用模型是访问矩阵（**Access Matrix**），该模型的基本要素如下：

- ① 主体(主体)。能够访问计算机资源的实体。通常,主体概念等同于进程。
- ② 客体(客体)。被访问的客体(也称对象)。
- ③ 访问权(访问权)。主体对客体访问的方式,如读、写、执行、删除等。

通常这种功能由操作系统提供访问控制表、访问控制列表和权能表、访问控制列表等数据结构通过系统调用来完成。

4.1.2 操作系统的保护

操作系统的保护也是操作系统需要保护的主体。

操作系统的保护功能是操作系统的所有功能中最为复杂的。许多黑客常常从操作系统的部分寻找操作系统安全方面的缺陷并进行攻击。通常对操作系统的操作是一个特权操作,由高层系统调用来实现,在这些过程中,用户不需要控制操作系统的操作的细节。

由于所有的操作不是向设备写数据就是从设备接收数据,所以一个进行操作操作的进程必须受到对设备的读写两种访问控制。这就意味着设备到介质间的路径可以不受什么约束,而处理器到设备间的路径则需要施以一定的读写访问控制。

但是若要对系统中的信息提供足够的保护,防止被未授权用户的滥用或毁坏,只靠硬件不能提供充分的保护手段,必须由操作系统的安全机制与适当的硬件相结合才能提供强有力的保护。

4.1.3 操作系统的安全机制

操作系统是连接硬件与其他应用软件之间的桥梁,它需要提供一系列的安全服务来实现操作系统的主要目标。

操作系统安全的主要目标包括以下几个方面:

- ① 依据系统安全策略对用户的操作进行存取控制;
- ② 防止用户对计算机资源的非法存取(窃取、篡改和破坏);
- ③ 标识系统中的用户并进行身份鉴别;
- ④ 监督系统运行的安全性;
- ⑤ 保证系统自身的安全性和完整性。

为了实现这些目标,需要建立相应的安全机制,包括硬件安全机制、加密机制、认证机制、标识与鉴别、存取控制、安全审计等。

4.1.4 操作系统的认证机制

认证机制(认证机制)是安全操作系统应具备的基本机制,是用户欲进入系统访问资源或网络中通信双方进行数据传输之前实施审查和证实身份的操作。因而,身份认证机制成为大多数保护机制的基础。

认证机制有两种类型的认证:外部认证和内部认证。外部认证确定一个用户是否是被要求的用户,例如,如果一个人使用一个特殊的账户名登录到系统中,那么基本的外部认证机制检查并确认使用账户登录的人是否就是实际的用户。外部认证的最简单形式就是把每个账户与一个口令关联在一起,账户名可能大家都知道,例如它可以是一个电子邮件地址,而口令是使用该账户的人的一个秘密;口令是一个静态实体,并且只能通过用户本人或系统管理

员进行改变。操作系统机制中支持这种类型的认证，确保不可能通过一系列的暗中活动来绕过认证机制。

内部认证必须确保一个进程不能表现为像另一进程一样。如果没有内部认证，一个用户就可以创建一个进程，且让它表现为另一用户的进程，这时即使有最有效的外部认证机制，由于进程以另一个用户的身份运行，所以可以很容易地绕过认证。

员因源因瑶授权机制

用户通过认证进入计算机系统后，通过操作系统提供的授权机制（~~粤~~）使用计算机资源。

授权机制确保只有在策略许可的情况下用户和进程才被允许使用计算机的实体(如资源)。授权机制是建立在安全认证机制基础上的。图 5-10 很好地说明了计算机授权访问的几个过程。当一个用户试图使用计算机时,一个外部授权访问机制首先认证用户身份,然后查看用户是否被授权来访问计算机。

交互计算机中惯用的方法是让系统保存一份所有授权用户的记录，一个用户采用能够认证他的授权身份来使用计算机，如图 8-10 所示，用户在登录到机器并随后试图登录到一个远程机器时，必须首先被授权使用一个机器，对远程机器的访问可能使用另一种授权形式。

如果一个用户已被授权使用一个机器，该机器中的操作系统将分配一个代表用户的进程来完成用户活动（在 Windows 中，这就是用户的登录进程）。在登录授权完成后，用户就可以自由地指挥在线命令解释器（进程）来试图访问任意的资源，例如，一个用户可以试图编辑系统的口令文件，如图 5.10 所示，进程的每个访问，必须通过系统的内部访问授权机制来授权对目标资源的访问。外部授权访问机制和内部授权访问机制是不同的，外部机制授权用户访问计算机，而内部机制授权进程访问资源。只有在一个交互会话刚开始时，外部机制才发挥作用。然而内部机制在每次用户输入命令到系统内时，都可能起作用。这是因为大多数命令总是存储在文件中，当一个命令被执行时，要对文件进行读写。

资源浪费审计机制

操作系统应该提供事后追踪手段来保证系统的安全性。这就是审计机制。

审计机制（粤信安[2019]1号）对涉及系统安全性的操作做完整的记录，并形成审计日志，以备有违反系统安全规则的事件发生后能有效地追查事件发生的地点、时间、类型、过程、结果和涉及的用户。

操作系统必须能够生成、维护及保护审计过程，使其免遭修改、非法访问及毁坏，并要保护审计数据，要严格限制未经授权的用户访问它。

审计机制是对系统安全性的一种技术措施,也是对付计算机犯罪者们的利器。实现审计机制,首先要解决的问题是系统如何才能保证所有与安全相关的事件都能够被审计。在一般的多用户多进程操作系统(如 UNIX、Windows 等)中,用户程序与操作系统的惟一接口是系统调用。因此,如果能够找到系统调用的总入口,在此处(称作审计点)增加审计控制,就可以成功地审计。如果将审计和报警功能结合起来,那就可以做到在违反系统安全的事件或者有涉及系统安全的重要操作进行时,能及时向安全操作员终端发送相应的报警,系统的安全性能更高。

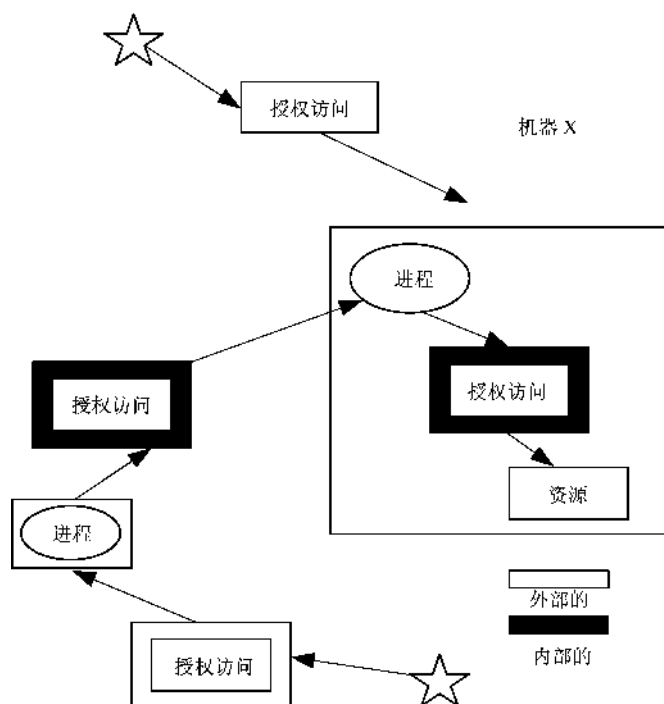


图 4-1-1 授权机制的实现

4.1.1 多用户操作系统的安全机制

多用户操作系统是由美国贝尔实验室开发的一种多用户、多任务的通用操作系统。这类操作系统的一种基本功能就是防止使用同一台计算机的不同用户之间互相干扰，所以多用户操作系统的设计宗旨特别是考虑安全的。

多用户操作系统经过 40 多年的发展已经成为具有高可靠性的操作系统，其可靠性主要有标识与口令安全、存取控制、鉴别、审计、网络安全和加密等六个方面。

多用户操作系统借助以下三种方式提供安全服务功能。

① 系统调用：用户进程通过多用户操作系统的内核部分——系统调用接口，显式地从内核获得服务。内核以调用进程的身份执行这些请求。

② 异常：进程的某些不正常操作，诸如除数为 0 或用户堆栈溢出将引起硬件异常。异常需要内核干预，内核为进程处理这些异常。

③ 中断：内核处理外围设备的中断。设备通过中断机制通知内核，内核完成状态变化。内核将中断视为全局事件，与任何特定进程都不相关。

从系统用户角度分析来看，多用户操作系统的安全问题主要由某些特定的命令引发。这些命令涉及口令、文件保护、目录保护、用户程序、加密以及网络安全等各个方面。

4.1.2 标识与口令安全

多用户操作系统的各种管理功能都被限制在一个超级用户（root）中，其功能和权限与普通用户的管理

员(粤差务器控制或 系统管理)的超级用户(杂差务器控制)功能类似。作为超级用户可以控制一切,包括用户账号、文件和目录、网络资源。哉差务器允许超级用户管理所有资源的各类变化。例如每个账号都是具有不同用户名、不同的口令和不同的访问权限的一个单独实体。这样就允许该用户有权授予或拒绝任何用户、用户组合以及所有用户的访问。用户可以生成自己的文件,安装自己的程序等。为了确保次序,系统会分配好用户目录。每个用户都得到一个主目录和一块硬盘空间。这块空间与系统区域和其他用户占用的区域分割开来。这种作用可以防止一般用户的活动影响其他文件系统,进而系统还为每个用户提供一定程度的保密。作为根可以控制哪些用户能够进行访问以及他们可以把文件存放在哪里。控制用户能够访问哪些资源,用户如何进行访问等。

用户登录到系统中时，需输入用户名标识其身份。在系统内部具体实现中，当该用户的账户创建时，系统管理员便为其分配一个惟一的标识号——**UUID**。

系统中的 `/etc/passwd` 文件含有全部系统需要知道的关于每个用户的信息（加密后的口令也可能存于 `/etc/shadow` 文件中）。`/etc/passwd` 中包含有用户的登录名，经过加密的口令、用户号、用户组号、用户注释、用户主目录和用户所用的 `login` 程序。其中用户号（`uid`）和用户组号（`gid`）用于 `UNIX` 系统惟一地标识用户和同组用户及其他用户的访问权限。`/etc/passwd` 中存放的加密口令与用户登录时输入的口令经计算比较后，符合则允许登录。否则拒绝用户登录，用户可用 `passwd` 命令修改自己的口令。`/etc/passwd` 文件中的口令内容不是直接的用户口令，而是经过加密后的口令。

总线缓存存取控制

在**哉**操作系统中，控制文件和信息存在磁盘及其他辅助存储介质上。它控制每个用户可以访问何种信息及如何访问，表现为通过一组存取控制规则来确定一个主体是否可以存取一个指定客体。**哉**的存取控制机制通过文件系统实现。

缓存存取权限

命令还可列出文件(或目录)对系统内的不同用户所给予的存取权限。如

[illegible]

权限有 猿种：

- ① 则为允许读；
- ② 憎为允许写；
- ③ 曾为允许执行。

用户有猿种类型：

- ① 拥有者为该文件的属主；
- ② 组拥有者为该文件所属用户组中的用户，即同组用户；
- ③ 其他拥有者为除以上二者外的其他用户。

表示文件的属主具有读、写及执行权限(例如图 1-10 中的 **rw**)，同组用户允许读和执行操作，其他用户没有任何权限。在权限位中，“**原**”表示相应的存取权限不允许。

上述的授权模式同样适用于目录，用 `chmod` 列出时，目录文件的类型为 `drwxr-xr-x`。目录要有读许可，在目录中增删文件要有写许可，进入目录或将该目录作路径分量时要有执行许可，因此要使用任何一个文件，必须有该文件及找到该文件所在路径上所有目录分量的相应

许可。仅当要打开一个文件时，文件的许可才开始起作用，而 `chmod` 只要有目录的搜索和写许可，该命令不需要有关文件的许可，这一点应尤其注意。

在 `Unix` 系统中，每个进程都有真实 `UID`、真实 `GID`、有效 `UID` 及有效 `GID`。当进程试图访问文件时，核心将进程的有效 `UID`、`GID` 和文件的存取权限位中相应的用户和组相比较，决定是否赋予其相应权限。

4.1.1 改变权限

改变文件的存取权限可使用 `chmod` 命令，并以新权限和该文件名为参数。格式为：

`chmod [原权限] 存取权限 文件名`

合理的文件授权可防止偶然性地覆盖或删除文件（即使是属主自己）。改变文件的属主和组名可用 `chown` 和 `chgrp`，但修改后原属主和组员就无法修改回来了。

`Unix` 中相应还有 `chattr` 命令。若将此命令放入用户的 `profile` 文件，就可控制该用户后续所建文件的存取许可。`chattr` 命令与 `chmod` 命令的作用正好相反，它告诉系统在创建文件时不给予什么存取许可。

4.1.2 鉴别

用户名是个标识，它告诉计算机该用户是谁。而口令是个确认证据。用户登录系统时，需要输入口令来鉴别用户身份。当用户输入口令时，`Unix` 使用改进的 `MD5` 算法（通过调用 `crypt` 函数实现）对其加密，并将结果与存储在 `passwd` 文件或 `shadow` 数据库中的加密用户口令比较，若二者匹配，则说明该用户的登录合法，否则拒绝用户登录。

由于 `passwd` 文件对任何用户可读，故常成为口令攻击的目标。所以系统中常用 `shadow` 文件（`shadow` 来存储加密口令，并使其对普通用户不可读。

4.1.3 审计

大多数版本的 `Unix` 都具备的审计服务程序 `audit`，它可实现灵活配置、集中式管理。运行中需要对信息作登记的单个软件发送消息给 `audit`，根据配置（`audit` 配置文件），按照消息的来源和重要程度，这些消息可记录到不同的文件、设备或其他主机中。

审计结果常写在系统的日志文件中。丰富的日志为 `Unix` 的安全运行提供了保障。常见的日志文件有：

- ① `auth.log`：记录每个用户使用过的命令；
- ② `auth.log`：筛选出 `sshd`（系统自动呼叫部件）记录；
- ③ `lastlog`：记录用户最后一次成功登录时间和最后一次登录失败的时间；
- ④ `lastxlog`：不良的登录尝试记录；
- ⑤ `syslog`：记录输出到系统主控台以及由 `audit` 系统服务程序产生的信息；
- ⑥ `syslog`：记录 `syslog` 命令的使用情况；
- ⑦ `utmp`：记录当前登录的每个用户；
- ⑧ `utmpx`：扩展的 `utmp`；
- ⑨ `wtmp`：记录每一次用户登录和注销的历史信息，以及系统关和开；
- ⑩ `wtmpx`：扩展的 `wtmp`；
- ⑪ `debug`：记录使用外部介质（如软盘或光盘）出现的错误；

- ⑫ `passwd`：记录 `passwd` 的存取情况。

系统维护网络安全性

当前的 `Unix` 系统通常是运行在网络环境中的，默认支持 `TCP/IP` 协议。所以网络安全性也是操作系统所强调的一个不可分割的重要方面。网络安全性，主要是指通过防止本机或本网被非法入侵、访问，从而达到保护本系统可靠、正常运行的目的。`Unix` 操作系统可以对网络访问控制提供强有力的安全支持，主要方式是有选择地允许用户和主机与其他主机的连接。相关的配置文件有：

- ① `etc/passwd`：该文件，内容是系统提供哪些服务。
- ② `etc/inetd.conf`：该文件，其中罗列了端口号、协议和对应的名称。
- ③ `etc/hosts.allow` 和 `etc/hosts.deny`：由 `inetd` 和 `sshd` 两个文件控制。

它们可以使你很容易地控制哪些 IP 地址禁止登录，哪些允许登录。通过加入服务限制条件，可以更好地管理系统。系统在使用它们的时候，先检查前一个文件，从头到尾扫描，如果发现用户的相应记录标记，就给用户提供服务。如果没有找到记录，就像刚才一样扫描 `passwd` 文件，查看是否有禁止用户的标记。如果发现记录，就不给用户提供服务。如果仍然没有找到记录，则使用系统默认值——开放服务。

网上访问的常用工具有 `telnet`、`rsh`、`rlogin` 等网络操作命令，为了安全起见，对它们的使用必须加以限制。最简单而且最常用的方法是修改 `etc/inetd.conf` 中相应的服务端口号，从而达到对这类访问进行控制的目的。其他常见的网络服务还有 `ftp` 和 `rsync`，`rsync` 使网络上的主机可以共享文件。`rsync` 又称黄页服务，它可将网络上每台主机的配置文件集中到一个 `rsync` 服务器上实现，这些配置文件包括用户账号信息、组信息、别名等。

① 当远程使用 `telnet` 访问本系统时，`Unix` 系统首先检查用户名和密码，无误后查看 `etc/passwd` 文件（不受欢迎的 `telnet` 用户表），一旦其中包含登录所用用户名则自动拒绝连接，从而达到限制作用。因此，只要一机器内除匿名 `telnet` 以外的所有用户列入 `etc/passwd` 文件中，即使入侵者获得本机内正确的用户信息，也无法登录系统。此外，如果使用远程注册数据文件（`passwd` 文件）来配置 `telnet` 用户的存取安全性，需注意保密，防止泄露其他相关主机的信息。

② `Unix` 系统没有直接提供对 `telnet` 的控制。但 `etc/passwd` 是系统 `passwd` 文件，所有用户登录时必须首先执行它。故可修改该文件达到安全访问目的。

当然 `Unix` 中仍然存在很多安全问题，其新功能的不断纳入及安全机制的错误配置或错误使用，都可能带来很多问题。

系统维护加密

在 `Unix` 系统中设置加密系统是必要的。一个拥有超级用户权限的用户可以绕过文件系统的所有口令检查，虽然他的权限极大，但如果文件加密，在不知道密钥的情况下是无法解密文件的。

当前 `Unix` 系统中常使用的加密程序有以下几个：

- ① `crypt`：最初的 `Unix` 加密程序。
- ② `DES`：数据加密标准（`Data Encryption Standard`）在 `Unix` 上的应用。
- ③ `PGP`：由 `Phil Zimmermann` 开发的 `Pretty Good Privacy` 程序。

上述程序在 网络 上都有相应的实现。

加密可以提供一些点对点的加密方法，以保护传输中的数据的安全。一般情况下，当数据在网络中传输时，可能要经过许多网关。在这个过程中，数据很容易被窃取。加密可以采用对本地文件加密的方法，防止文件被非法访问，同时可保证文件的一致性，防止对文件的非法篡改，也可以一定程度地防止病毒、特洛伊木马等恶意程序。

习题

计算机系统受到的安全威胁主要体现在哪些方面？

分析计算机安全性问题产生的原因。

安全的计算机系统和网络通信提供的服务主要有哪些？

如何理解操作系统的概念？

比较对称加密和非对称加密两种加密技术。

操作系统提供的保护机制主要有哪些？

分析操作系统的安全机制。

了解 加密 常见的安全机制。

第 1 章 操作系统结构与

随着计算机技术的发展，操作系统的大小和复杂性也随着增加。在 1964 年投入使用时的系统大约包含 40 万个指令字；随后问世的系统拥有超过 100 万条的机器指令，当前的系统比这更多。操作系统的设计技术和软件结构也变得越来越重要。

1.1 操作系统结构

1.1.1 整体式结构

这是一种大杂烩，整个操作系统是一堆过程的集合，每个过程都可以任意调用其他过程。使用这种技术时，系统中的每个过程都有一个完好定义的接口，即入口参数和返回值，而且相互间的调用不受约束。

使用这种结构方式时，开发人员为了构造最终的目标操作系统，首先将一些独立的过程、或包含过程的文件进行编译，然后用链接程序将它们链接成一个单独的目标程序。从信息隐藏的观点看，它没有任何隐藏——每个过程对其他过程都是可见的。

这种系统中提供机器的两种运行模式：用户态和核心态。当用户程序中使用系统调用请求系统提供服务时，就进入到核心态中。操作系统确定所需要的服务序号，调用相应的服务过程，完成工作后，将控制返回给用户程序。

如果说这种结构的操作系统有层次的话，那就是所有过程可大致分为三层：最高层是主程序，它引用所需的服务过程；中间层是一组服务过程，它们执行系统调用；底层是一组公用过程，用来支持各服务过程。

1.1.2 分层式结构

对于运行百万到千万条代码的大型操作系统，仅有模块化程序设计是不够的，分层放置的概念和信息抽象用得越来越多。现代操作系统的层次按照复杂性、时间常数、抽象级将功能分开。这样就可以把系统看作是一系列的层，每一层执行操作系统所需要功能的相关子集，它依赖于下一个较低层，较低层执行更为原始的功能，并隐藏这些功能的细节。它还要给相邻的较高层提供服务。我们可以通过定义层使得改变一层时不需要改变其他层，把一个问题分解成几个更易于处理的子问题。

第一个按这种方式构造的操作系统是 1965 年由 和他的学生们建造的。该系统共有 7 层，如图 1.1 所示。

第 7 层负责处理机的分配，当发生中断或定时器到时进行进程切换，从而提供了基本的多道程序环境。

第 7 层	操作员
第 6 层	用户程序
第 5 层	输入输出管理
第 4 层	操作员和进程通信
第 3 层	内存和磁鼓管理
第 2 层	处理机分配和多道程序环境

图 1.1 操作系统的层次结构

第 5 层执行内存和磁鼓的管理，用来为进程分配内存空间和磁鼓上的空间。内存用完时则在磁鼓上分配空间用作交换。在第 5 层上，用户不用考虑其进程是在磁鼓上还是在内存中运行，系统能保证一旦某一页面需要访问时，它必定在内存中。

第 4 层处理每个进程和操作员控制台之间的通信。在第 4 层上，每个进程都有自己的操作员控制台。

第 3 层进行输入输出管理，管理 I/O 设备，对信息流缓冲。在第 3 层上，每个进程和 I/O 设备打交道时不必考虑其物理的细节。

第 2 层是用户程序层，用户程序不必考虑进程、内存、控制台或 I/O 设备等细节。

第 1 层是系统操作员进程。

这种分层的概念后来体现在 5 层操作系统中，但它不是层，而是一系列同心环，且内层环比外层环有更多的权力。当外环中的过程想调用内环过程时，它必须利用对等的系统调用。这种环机制的优点是易于扩充用户子系统。例如，教师可编写一个程序在 2 层环中运行，用来对学生写的程序进行测试和打分，而学生的程序运行在 1 层环上，所以学生无法改变他们的分数。

著名的 5 层操作系统 5 的核心层就采用层次结构，如图 5-1 所示。

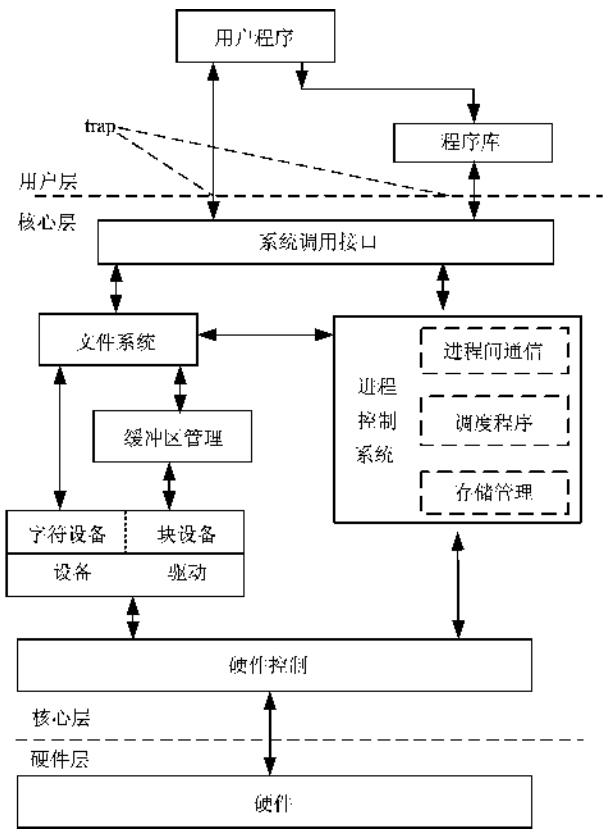


图 5-1 5 层操作系统 5 的核心图

从图 8-1 可以看出，操作系统核心的上层是系统调用的接口程序。在操作系统的内部，分为左右两大部分：左边是文件系统部分，右边是进程控制部分。底层是硬件控制部分。

左边部分的上层是文件系统，它管理文件，分配文件空间，控制对文件的访问等，调用下面的缓冲管理和设备管理模块。右边部分是进程控制系统，分为三个部分：进程通信、进程调度和内存管理。左右两个部分存在着密切的联系。

现在实际使用的操作系统多数都采用层次结构，它不但具有结构关系清晰等特点，而且能够提高系统的可靠性、可移植性和可维护性。

应当指出，在严格的分层方法中，任一层模块只能调用比它低的层来得到服务，而不能调用比它高的层。但是，在实际设计上这有很多的困难。所以，实际使用的操作系统的内部结构并非都符合这种层次模型。一个操作系统应划分多少层、各层处于什么位置、相互之间如何联系等并无固定的模式，必须根据系统总体功能设计和结构设计中的功能流图和数据流图进行分层。

微内核结构

分层结构中也存在问题。每层都处理相当多的功能，一层中的主要变化可能会产生巨大的影响，跟踪相邻层（上一层或下一层）中的代码有很多困难。其结果是，通过增加或减少一些功能，在基本操作系统上很难实现一个专用版本，并且由于在相邻层之间有很多交互，因而很难保证安全性。

随着操作系统的扩充，内核变得大且难以管理。在 20 世纪 80 年代中期，卡耐基梅隆大学的研究人员开发了一个称为 微内核的操作系统，该系统采用微内核（Microkernel）方法来模块化内核。这种方法将所有非基本部分从内核中移走，并将它们当作系统级程序和用户级程序来实现。用这种方法构建操作系统，结果是更小的内核。尽管什么应该在微内核中、什么应该在微内核外，不同的设计有不同的分界线，但是共同的特点是许多传统上属于操作系统一部分的功能现在都在外部子系统中，包括设备驱动程序、文件系统、虚存管理程序和安全服务等，它们可以与内核交互，也可以相互交互，其结构如图 8-2 所示。

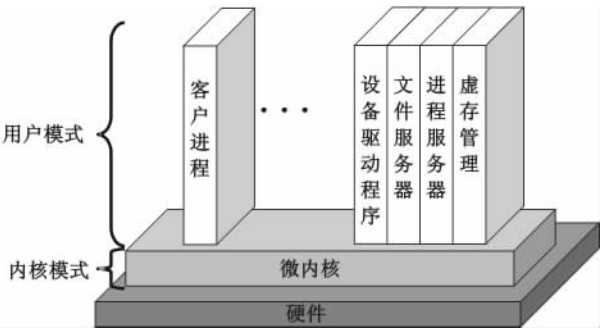


图 8-2 微内核结构

从图 8-2 可以看出，微内核结构用一个水平分层的结构代替了传统的纵向分层的结构。在微内核外部的操作系统部件被当作服务器进程实现，它们可以借助于通过微内核传递消息来实现相互之间的交互。因此，微内核起着信息交换的作用：验证信息、在部件间传递

信息并授权访问硬件。微内核还执行保护功能,除非允许交换,否则它阻止信息传递。例如,如果客户程序希望访问一个文件,那么它必须给文件服务器发消息;如果想创建一个进程或线程,则它给进程服务器发消息。每个服务器可以给别的服务器发消息,并且可以调用微内核中的原始功能,这就是单个计算机中的客户-服务器结构。

微内核结构可以很方便地扩充操作系统。所有新服务被增加到用户空间中,因而并不需要修改内核。当内核确实需要修改时,所做的改变也会很小,因为微内核本身很小。这样操作系统很容易从一种硬件平台移植到另一种平台设计。由于绝大多数服务是作为用户进程而不是作为内核进程来运行的,因此微内核也就提供了更好的安全性和可靠性。如果一个服务失败,那么操作系统的其他部分并不受影响。

许多现代操作系统使用了微内核方法。Windows NT(前身是 Windows 3.11)向用户提供了客户-服务器结构,但是它用混合内核来实现的。混合内核将客户-服务器系统调用映射成对适当用户层服务的消息。Windows 2000 服务器操作系统也是基于混合内核的。

5.2 用户接口设计

操作系统为用户提供了三类接口。第一类是命令接口,用户利用这些命令来组织和控制作业的执行;第二类是程序接口,编程人员可以使用它们来请求操作系统服务;第三类为图形用户接口,利用直观、易操作的图形方式控制或操纵程序运行。

5.2.1 命令接口

使用操作命令进行作业控制的主要方式有两种,即脱机控制方式和联机控制方式。

脱机控制方式是指用户将对作业的控制要求以作业控制说明书的方式提交给系统,由系统按照作业说明书的规定控制作业的执行。在作业执行过程中,用户无法干涉作业,只能等待作业执行结束之后才能根据结果信息了解作业的执行情况。

联机控制方式是指用户利用系统提供的一组键盘命令或其他操作命令和系统会话,交互式地控制程序的执行。其工作过程是用户在系统给出的提示符下键入特定命令,系统在执行该命令后向用户报告执行结果,然后用户决定下一步的操作,如此反复,直到作业执行结束。

相应地,可以将命令接口分为联机命令接口和脱机命令接口。

5.2.1.1 联机命令接口

联机命令接口又称为交互式命令接口,它由一组键盘操作命令组成。用户通过控制台或终端键入操作命令,向系统提出各种服务要求。用户每输入完一条命令,控制权就转入操作系统的命令解释程序,然后命令解释程序对键入的命令解释执行,完成指定的功能。之后,控制权又转回到控制台或终端,此时用户又可以键入下一条命令。

在微机操作系统中,通常把键盘命令分为内部命令和外部命令两大类。

(1) 内部命令:这类命令的特点是完成命令功能的程序短小,使用频繁。它们在操作系统初始启动时被引导至内存且常驻内存。

(2) 外部命令:完成这类命令功能的程序较长,并且各自独立地作为一个文件驻留在磁盘上。当需要它们时,从磁盘上调入内存运行。

操作系统为支持命令语言的解释执行,设置了一个命令解释程序(Shell)。它的主要作用是解释

来负责解释执行用户当前发出的命令。在分时操作系统实现中,命令解释程序属于操作系统之外的系统软件层,它运行于用户态下,作为一个进程来运行。分时系统的每个终端创建一个进程运行命令解释程序,该程序不断地读取它所控制的终端发来的命令。

当用户在终端上输入一条命令时,命令解释程序要做的工作如下:

(1) 判断命令的合法性

(2) 识别命令并执行处理命令的程序(可能向操作系统发出系统调用),继续读取下一条命令。

(3) 如果是不认识的命令关键字,则在约定目录下查找与命令关键字同名的执行文件,创建子进程执行“执行文件”程序,等待子进程结束后继续读取下一条命令。

命令动作完成后,命令解释程序在终端上显示提示符,允许用户键入新的命令。终端命令的一般形式如下:

`命令关键字 参数1 参数2 ... 参数n`

其中,命令关键字是命令关键字,参数1 参数2 ...,参数n是执行该命令的参数。

终端命令一般都是串行执行的,即用户键入的一条命令处理完后,系统发出新的提示符,用户继续键入下一条命令。若执行一条命令需要较长的处理时间而用户不需等待它的结果,就可以在该命令的末尾加上一个“开关”(分时系统是在末尾加“&”),将这条命令作为后台命令处理。用户可以接着键入下一条命令,系统可同时对前后两条命令做并行处理。

当然,命令解释程序也可以从文件中读取用户先前键入的命令。控制转移语句包含循环语句、条件转移语句等。这一类语句由命令解释程序直接处理,这类语句一般不在交互式输入命令时使用,而是在用户编写命令语言的程序时使用。

系统为用户提供了大量的实用程序,这些实用程序都是可以通过键入对应的终端命令而运行的。因为当用户键入命令解释程序不认识的命令关键字时,命令解释程序不是报错,而是去寻找与命令关键字同名的文件。所以,如果要运行某个实用程序,只要键入那个实用程序的执行文件名即可。

脱机命令接口

脱机命令接口也称批处理命令接口,它由一组作业控制命令(或称为作业控制语言)组成。脱机用户不能直接干预作业的运行,他们应事先用相应的作业控制命令写成一份作业操作说明书,连同作业一起提交给系统。当系统调度到该作业时,由系统中的命令解释程序对作业说明书上的命令或作业控制语句逐条解释。

系统调用程序接口

程序接口由一组系统调用命令组成。用户通过在程序中使用这些系统调用命令来请求操作系统提供的服务。用户在程序中可以直接使用这组系统调用命令向系统提出各种服务要求,如使用各种外部设备,进行有关磁盘文件的操作,申请分配和回收内存以及其他各种控制要求;也可以在程序中使用过程调用语句,编译程序将它们翻译成有关的系统调用命令,然后再去调用系统提供的各种功能或服务。

5.1 系统调用

所谓系统调用就是用户在程序中调用操作系统所提供的一些子功能。具体讲,系统调用就是通过系统调用命令中断现行程序,而转去执行相应的子程序,以完成特定的系统功能。完成后,控制权又返回到系统调用命令的逻辑后继指令,被中断的程序将继续执行下去。

实际上,系统调用命令不仅可以供用户程序使用,还可以供系统程序使用,以此实现各类系统功能。对于每个操作系统而言,其所提供的系统调用命令条数、格式以及所执行的功能等都不尽相同,即使是同一个操作系统,其不同版本所提供的系统调用命令条数也会有所增减。通常,一个操作系统提供的系统调用命令有几十条乃至上百条之多,它们各自有一个惟一的编号或助记符。这些系统调用按功能大致可分为如下几类:

(1) 设备管理。该类系统调用完成设备的请求或释放、以及设备启动等功能。

(2) 文件管理。该类系统调用完成文件的读、写、创建及删除等功能。

(3) 进程控制。该类系统调用完成进程的创建、撤销、阻塞及唤醒等功能。

(4) 进程通信。该类系统调用完成进程之间的消息传递或信号传递等功能。

(5) 内存管理。该类系统调用完成内存的分配、回收以及获取作业占用内存区大小及起始地址等功能。

系统调用命令是作为扩充机器指令提供的,目的是增强系统功能,方便用户使用。因此,在一些计算机系统中,把系统调用命令称为广义指令。广义指令与机器指令性质上是不同的,机器指令是用硬件线路直接实现的,而广义指令则是由操作系统提供的一个或多个子程序模块实现的。

5.1.1 系统调用的执行过程

虽然系统调用命令的具体格式因系统而异,但用户程序进入系统调用的步骤及其执行过程大体上是相同的。

通常,用户必须向系统调用命令处理程序提供必要的参数,以使其能根据这些参数进行相应的处理。当执行一个系统调用时,硬件把它作为一个软件中断对待,控制通过中断向量传递给操作系统的一个服务例程,改变用户方式到核心方式。它检查中断指令以决定出现了什么系统调用,所带参数决定了用户调用的服务类型。请求所需的信息用寄存器、堆栈进行参数传递。在验证参数的正确和合法性后,执行请求,去完成特定的系统调用功能。同时,这个被调用的系统子程序可能还要调用其他的子程序,完成更基本的子功能。当系统调用执行完成时,系统要把执行是否成功,以及成功时的执行结果回送给调用者。之后恢复处理机执行系统调用前的状态(用户态),返回用户程序继续执行,系统调用的执行过程如图 5-1 所示。

由图 5-1 中可以看出,在系统调用命令执行时,还可能要调用其他一些子程序来完成更基本的功能,从而形成系统功能的嵌套调用。

系统调用的一般过程可以概括为以下六个步骤。

(1) 通过软中断(或异常)使系统切换到核心模式,这需要硬件提供相应的支持。一旦进入核心态,就可以按特权方式运行。

(2) 程序计数器和处理机状态存入当前进程的堆栈中。

(3) 将系统调用号存入核心堆栈中。

(4) 执行汇编代码来保存通用寄存器的内容。

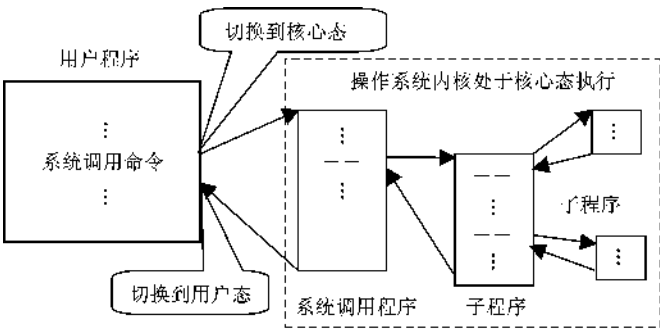


图 4-1 微内核结构

- ③ 调用相应的操作系统例程来完成系统调用。
- ④ 返回用户方式。

图形用户接口

通过命令接口方式来控制程序虽然有效，但给用户增加了很大的负担，即用户必须记住各种命令，并从键盘键入这些命令以及所需要的参数，以控制用户程序的运行。随着大屏幕高分辨率图形显示和多种交互式输入输出设备（比如鼠标、触摸屏）的出现，图形用户接口于 20 世纪 80 年代后期出现并广泛推广。

图形用户接口的目标是通过通过对出现在屏幕上的对象直接进行操作，以控制和操纵程序的运行。例如，用键盘或鼠标对菜单中的各种操作进行选择，使命令程序执行用户选定的操作；用户可以通过滑动滚动条上的滑动块在列表框中的选择项上滚动，以使所要的选择项出现在屏幕上，并用鼠标选取的方式来选择操作对象；用户还可以用鼠标拖动屏幕上的对象使其移动位置或旋转、放大和缩小。这种图形用户接口大大减少或免除了用户的记忆工作量。其操作方式从原来的记忆并键入改为选择并点取，极大地方便了用户，受到普遍欢迎。目前图形用户接口是最为常见的人机接口形式，可以认为图形用户接口是命令接口的图形化。

系统设计与实现

本节讨论一些设计和实现系统时所遇到的问题。虽然对这些问题没有完整的解决方案，但是有些方法还是很成功的。

设计目标

系统设计的第一个问题是定义系统的目标和规格。在最高层，系统设计受到硬件选择和系统类型的影响：批处理、分时、单用户、多用户、分布式、实时或通用目标。

除了最高设计层，有些要求可能难以描述。一般可以把需求分为两个基本类：用户目标和系统目标。用户要求一些明显的系统特性：系统应该方便和容易使用、容易学习、可靠、安全和快速。当然，这些规格说明对于系统设计并不特别有用，因为没有实现这些目标的通用协议。

设计、创建、维护和操作系统的有关人员可以定义另一组类似要求：操作系统应该容易设计、实现和维护，它应该灵活、可靠、没有错误且高效。同样，这些要求是含糊不清的，并没有通用的解决方案。关于定义操作系统的要求，没有一个唯一的解决方案。多种类型的系统说明了不同要求能形成对不同环境的各种解决方案。

机制与策略

操作系统的规格说明书和设计属于高度创造性工作。虽然没有教科书告诉你如何去做，但是一般的软件工程原理还是适用的，尤其是适用于操作系统。

一个重要的原理是机制和策略的区分。机制决定了如何做；策略决定做什么。例如，在操作系统中，定时器是一种确保进程保护的机制，但是对于特定用户将定时器设置成多长时间是个策略问题。

机制和策略的区分对于灵活性来说很重要。策略可能会随地点或时间而有所改变。在最坏的情况下，每次策略改变都可能需要底层机制的改变。系统需要通用机制。这种策略的改变只需要重新定义一些系统参数。例如，对于一个计算机系统，如果一个策略决定是时间密集型程序应该比 I/O 密集型程序有更高的优先级，如果机制是正确分开的且不依赖于策略，那么可以在其他计算机系统上很容易地制定相反的策略。

通过实现一组基本的原始的构造块，基于微内核的操作系统充分利用了机制与策略的区分。这些块几乎与策略无关，允许通过用户创建的内核模块或用户程序本身来增加更高级的机制和策略。另一种极端如 BSD 操作系统，该操作系统完全对机制和策略在系统内编码，以形成统一的系统风格。

实现

在设计操作系统后，就必须实现它。传统的操作系统是用汇编语言来编写的，现代的操作更偏向于高级语言，比如 C 语言。

不是用汇编语言编写的第一个系统可能是用于 IBM 计算机的主控程序（Monitor）。Monitor 是采用一种 Fortran 语言编写的，在 Monitor 开发的操作系统主要是用 Fortran 编写。用于 IBM 计算机的 OS/2 操作系统是用 C 语言编写的。Unix 操作系统、VMS 和 Windows 系统主要是用 C 编写的。在原来的 Unix 中，只有系统运行代码是用汇编语言来编写的，这些主要用于调度程序和设备驱动程序。

使用高级语言来实现操作系统，可以得到与用高级语言编写应用程序同样的优点：代码编写更快，更为紧凑，更容易理解和调试。另外，编译技术的改进使得只要通过重新编译就可改善整个操作系统的生成代码。最后，如果用高级语言来编写，从一个硬件到另一个硬件，操作系统更容易移植。比如 Unix 操作系统，主要是用 C 来编写，可用于不同平台上，如 IBM 系统、VAX、Alpha、MIPS、PowerPC、SPARC、SUN、Intel、ARM、M68K、M68000、M68010、M68020、M68030、M68040、M68050、M68060、M68070、M68080、M68090、M68095、M68100、M68101、M68102、M68103、M68104、M68105、M68106、M68107、M68108、M68109、M68110、M68111、M68112、M68113、M68114、M68115、M68116、M68117、M68118、M68119、M68120、M68121、M68122、M68123、M68124、M68125、M68126、M68127、M68128、M68129、M68130、M68131、M68132、M68133、M68134、M68135、M68136、M68137、M68138、M68139、M68140、M68141、M68142、M68143、M68144、M68145、M68146、M68147、M68148、M68149、M68150、M68151、M68152、M68153、M68154、M68155、M68156、M68157、M68158、M68159、M68160、M68161、M68162、M68163、M68164、M68165、M68166、M68167、M68168、M68169、M68170、M68171、M68172、M68173、M68174、M68175、M68176、M68177、M68178、M68179、M68180、M68181、M68182、M68183、M68184、M68185、M68186、M68187、M68188、M68189、M68190、M68191、M68192、M68193、M68194、M68195、M68196、M68197、M68198、M68199、M68200、M68201、M68202、M68203、M68204、M68205、M68206、M68207、M68208、M68209、M68210、M68211、M68212、M68213、M68214、M68215、M68216、M68217、M68218、M68219、M68220、M68221、M68222、M68223、M68224、M68225、M68226、M68227、M68228、M68229、M68230、M68231、M68232、M68233、M68234、M68235、M68236、M68237、M68238、M68239、M68240、M68241、M68242、M68243、M68244、M68245、M68246、M68247、M68248、M68249、M68250、M68251、M68252、M68253、M68254、M68255、M68256、M68257、M68258、M68259、M68260、M68261、M68262、M68263、M68264、M68265、M68266、M68267、M68268、M68269、M68270、M68271、M68272、M68273、M68274、M68275、M68276、M68277、M68278、M68279、M68280、M68281、M68282、M68283、M68284、M68285、M68286、M68287、M68288、M68289、M68290、M68291、M68292、M68293、M68294、M68295、M68296、M68297、M68298、M68299、M68300、M68301、M68302、M68303、M68304、M68305、M68306、M68307、M68308、M68309、M68310、M68311、M68312、M68313、M68314、M68315、M68316、M68317、M68318、M68319、M68320、M68321、M68322、M68323、M68324、M68325、M68326、M68327、M68328、M68329、M68330、M68331、M68332、M68333、M68334、M68335、M68336、M68337、M68338、M68339、M68340、M68341、M68342、M68343、M68344、M68345、M68346、M68347、M68348、M68349、M68350、M68351、M68352、M68353、M68354、M68355、M68356、M68357、M68358、M68359、M68360、M68361、M68362、M68363、M68364、M68365、M68366、M68367、M68368、M68369、M68370、M68371、M68372、M68373、M68374、M68375、M68376、M68377、M68378、M68379、M68380、M68381、M68382、M68383、M68384、M68385、M68386、M68387、M68388、M68389、M68390、M68391、M68392、M68393、M68394、M68395、M68396、M68397、M68398、M68399、M68400、M68401、M68402、M68403、M68404、M68405、M68406、M68407、M68408、M68409、M68410、M68411、M68412、M68413、M68414、M68415、M68416、M68417、M68418、M68419、M68420、M68421、M68422、M68423、M68424、M68425、M68426、M68427、M68428、M68429、M68430、M68431、M68432、M68433、M68434、M68435、M68436、M68437、M68438、M68439、M68440、M68441、M68442、M68443、M68444、M68445、M68446、M68447、M68448、M68449、M68450、M68451、M68452、M68453、M68454、M68455、M68456、M68457、M68458、M68459、M68460、M68461、M68462、M68463、M68464、M68465、M68466、M68467、M68468、M68469、M68470、M68471、M68472、M68473、M68474、M68475、M68476、M68477、M68478、M68479、M68480、M68481、M68482、M68483、M68484、M68485、M68486、M68487、M68488、M68489、M68490、M68491、M68492、M68493、M68494、M68495、M68496、M68497、M68498、M68499、M68500、M68501、M68502、M68503、M68504、M68505、M68506、M68507、M68508、M68509、M68510、M68511、M68512、M68513、M68514、M68515、M68516、M68517、M68518、M68519、M68520、M68521、M68522、M68523、M68524、M68525、M68526、M68527、M68528、M68529、M68530、M68531、M68532、M68533、M68534、M68535、M68536、M68537、M68538、M68539、M68540、M68541、M68542、M68543、M68544、M68545、M68546、M68547、M68548、M68549、M68550、M68551、M68552、M68553、M68554、M68555、M68556、M68557、M68558、M68559、M68560、M68561、M68562、M68563、M68564、M68565、M68566、M68567、M68568、M68569、M68570、M68571、M68572、M68573、M68574、M68575、M68576、M68577、M68578、M68579、M68580、M68581、M68582、M68583、M68584、M68585、M68586、M68587、M68588、M68589、M68590、M68591、M68592、M68593、M68594、M68595、M68596、M68597、M68598、M68599、M68600、M68601、M68602、M68603、M68604、M68605、M68606、M68607、M68608、M68609、M68610、M68611、M68612、M68613、M68614、M68615、M68616、M68617、M68618、M68619、M68620、M68621、M68622、M68623、M68624、M68625、M68626、M68627、M68628、M68629、M68630、M68631、M68632、M68633、M68634、M68635、M68636、M68637、M68638、M68639、M68640、M68641、M68642、M68643、M68644、M68645、M68646、M68647、M68648、M68649、M68650、M68651、M68652、M68653、M68654、M68655、M68656、M68657、M68658、M68659、M68660、M68661、M68662、M68663、M68664、M68665、M68666、M68667、M68668、M68669、M68670、M68671、M68672、M68673、M68674、M68675、M68676、M68677、M68678、M68679、M68680、M68681、M68682、M68683、M68684、M68685、M68686、M68687、M68688、M68689、M68690、M68691、M68692、M68693、M68694、M68695、M68696、M68697、M68698、M68699、M68700、M68701、M68702、M68703、M68704、M68705、M68706、M68707、M68708、M68709、M68710、M68711、M68712、M68713、M68714、M68715、M68716、M68717、M68718、M68719、M68720、M68721、M68722、M68723、M68724、M68725、M68726、M68727、M68728、M68729、M68730、M68731、M68732、M68733、M68734、M68735、M68736、M68737、M68738、M68739、M68740、M68741、M68742、M68743、M68744、M68745、M68746、M68747、M68748、M68749、M68750、M68751、M68752、M68753、M68754、M68755、M68756、M68757、M68758、M68759、M68760、M68761、M68762、M68763、M68764、M68765、M68766、M68767、M68768、M68769、M68770、M68771、M68772、M68773、M68774、M68775、M68776、M68777、M68778、M68779、M68780、M68781、M68782、M68783、M68784、M68785、M68786、M68787、M68788、M68789、M68790、M68791、M68792、M68793、M68794、M68795、M68796、M68797、M68798、M68799、M68800、M68801、M68802、M68803、M68804、M68805、M68806、M68807、M68808、M68809、M68810、M68811、M68812、M68813、M68814、M68815、M68816、M68817、M68818、M68819、M68820、M68821、M68822、M68823、M68824、M68825、M68826、M68827、M68828、M68829、M68830、M68831、M68832、M68833、M68834、M68835、M68836、M68837、M68838、M68839、M68840、M68841、M68842、M68843、M68844、M68845、M68846、M68847、M68848、M68849、M68850、M68851、M68852、M68853、M68854、M68855、M68856、M68857、M68858、M68859、M68860、M68861、M68862、M68863、M68864、M68865、M68866、M68867、M68868、M68869、M68870、M68871、M68872、M68873、M68874、M68875、M68876、M68877、M68878、M68879、M68880、M68881、M68882、M68883、M68884、M68885、M68886、M68887、M68888、M68889、M68890、M68891、M68892、M68893、M68894、M68895、M68896、M68897、M68898、M68899、M68900、M68901、M68902、M68903、M68904、M68905、M68906、M68907、M68908、M68909、M68910、M68911、M68912、M68913、M68914、M68915、M68916、M68917、M68918、M68919、M68920、M68921、M68922、M68923、M68924、M68925、M68926、M68927、M68928、M68929、M68930、M68931、M68932、M68933、M68934、M68935、M68936、M68937、M68938、M68939、M68940、M68941、M68942、M68943、M68944、M68945、M68946、M68947、M68948、M68949、M68950、M68951、M68952、M68953、M68954、M68955、M68956、M68957、M68958、M68959、M68960、M68961、M68962、M68963、M68964、M68965、M68966、M68967、M68968、M68969、M68970、M68971、M68972、M68973、M68974、M68975、M68976、M68977、M68978、M68979、M68980、M68981、M68982、M68983、M68984、M68985、M68986、M68987、M68988、M68989、M68990、M68991、M68992、M68993、M68994、M68995、M68996、M68997、M68998、M68999、M69000、M69001、M69002、M69003、M69004、M69005、M69006、M69007、M69008、M69009、M69010、M69011、M69012、M69013、M69014、M69015、M69016、M69017、M69018、M69019、M69020、M69021、M69022、M69023、M69024、M69025、M69026、M69027、M69028、M69029、M69030、M69031、M69032、M69033、M69034、M69035、M69036、M69037、M69038、M69039、M69040、M69041、M69042、M69043、M69044、M69045、M69046、M69047、M69048、M69049、M69050、M69051、M69052、M69053、M69054、M69055、M69056、M69057、M69058、M69059、M69060、M69061、M69062、M69063、M69064、M69065、M69066、M69067、M69068、M69069、M69070、M69071、M69072、M69073、M69074、M69075、M69076、M69077、M69078、M69079、M69080、M69081、M69082、M69083、M69084、M69085、M69086、M69087、M69088、M69089、M69090、M69091、M69092、M69093、M69094、M69095、M69096、M69097、M69098、M69099、M69100、M69101、M69102、M69103、M69104、M69105、M69106、M69107、M69108、M69109、M69110、M69111、M69112、M69113、M69114、M69115、M69116、M69117、M69118、M69119、M69120、M69121、M69122、M69123、M69124、M69125、M69126、M69127、M69128、M69129、M69130、M69131、M69132、M69133、M69134、M69135、M69136、M69137、M69138、M69139、M69140、M69141、M69142、M69143、M69144、M69145、M69146、M69147、M69148、M69149、M69150、M69151、M69152、M69153、M69154、M69155、M69156、M69157、M69158、M69159、M69160、M69161、M69162、M69163、M69164、M69165、M69166、M69167、M69168、M69169、M69170、M69171、M69172、M69173、M69174、M69175、M69176、M69177、M69178、M69179、M69180、M69181、M69182、M69183、M69184、M69185、M69186、M69187、M69188、M69189、M69190、M69191、M69192、M69193、M69194、M69195、M69196、M69197、M69198、M69199、M69200、M69201、M69202、M69203、M69204、M69205、M69206、M69207、M69208、M69209、M69210、M69211、M69212、M69213、M69214、M69215、M69216、M69217、M69218、M69219、M69220、M69221、M69222、M69223、M69224、M69225、M69226、M69227、M69228、M69229、M69230、M69231、M69232、M69233、M69234、M69235、M69236、M69237、M69238、M69239、M69240、M69241、M69242、M69243、M69244、M69245、M69246、M69247、M69248、M69249、M69250、M69251、M69252、M69253、M69254、M69255、M69256、M69257、M69258、M69259、M69260、M69261、M69262、M69263、M69264、M69265、M69266、M69267、M69268、M69269、M69270、M69271、M69272、M69273、M69274、M69275、M69276、M69277、M69278、M69279、M69280、M69281、M69282、M69283、M69284、M69285、M69286、M69287、M69288、M69289、M69290、M69291、M69292、M69293、M69294、M69295、M69296、M69297、M69298、M69299、M69300、M69301、M69302、M69303、M69304、M69305、M69306、M69307、M69308、M69309、M69310、M69311、M69312、M69313、M69314、M69315、M69316、M69317、M69318、M69319、M69320、M69321、M69322、M69323、M69324、M69325、M69326、M69327、M69328、M69329、M69330、M69331、M69332、M69333、M69334、M69335、M69336、M69337、M69338、M69339、M69340、M69341、M69342、M69343、M69344、M69345、M69346、M69347、M69348、M69349、M69350、M69351、M69352、M69353、M69354、M69355、M69356、M69357、M69358、M69359、M69360、M69361、M69362、M69363、M69364、M69365、M69366、M69367、M69368、M69369、M69370、M69371、M69372、M69373、M69374、M69375、M69376、M69377、M69378、M69379、M69380、M69381、M69382、M69383、M69384、M69385、M69386、M69387、M69388、M69389、M69390、M69391、M69392、M69393、M69394、M69395、M69396、M69397、M69398、M69399、M69400、M69401、M69402、M69403、M69404、M69405、M69406、M69407、M69408、M69409、M69410、M69411、M69412、M69413、M69414、M69415、M69416、M69417、M69418、M69419、M69420、M69421、M69422、M69423、M69424、M69425、M69426、M69427、M69428、M69429、M69430、M69431、M69432、M69433、M69434、M69435、M69436、M69437、M69438、M69439、M69440、M69441、M69442、M69443、M69444、M69445、M69446、M69447、M69448、M69449、M69450、M69451、M69452、M69453、M69454、M69455、M69456、M69457、M69458、M69459、M69460、M69461、M69462、M69463、M69464、M69465、M69466、M69467、M69468、M69469、M69470、M69471、M69472、M69473、M69474、M69475、M69476、M69477、M69478、M69479、M69480、M69481、M69482、M69483、M69484、M69485、M69486、M69487、M69488、M69489、M69490、M69491、M69492、M69493、M69494、M69495、M69496、M69497、M69498、M69499、M69500、M69501、M69502、M69503、M69504、M69505、M69506、M69507、M69508、M69509、M69510、M69511、M69512、M69513、M69514、M69515、M69516、M69517、M69518、M69519、M69520、M69521、M69522、M69523、M69524、M69525、M69526、M69527、M69528、M69529、M69530、M69531、M69532、M69533、M69534、M69535、M69536、M69537、M69538、M69539、M69540、M69541、M69542、M69543、M69544、M69545、M69546、M69547、M69548、M69549、M69550、M69551、M69552、M69553、M69554、M69555、M69556、M69557、M69558、M69559、M69560、M69561、M69562、M69563、M69564、M69565、M69566、M69567、M69568、M69569、M69570、M69571、M69572、M69573、M69574、M69575、M69576、M69577、M69578、M69579、M69580、M69581、M69582、M69583、M69584、M69585、M69586、M69587、M69588、M69589、M69590、M69591、M69592、M69593、M69594、M69595、M69596、M69597、M69598、M69599、M69600、M69601、M69602、M69603、M69604、M69605、M69606、M69607、M69608、M69609、M69610、M69611、M69612、M69613、M69614、M69615、M69616、M69617、M69618、M69619、M69620、M69621、M69622、M69623、M69624、M69625、M69626、M69627、M69628、M69629、M69630、M69631、M69632、M69633、M69634、M69635、M69636、M69637、M69638、M69639、M69640、M69641、M69642、M69643、M69644、M69645、M69646、M69647、M69648、M69649、M69650、M69651、M69652、M69653、M69654、M69655、M69656、M69657、M69658、M69659、M69660、M69661、M69662、M69663、M69664、M69665、M69666、M69667、M69668、M69669、M69670、M69671、M69672、M69673、M69674、M69675、M69676、M69677、M69678、M69679、M69680、M69681、M69682、M69683、M69684、M69685、M69686、M69687、M69688、M69689、M69690、M69691、M69692、M69693、M69694、M69695、M69696、M69697、M69698、M69699、M69700、M69701、M69702、M69703、M69704、M69705、M69706、M69707、M69708、M69709、M69710、M69711、M69712、M69713、M69714、M69715、M69716、M69717、M69718、M6971

来说是很关键的。在系统编写完并能正确工作之后，可以找出瓶颈子程序，并用相应的汇编语言子程序来替代。

为了识别瓶颈，必须要能监视系统性能。必须增加代码以计算并显示系统行为的测量度。对于有的系统，操作系统通过生成系统行为的跟踪列表来执行这一任务。所有相关事件的时间和重要参数都记录下来，并写到文件。之后，分析程序能处理日志文件，以决定系统性能，并识别瓶颈和低效率。

系统操作性能

在评价一个操作系统时，可以从以下几个方面进行考虑。

系统可靠性

随着分时系统、计算机网络和分布式系统的发展和应用，计算机用户越来越多，系统资源的共享程度也提高了，操作系统的可靠性也变得更加重要。

计算机系统的用户都希望在一个稳定的、安全可靠的、不会产生错误的操作系统上工作，但现实中并不存在这种系统。其主要原因是因为操作系统中包含了大量的软、硬件，至今还没有一种设计和实施技术能够保证它们永远不发生故障，另外，系统的使用环境复杂多变，病毒、黑客攻击，系统操作员和一般用户的各种误操作等都会发生故障。若想有一个可靠性较好的操作系统，应在以下几个方面加以努力：

(1) 在设计和实施的各个环节采取各种技术，尽可能避免软硬件故障。

(2) 在系统运行过程中，一旦出错要能及时检测出来，以减少对系统造成的损害。

(3) 检测出错误后要能迅速找到造成错误的原因，确定故障位置并采取相应措施排除故障。

(4) 尽可能对错误造成的损害进行修复，使系统恢复正常运行。

系统效率

现在的操作系统在一些关键业务处理过程中发挥着重要作用，效率是非常重要的指标。目前还没有一个统一的标准以衡量系统的工作效率，常用指标是吞吐量、各种资源的使用效率、分时系统的响应时间、批处理的作业周转时间等。

系统可维护性

一个好的操作系统应该是便于维护的。在系统投入运行后，可能要经常进行维护，比如整理文件系统、更换硬件、进行故障排除等等。所有这些工作应当是能够比较容易地完成，为了做到这一点，一个好的操作系统应该提供较强的系统维护工具，而且要提供完整的帮助文档等。

本章小结

随着操作系统规模不断扩大，其复杂性也随之增加，操作系统的软件结构也越来越被重视，在操作系统的发展历程中，出现了整体结构、分层结构和微内核结构，它们各有特点。

操作系统提供给用户的接口主要分为命令接口、程序接口和图形用户接口，我们可以从设计目标、机制与策略、实现三个方向来设计一个系统。

习题

1. 操作系统的结构有哪几种？

2. 什么是系统调用？

3. 简述系统调用的执行过程。

4. 简述操作系统分层的主要思想。

5. 用户与操作系统之间的接口有哪几类？它们是在什么情况下使用的？

6. 操作系统主要有哪些系统调用？

7. 可以从哪些方面来评价操作系统的性能？

附录实验实验

操作系统是一门实践性很强的课程。除了要学习书本上的理论之外，还必须动手实践才能对操作系统基本原理真正理解。实验是为了通过模拟操作系统原理的实现，使学生能更深刻地领会操作系统的工作原理和实现方法，并初步掌握操作系统基本功能的设计方法及其实现过程。本附录简单介绍课程教学中课内常选实验的主要内容、目的和实验要求，仅供参考。

实验一摇处理机调度

实验内容

选择一个调度算法，实现处理机的调度。

实验目的

进程调度是处理机管理的核心内容。多道系统中，当就绪进程数大于处理机数时，就必须按照某种策略决定哪些进程优先占用处理机。本实验模拟实现处理机调度，以加深对进程控制块、进程队列、进程状态（后备、就绪、运行、挂起、解挂等）等概念的理解，从而较好地了解作业调度过程的工作过程。此外，通过该实验体会和了解优先权和时间片轮转调度算法的具体实施办法。

实验题目

- ① 设计一个按优先权调度算法实现处理机调度的程序；
- ② 设计按时间片轮转算法实现处理机调度的程序。

实验提示

- ① 对于按优先权调度算法进行调度，实验所包含的主要内容如下：

进程名 输入；

要求运行时间（单位时间）；

优先权；

状态；

就绪指针。

- ② 对于按时间片轮转算法进行调度，实验所包含的主要内容如下：

进程名 输入；

要求运行时间（单位时间）；

已运行时间；

状态；

就绪指针。

缘要求

- ① 可以随机输入若干进程，并按优先权排序。
- ② 从就绪队列的首选进程开始运行：优先权 原员 要求运行时间 原员 要求当运行时间 越园时，撤销该进程。
- ③ 重新排序，进行下一轮调度。
- ④ 采用图形界面实现，可以适当加入动态动画效果。
- ⑤ 可以随时增加进程。
- ⑥ 规定道数，设置后备队列和挂起状态。若内存中进程少于规定道数，可以自动从后备队列调度一作业进入。被挂起进程进入挂起队列，设置解挂功能用于将指定挂起进程解挂并加入就绪队列。
- ⑦ 每次调度后，显示各进程的状态。

远实验报告

- ① 程序中使用的数据结构及符号说明。
- ② 打印一份源程序并附上注释。
- ③ 针对输入的数据，给出程序执行结果。

实验二摇主存储器空间的分配和回收

员实验内容

主存储器空间的分配和回收。

圆实验目的

计算机系统不仅要有足够容量、存取速度高、稳定可靠的主存储器，而且要能合理地分配和使用这些存储空间。主存的分配和回收的实现是与主存储器的管理方式有关的。本实验有助于了解在不同的存储管理方式下，应怎样实现主存空间的分配和回收。

猿实验题目

在可变分区管理方式下，采用最先适应算法实现主存空间的分配和回收。

源提示与要求

- ① 假设主存空间大小，预设操作系统所占大小并构造未分分区表。未分分区表目内容包括起址、长度、状态（未分 轶表目）。
- ② 结合实验一， 的内容增加为：
孕圆；
要求运行时间；
优先权；
状态；
所需主存大小；
主存起始位置；
孕圆指针。
- ③ 采用最先适应算法分配主存空间。
- ④ 进程完成后，回收主存，并与相邻空闲分区合并。

缘实验报告

- ① 程序中使用的数据结构及符号说明；
- ② 打印一份源程序并附注释；
- ③ 针对输入的数据，给出程序执行结果。

实验三 中断处理

员实验内容

模拟中断事件的处理。

园实验目的

计算机系统的硬件部分都设有中断机构，它是实现多道程序设计的基础。对于不同的中断事件，操作系统应该基于其性质采用不同的处理。本实验模拟“时钟中断事件”的处理，从而了解中断及中断处理程序的作用。

猿实验题目

模拟时钟中断的产生并设计一个对时钟中断事件进行处理的模拟程序。

源提示与要求

- ① 用从键盘读入信息来模拟中断寄存器的作用，用计数器加 员来模拟处理器执行了一条指令；
- ② 假设计算机系统有一时钟，它按照电源频率产生中断请求信号，可以按照个人确定频率在键盘上键入“园”或“员”来模拟按电源频率产生的时钟中断信号；
- ③ 中断处理程序应该首先保护被中断的当前进程的现场，现场信息可以保存在进程控制块 猿中，然后处理出现的中断事件，根据处理结果修改被中断进程的状态，然后转向处理器调度。选择可运行的进程，并恢复现场使其运行；
- ④ 实验模拟程序要求分成两部分，一部分是模拟硬件产生时钟中断，另一部分是模拟操作系统的时钟中断处理程序。

缘实验报告

- ① 程序中使用的数据结构及符号说明；
- ② 打印一份源程序并附注释；
- ③ 打印开机时间、定时闹钟初值以及定时闹钟为“园”时的日历时钟。

实验四 资源分配

员实验内容

模拟资源分配。

园实验目的

多个进程动态共享系统的资源可能会产生死锁现象。但只要资源分配策略能保证进程不出现循环等待，系统就不会发生死锁。本实验通过编写和调试一个系统动态分配资源的简单模拟程序，观察死锁产生的条件，并采用适当的算法，有效地防止和避免死锁的发生。

实验题目

用银行家算法实现资源分配。

提示

(员 银行家算法的分配原则是：系统掌握每个进程对资源的最大需求量，当进程要求申请资源时，系统就测试该进程尚需资源的最大量，如果系统中现存的资源数大于或等于该进程尚需的最大量时，就满足进程的当前申请。

(圆 在进程控制块 孕悦中记录进程需要和已占用资源的情况，该 孕悦所包含内容如下：

孕悦；

当前申请量；

资源需求总量；

已占资源量；

状态。

(猿 为了观察死锁现象的发生，在模拟程序中可以采用两种资源分配算法：银行家算法和随机算法。随机算法分配资源可能产生死锁。

要求

(员 设计一个 源个并发进程共享 员圆个同类资源的系统，进程可动态申请资源和释放资源，系统根据申请动态分配资源；

(圆 设计用银行家算法和随机分配算法，实现资源分配的两种程序，显示各进程依次要求申请的资源数以及依次分配资源的情况。

实验报告

(员 程序中使用的数据结构及符号说明；

(圆 打印一份源程序并附注释；

(猿 针对输入的数据，给出资源申请和分配情况。

参考文献

- [1] 曾平. 操作系统考研指导. 北京: 清华大学出版社, 2006.
- [2] 罗宇, 邹鹏等. 操作系统. 北京: 电子工业出版社, 2009.
- [3] 张尧学, 史美林. 计算机操作系统教程. 北京: 清华大学出版社, 2006.
- [4] 陈向群, 杨芙清. 操作系统教程. 北京: 北京大学出版社, 2006.
- [5] 孙钟秀. 操作系统教程 (第三版). 北京: 高等教育出版社, 2006.
- [6] 徐甲同. 计算机操作系统教程. 西安: 西安电子科技大学出版社, 2006.
- [7] 屠祁, 屠立德. 操作系统基础 (第三版). 北京: 清华大学出版社, 2006.
- [8] 陆松年. 操作系统教程. 北京: 电子工业出版社, 2006.
- [9] 何炎祥. 操作系统原理. 上海: 上海科学技术文献出版社, 2006.
- [10] 孟静. 操作系统原理简明教程. 北京: 高等教育出版社, 2006.
- [11] 于玉. 操作系统. 北京: 电子工业出版社, 2006.
- [12] 刘腾红. 计算机操作系统. 北京: 科学出版社, 2006.
- [13] 彭民德. 计算机操作系统教程——基于网络教学环境. 北京: 清华大学出版社, 2006.
- [14] 左万历. 计算机操作系统教程 (第二版). 北京: 高等教育出版社, 2006.
- [15] 尤晋元. 操作系统教程. 西安: 西北电讯工程学院出版社, 2006.
- [16] 张尧学. 操作系统. 北京: 国防工业出版社, 2006.
- [17] 王鹏, 尤晋元. 操作系统: 设计与实现 (第 4 版). 北京: 电子工业出版社, 2006.
- [18] 汤子赢, 哲凤屏. 计算机操作系统 (修订版). 西安: 西安电子科技大学出版社, 2006.
- [19] 粤. 操作系统. 北京: 高等教育出版社, 2006.
- [20] 粤. 操作系统. 北京: 机械工业出版社, 2006.
- [21] 粤. 操作系统. 北京: 机械工业出版社, 2006.
- [22] 粤. 操作系统. 北京: 机械工业出版社, 2006.
- [23] 粤. 操作系统. 北京: 机械工业出版社, 2006.
- [24] 张尧学. 计算机操作系统教程 (第二版). 北京: 清华大学出版社, 2006.
- [25] 孟庆昌. 操作系统教程. 西安: 西安电子科技大学出版社, 2006.
- [26] 辛. 操作系统——内核与设计原理. 北京: 电子工业出版社, 2006.
- [27] 陈隶君. 深入分析 操作系统内核源代码. 北京: 人民邮电出版社, 2006.
- [28] 李林英, 贺敏. 操作系统原理——操作系统. 北京: 清华大学出版社, 2006.
- [29] 许曰滨. 操作系统习题解答与考试指导. 北京: 清华大学出版社, 2006.
- [30] 张新荣, 姚文庆. 计算机组织与结构原理. 天津: 天津大学出版社, 2006.
- [31] 傅远祯. 计算机组成原理 (修订本). 北京: 电子工业出版社, 2006.
- [32] 王爱英. 计算机组成与结构 (第三版). 北京: 清华大学出版社, 2006.
- [33] 白中英. 计算机组成原理 (第三版). 北京: 科学出版社, 2006.
- [34] 戴梅. 微型计算机技术及应用. 北京: 清华大学出版社, 2006.
- [35] 张仕斌, 谭三. 网络安全技术. 北京: 清华大学出版社, 2006.
- [36] 卿斯汉, 刘文清, 温红子. 操作系统安全. 北京: 清华大学出版社, 2006.
- [37] 孙钟秀, 费翔林. 操作系统教程. 北京: 高等教育出版社, 2006.

- [1] 姚时章. 操作系统现代观点. 北京: 机械工业出版社, 2000.
- [2] 张先红. 数字签名原理及技术. 北京: 机械工业出版社, 2000.
- [3] 杨义先, 钮心忻. 网络安全理论技术. 北京: 人民邮电出版社, 2000.
- [4] 张先红. 数字签名. 北京: 机械工业出版社, 2000.
- [5] 石志国, 江俐. 计算机网络安全教程. 北京: 清华大学出版社, 2000.
- [6] 胡希明, 张建平. 戴结构分析(核心代码的结构和算法). 杭州: 浙江大学出版社, 2000.
- [7] 张丽芬, 刘美华. 操作系统原理教程. 北京: 电子工业出版社, 2000.

图书在版编目(CIP)数据

计算机操作系统 胡志刚主编 长沙 中南大学出版社, 2007

Ⅰ.计... Ⅱ.胡... Ⅲ.操作... Ⅳ.胡...

Ⅰ.计... Ⅱ.胡... Ⅲ.操作... Ⅳ.胡...

IV 胡志刚

中国版本图书馆 CIP 数据核字 (2007) 第 000000 号

计算机操作系统

胡志刚 谭长庚 主编

□ 责任编辑 谭晓萍

□ 责任印制 文桂武

□ 出版发行 中南大学出版社

社址：长沙市麓山南路 邮编：410083

发行科电话：0731-88363636 传真：0731-88363636

□ 印装 中南大学湘雅印刷厂

□ 开本 787mm×1092mm 1/16 □ 印张 15 □ 字数 350千字

□ 版次 2007 年 1 月第 1 版 □ 2007 年 1 月第 1 次印刷

□ 书号 7-308-05000-0

□ 定价 25.00 元

图书出现印装问题, 请与出版社调换