

目 录

第 1 章 计算机系统概述.....	1
1.1 操作系统的基本概念	1
1.1.1 操作系统的概念	1
1.1.2 操作系统的特征	2
1.1.3 操作系统的功能	3
1.1.4 本节习题精选	5
1.1.5 答案与解析	6
1.2 操作系统的发展与分类.....	8
1.2.1 手工操作阶段（此阶段无操作系统）	8
1.2.2 批处理阶段（操作系统开始出现）	8
1.2.3 分时操作系统	9
1.2.4 实时操作系统	10
1.2.5 网络操作系统和分布式计算机系统	10
1.2.6 个人计算机操作系统	10
1.2.7 本节习题精选	11
1.2.8 答案与解析	12
1.3 操作系统的运行环境	15
1.3.1 操作系统的运行机制	15
1.3.2 中断和异常的概念	16
1.3.3 系统调用	17
1.3.4 本节习题精选	19
1.3.5 答案与解析	21
1.4 操作系统的体系结构	25
1.4.1 大内核和微内核	25
1.4.2 本节习题精选	26
1.4.3 答案与解析	26
1.5 本章疑难点	26
第 2 章 进程管理	28
2.1 进程与线程	29
2.1.1 进程的概念和特征	29
2.1.2 进程的状态与转换	30
2.1.3 进程控制	31
2.1.4 进程的组织	33
2.1.5 进程的通信	34

2.1.6 线程概念和多线程模型	35
2.1.7 本节小结	38
2.1.8 本节习题精选	39
2.1.9 答案与解析	44
2.2 处理机调度	51
2.2.1 调度的概念	51
2.2.2 调度的时机、切换与过程	53
2.2.3 进程调度方式	53
2.2.4 调度的基本准则	54
2.2.5 典型的调度算法	54
2.2.6 本节小结	58
2.2.7 本节习题精选	58
2.2.8 答案与解析	65
2.3 进程同步	75
2.3.1 进程同步的基本概念	75
2.3.2 实现临界区互斥的基本方法	76
2.3.3 信号量	79
2.3.4 管程	82
2.3.5 经典同步问题	84
2.3.6 本节小结	90
2.3.7 本节习题精选	91
2.3.8 答案与解析	102
2.4 死锁	121
2.4.1 死锁的概念	121
2.4.2 死锁的处理策略	123
2.4.3 死锁预防	124
2.4.4 死锁避免	124
2.4.5 死锁检测和解除	129
2.4.6 本节小结	130
2.4.7 本节习题精选	130
2.4.8 答案与解析	136
2.5 本章疑难点	145
第3章 内存管理	148
3.1 内存管理概念	148
3.1.1 内存管理的基本原理和要求	149
*3.1.2 覆盖与交换	151
3.1.3 连续分配管理方式	152
3.1.4 非连续分配管理方式	155
3.1.5 本节小结	163
3.1.6 本节习题精选	163
3.1.7 答案与解析	171

3.2 虚拟内存管理	181
3.2.1 虚拟内存的基本概念	182
3.2.2 请求分页管理方式	183
3.2.3 页面置换算法（决定应该换入哪页、换出哪页）	185
3.2.4 页面分配策略	188
3.2.5 抖动	190
3.2.6 工作集	190
3.2.7 地址翻译	190
3.2.8 本节小结	193
3.2.9 本节习题精选	193
3.2.10 答案与解析	202
3.3 本章疑难点	214
第 4 章 文件管理	215
4.1 文件系统基础	216
4.1.1 文件的概念	216
4.1.2 文件的逻辑结构	219
4.1.3 目录结构	221
4.1.4 文件共享	224
4.1.5 文件保护	226
4.1.6 本节小结	227
4.1.7 本节习题精选	227
4.1.8 答案与解析	231
4.2 文件系统实现	235
4.2.1 文件系统层次结构	235
4.2.2 目录实现	236
4.2.3 文件实现——文件分配方式	237
4.2.3 文件实现——文件存储空间管理	240
4.2.5 本节小结	242
4.2.6 本节习题精选	243
4.2.7 答案与解析	249
4.3 磁盘组织与管理	255
4.3.1 磁盘的结构	256
4.3.2 磁盘调度算法	257
4.3.3 磁盘的管理	260
4.3.4 本节小结	261
4.3.5 本节习题精选	261
4.3.6 答案与解析	264
4.4 本章疑难点	268
第 5 章 输入/输出 (I/O) 管理	271
5.1 I/O 管理概述	272

5.1.1 I/O 设备	272
5.1.2 I/O 控制方式	272
5.1.3 I/O 子系统的层次结构	275
5.1.4 本节小结	277
5.1.5 本节习题精选	277
5.1.6 答案与解析	280
5.2 I/O 核心子系统	283
5.2.1 I/O 子系统概述	283
5.2.2 I/O 调度概念	283
5.2.3 高速缓存与缓冲区	284
5.2.4 设备分配与回收	287
5.2.5 SPOOLing 技术（假脱机技术）	289
5.2.6 本节小结	290
5.2.7 本节习题精选	291
5.2.8 答案与解析	294
5.3 本章疑难点	299
参考文献	300

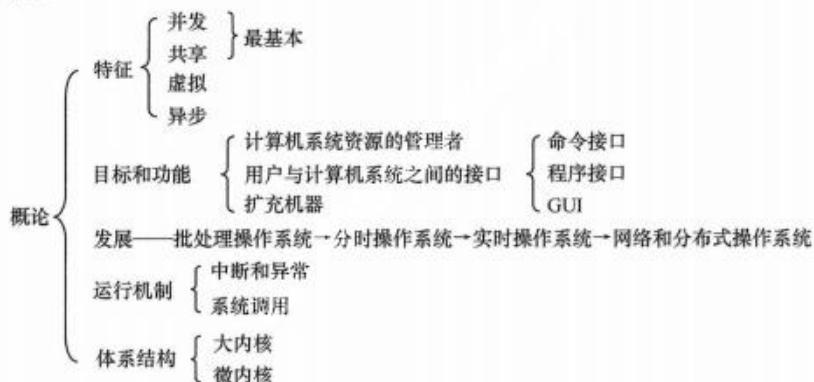
第1章

计算机系统概述

【考纲内容】

- (一) 操作系统的概念、特征、功能和提供的服务
- (二) 操作系统的发展与分类
- (三) 操作系统的运行环境
 - 内核态与用户态；中断、异常；系统调用
- (四) 操作系统体系结构

【知识框架】



【复习提示】

本章内容通常以选择题的形式考查，重点考查操作系统的功能、运行环境和提供的服务。要求读者能在宏观上把握操作系统各个部分的功能，微观上掌握细微的知识点。因此，在复习操作系统时，首先要在形成大体框架后，通过反复做题巩固、完善知识体系，然后把操作系统的所有内容串成一个整体。本章的内容有助于读者整体上初步认识操作系统，为后面展开各章节的知识点奠定基础，进而整体把握课程。不要因为本章内容在历年考题中出现的比例不高而忽视。

1.1 操作系统的基本概念

1.1.1 操作系统的概念

在信息化时代，软件是计算机系统的灵魂，而作为软件核心的操作系统，已与现代计算机系统密不可分、融为一体。计算机系统自下而上可大致分为4部分：硬件、操作系统、应用程序和用户（这里的划分与计算机组成原理中的分层不同）。操作系统管理各种计算机硬件，为应用程

序提供基础，并充当计算机硬件与用户之间的中介。

硬件如中央处理器、内存、输入/输出设备等，提供基本的计算资源。应用程序如字处理程序、电子制表软件、编译器、网络浏览器等，规定按何种方式使用这些资源来解决用户的计算问题。操作系统控制和协调各用户的应用程序对硬件的分配与使用。

在计算机系统的运行过程中，操作系统提供了正确使用这些资源的方法。

综上所述，操作系统（Operating System, OS）是指控制和管理整个计算机系统的硬件与软件资源，合理地组织、调度计算机的工作与资源的分配，进而为用户和其他软件提供方便接口与环境的程序集合。操作系统是计算机系统中最基本的系统软件。

1.1.2 操作系统的特征

操作系统是一种系统软件，但与其他系统软件和应用软件有很大的不同，它有自己的特殊性即基本特征。操作系统的基本特征包括并发、共享、虚拟和异步。这些概念对理解和掌握操作系统的核心至关重要，将一直贯穿于各个章节中。

1. 并发（Concurrence）

并发是指两个或多个事件在同一时间间隔内发生。操作系统的并发性是指计算机系统中同时存在多个运行的程序，因此它具有处理和调度多个程序同时执行的能力。在操作系统中，引入进程的目的是使程序能并发执行。

注意同一时间间隔（并发）和同一时刻（并行）的区别。在多道程序环境下，一段时间内，宏观上有多道程序在同时执行，而在每个时刻，单处理机环境下实际仅能有一道程序执行，因此微观上这些程序仍是分时交替执行的。操作系统的并发性是通过分时得以实现的。

注意，并行性是指系统具有同时进行运算或操作的特性，在同一时刻能完成两种或两种以上的工作。并行性需要有相关硬件的支持，如多流水线或多处理机硬件环境。

我们以现实生活中的直观例子来认识并发和并行的区别。例如，如果你在 9:00~9:10 仅吃面包，在 9:10~9:20 仅写字，在 9:20~9:30 仅吃面包，在 9:30~10:00 仅写字，那么在 9:00~10:00 吃面包和写字这两种行为就是并发执行的；再如，如果你在 9:00~10:00 右手写字，左手同时拿着面包吃，那么这两个动作就是并行执行的。

2. 共享（Sharing）

资源共享即共享，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。共享可分为以下两种资源共享方式。

(1) 互斥共享方式

系统中的某些资源，如打印机、磁带机，虽然可供多个进程使用，但为使得所打印或记录的结果不致造成混淆，应规定在一段时间内只允许一个进程访问该资源。

为此，当进程 A 访问某个资源时，必须先提出请求，若此时该资源空闲，则系统便将之分配给进程 A 使用，此后有其他进程也要访问该资源时（只要 A 未用完）就必须等待。仅当进程 A 访问完并释放该资源后，才允许另一个进程对该资源进行访问。我们把这种资源共享方式称为互斥式共享，而把在一段时间内只允许一个进程访问的资源称为临界资源或独占资源。计算机系统中的大多数物理设备及某些软件中所用的栈、变量和表格，都属于临界资源，它们都要求被互斥地共享。

(2) 同时访问方式

系统中还有另一类资源，这类资源允许在一段时间内由多个进程“同时”访问。这里所说的

“同时”通常是宏观上的，而在微观上，这些进程可能是交替地对该资源进行访问即“分时共享”的。可供多个进程“同时”访问的典型资源是磁盘设备，一些用重入码编写的文件也可被“同时”共享，即允许若干个用户同时访问该文件。

注意，互斥共享要求一种资源在一段时间内（哪怕是一段很小的时间）只能满足一个请求，否则就会出现严重的问题，（你能想象打印机第一行打印 A 文档的内容、第二行打印 B 文档的效果吗？）而同时访问共享通常要求一个请求分几个时间片段间隔地完成，其效果与连续完成的效果相同。

并发和共享是操作系统两个最基本的特征，两者之间互为存在的条件：① 资源共享是以程序的并发为条件的，若系统不允许程序并发执行，则自然不存在资源共享问题；② 若系统不能对资源共享实施有效的管理，则必将影响到程序的并发执行，甚至根本无法并发执行。

3. 虚拟 (Virtual)

虚拟是指把一个物理上的实体变为若干逻辑上的对应物。物理实体（前者）是实的，即实际存在的；而后者是虚的，是用户感觉上的事物。用于实现虚拟的技术，称为虚拟技术。操作系统中利用了多种虚拟技术来实现虚拟处理器、虚拟内存和虚拟外部设备等。

虚拟处理器技术是通过多道程序设计技术，采用让多道程序并发执行的方法，来分时使用一个处理器的。此时，虽然只有一个处理器，但它能同时为多个用户提供服务，使每个终端用户都感觉有一个中央处理器（CPU）在专门为它服务。利用多道程序设计技术把一个物理上的 CPU 虚拟为多个逻辑上的 CPU，称为虚拟处理器。

类似地，可以采用虚拟存储器技术将一台机器的物理存储器变为虚拟存储器，以便从逻辑上扩充存储器的容量。当然，这时用户所感觉到的内存容量是虚的。我们把用户感觉到（但实际不存在）的存储器称为虚拟存储器。

还可采用虚拟设备技术将一台物理 I/O 设备虚拟为多台逻辑上的 I/O 设备，并允许每个用户占用一台逻辑上的 I/O 设备，使原来仅允许在一段时间内由一个用户访问的设备（即临界资源）变为在一段时间内允许多个用户同时访问的共享设备。

因此，操作系统的虚拟技术可归纳为：时分复用技术，如处理器的分时共享；空分复用技术，如虚拟存储器。

4. 异步 (Asynchronism)

多道程序环境允许多个程序并发执行，但由于资源有限，进程的执行并不是一貫到底的，而是走走停停的，它以不可预知的速度向前推进，这就是进程的异步性。

异步性使得操作系统运行在一种随机的环境下，可能导致进程产生与时间有关的错误（就像对全局变量的访问顺序不当会导致程序出错一样）。然而，只要运行环境相同，操作系统就须保证多次运行进程后都能获得相同的结果。

1.1.3 操作系统的目标和功能

为了给多道程序提供良好的运行环境，操作系统应具有以下几方面的功能：处理器管理、存储器管理、设备管理和文件管理。为了方便用户使用操作系统，还必须向用户提供接口。同时，操作系统可用来扩充机器，以提供更方便的服务、更高的资源利用率。

我们用一个直观的例子来理解这种情况。例如，用户是雇主，操作系统是工人（用来操作机器），计算机是机器（由处理器、存储器、设备、文件几个部件构成），工人有熟练的技能，能够控制和协调各个部件的工作，这就是操作系统对资源的管理；同时，工人必须接收雇主的命令，

这就是“接口”；有了工人，机器就能发挥更大的作用，因此工人就成了“扩充机器”。

1. 操作系统作为计算机系统资源的管理者

(1) 处理机管理

在多道程序环境下，处理机的分配和运行都以进程（或线程）为基本单位，因而对处理机的管理可归结为对进程的管理。并发是指在计算机内同时运行多个进程，因此进程何时创建、何时撤销、如何管理、如何避免冲突、合理共享就是进程管理的最主要的任务。进程管理的主要功能包括进程控制、进程同步、进程通信、死锁处理、处理机调度等。

(2) 存储器管理

存储器管理是为了给多道程序的运行提供良好的环境，方便用户使用及提高内存的利用率，主要包括内存分配与回收、地址映射、内存保护与共享和内存扩充等功能。

(3) 文件管理

计算机中的信息都是以文件的形式存在的，操作系统中负责文件管理的部分称为文件系统。文件管理包括文件存储空间的管理、目录管理及文件读写管理和保护等。

(4) 设备管理

设备管理的主要任务是完成用户的 I/O 请求，方便用户使用各种设备，并提高设备的利用率，主要包括缓冲管理、设备分配、设备处理和虚拟设备等功能。

这些工作都由“工人”负责，“雇主”无须关注。

2. 操作系统作为用户与计算机硬件系统之间的接口

为了让用户方便、快捷、可靠地操纵计算机硬件并运行自己的程序，操作系统还提供了用户接口。操作系统提供的接口主要分为两类：一类是命令接口，用户利用这些操作命令来组织和控制作业的执行；另一类是程序接口，编程人员可以使用它们来请求操作系统服务。

(1) 命令接口

使用命令接口进行作业控制的主要方式有两种，即联机控制方式和脱机控制方式。按作业控制方式的不同，可将命令接口分为联机命令接口和脱机命令接口。

联机命令接口又称交互式命令接口，适用于分时或实时系统的接口。它由一组键盘操作命令组成。用户通过控制台或终端输入操作命令，向系统提出各种服务要求。用户每输入一条命令，控制权就转给操作系统的命令解释程序，然后由命令解释程序解释并执行输入的命令，完成指定的功能。之后，控制权转回控制台或终端，此时用户又可输入下一条命令。联机命令接口可以这样理解：“雇主”说一句话，“工人”做一件事，并做出反馈，这就强调了交互性。

脱机命令接口又称批处理命令接口，适用于批处理系统，它由一组作业控制命令组成。脱机用户不能直接干预作业的运行，而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。系统调度到该作业时，由系统中的命令解释程序逐条解释执行作业说明书上的命令，从而间接地控制作业的运行。脱机命令接口可以这样理解：“雇主”把要“工人”做的事写在清单上，“工人”按照清单命令逐条完成这些事，这就是批处理。

(2) 程序接口

程序接口由一组系统调用（也称广义指令）组成。用户通过在程序中使用这些系统调用请求操作系统为其提供服务，如使用各种外部设备、申请分配和回收内存及其他各种要求。

当前最为流行的是图形用户界面（GUI），即图形接口。GUI 最终是通过调用程序接口实现的，用户通过鼠标和键盘在图形界面上单击或使用快捷键，就能很方便地使用操作系统。严格来说，图形接口不是操作系统的一部分，但图形接口所调用的系统调用命令是操作系统的一部分。

3. 操作系统用作扩充机器

没有任何软件支持的计算机称为裸机，它仅构成计算机系统的物质基础，而实际呈现在用户面前的计算机系统是经过若干层软件改造的计算机。裸机在最里层，其外面是操作系统。操作系统所提供的资源管理功能和方便用户的各种服务功能，将裸机改造成功能更强、使用更方便的机器；因此，我们通常把覆盖了软件的机器称为扩充机器或虚拟机。

“工人”操作机器，机器就有更大的作用，于是“工人”便成了“扩充机器”。

注意，本课程所关注的内容是操作系统如何控制和协调处理机、存储器、设备和文件，而不关注接口和扩充机器，后两者读者只需要有个印象，能理解即可。

1.1.4 本节习题精选

一、单项选择题

1. 操作系统是一种（ ）。

A. 通用软件	B. 系统软件	C. 应用软件	D. 软件包
---------	---------	---------	--------
2. 操作系统是对（ ）进行管理的软件。

A. 软件	B. 硬件	C. 计算机资源	D. 应用程序
-------	-------	----------	---------
3. 下面的（ ）资源不是操作系统应该管理的。

A. CPU	B. 内存	C. 外存	D. 源程序
--------	-------	-------	--------
4. 下列选项中，（ ）不是操作系统关心的问题。

A. 管理计算机裸机	B. 设计、提供用户程序与硬件系统的界面	C. 管理计算机系统资源	D. 高级程序设计语言的编译器
------------	----------------------	--------------	-----------------
5. 操作系统的基本功能是（ ）。

A. 提供功能强大的网络管理工具	B. 提供用户界面方便用户使用
C. 提供方便的可视化编辑程序	D. 控制和管理系统内的各种资源
6. 现代操作系统中最基本的两个特征是（ ）。

A. 并发和不确定	B. 并发和共享	C. 共享和虚拟	D. 虚拟和不确定
-----------	----------	----------	-----------
7. 下列关于并发性的叙述中，正确的是（ ）。

A. 并发性是指若干事件在同一时刻发生	B. 并发性是指若干事件在不同时刻发生	C. 并发性是指若干事件在同一时间间隔内发生	D. 并发性是指若干事件在不同时间间隔内发生
---------------------	---------------------	------------------------	------------------------
8. 【2009 统考真题】单处理机系统中，可并行的是（ ）。

I. 进程与进程	II. 处理机与设备	III. 处理机与通道	IV. 设备与设备
A. I, II, III	B. I, II, IV	C. I, III, IV	D. II, III, IV
9. 用户可以通过（ ）两种方式来使用计算机。

A. 命令接口和函数	B. 命令接口和系统调用
C. 命令接口和文件管理	D. 设备管理方式和系统调用
10. 系统调用是由操作系统提供给用户的，它（ ）。

A. 直接通过键盘交互方式使用	B. 只能通过用户程序间接使用
C. 是命令接口中的命令	D. 与系统的命令一样

11. 【2010 统考真题】下列选项中，操作系统提供给应用程序的接口是（ ）。
- A. 系统调用 B. 中断 C. 库函数 D. 原语
12. 操作系统提供给编程人员的接口是（ ）。
- A. 库函数 B. 高级语言 C. 系统调用 D. 子程序
13. 系统调用的目的是（ ）。
- A. 请求系统服务 B. 中止系统服务 C. 申请系统资源 D. 释放系统资源
14. 为了方便用户直接或间接地控制自己的作业，操作系统向用户提供了命令接口，该接口又可进一步分为（ ）。
- A. 联机用户接口和脱机用户接口 B. 程序接口和图形接口
C. 联机用户接口和程序接口 D. 脱机用户接口和图形接口
15. 用户在程序中试图读某文件的第 100 个逻辑块，使用操作系统提供的（ ）接口。
- A. 系统调用 B. 键盘命令 C. 原语 D. 图形用户接口
16. 操作系统与用户通信接口通常不包括（ ）。
- A. shell B. 命令解释器 C. 广义指令 D. 缓存管理指令
17. 下列选项中，不属于多道程序设计的基本特征是（ ）。
- A. 制约性 B. 间断性 C. 顺序性 D. 共享性
18. 以下关于操作系统的叙述中，错误的是（ ）。
- A. 操作系统是管理资源的程序
B. 操作系统是管理用户程序执行的程序
C. 操作系统是能使系统资源提高效率的程序
D. 操作系统是用来编程的程序
19. 【2013 统考真题】计算机开机后，操作系统最终被加载到（ ）。
- A. BIOS B. ROM C. EPROM D. RAM

二、综合应用题

1. 说明库函数与系统调用的区别和联系。

1.1.5 答案与解析

一、单项选择题

1. B

系统软件包括操作系统、数据库管理系统、语言处理程序、服务性程序、标准库程序等。

2. C

操作系统管理计算机的硬件和软件资源，这些资源统称为计算机资源。注意，操作系统不仅管理处理器、存储器等硬件资源，而且也管理文件，文件不属于硬件资源，但属于计算机资源。

3. D

源程序是一种计算机代码，是用程序设计语言编写的程序，经编译或解释后可形成具有一定功能的可执行文件，是直接面向程序员用户的，而不是操作系统的管理内容。本题采用排除法可轻易得到答案，但有人会问操作系统不是也管理“文件”吗？源程序也存储在文件中吧？出现这种疑问的原因是，对操作系统管理文件的理解存在偏颇。操作系统管理文件，是指操作系统关心计算机中的文件的逻辑结构、物理结构、文件内部结构、多文件之间如何组织的问题，而不是关心文件的具体内容。这就好比你是操作系统，有十个水杯让你管理，你负责的是将这些水杯放在

何处比较合适，而不关心水杯中的是水还是饮料。后续章节会详细介绍文件的管理。

4. D

操作系统管理计算机软/硬件资源，扩充裸机以提供功能更强大的扩充机器，并充当用户与硬件交互的中介。高级程序设计语言的编译器显然不是操作系统关心的问题。编译器的实质是一段程序指令，它存储在计算机中，是上述水杯中的水。

5. D

操作系统是指控制和管理整个计算机系统的硬件和软件资源，合理地组织、调度计算机的工作和资源的分配，以便为用户和其他软件提供方便的接口与环境的程序集合。A、B、C 都可理解成应用程序为用户提供的服务，是应用程序的功能，而不是操作系统的功能。

6. B

操作系统最基本的特征是并发和共享，两者互为存在条件。

7. C

并发性是指若干事件在同一时间间隔内发生，而并行性是指若干事件在同一时刻发生。

8. D

在单处理器系统（不包含多核的情况）中，同一时刻只能有一个进程占用处理器，因此进程之间不能并行执行。通道是独立于 CPU 的、控制输入/输出的设备，两者可以并行。显然，处理器与设备是可以并行的，难道 CPU 和显示屏不能并行工作？设备与设备是可以并行的，难道显示屏与打印机不能并行工作？

9. B

操作系统主要向用户提供命令接口和程序接口（系统调用），此外还提供图形接口；当然，图形接口其实是调用了系统调用而实现的功能。

10. B

系统调用是操作系统为应用程序使用内核功能所提供的接口。

11. A

操作系统接口主要有命令接口和程序接口（也称系统调用）。库函数是高级语言中提供的与系统调用对应的函数（也有些库函数与系统调用无关），目的是隐藏“访管”指令的细节，使系统调用更为方便、抽象。但是，库函数属于用户程序而非系统调用，是系统调用的上层。

12. C

操作系统为编程人员提供的接口是程序接口，即系统调用。

13. A

操作系统不允许用户直接操作各种硬件资源，因此用户程序只能通过系统调用的方式来请求内核为其服务，间接地使用各种资源。

14. A

程序接口、图形接口与命令接口三者并没有从属关系。按命令控制方式的不同，命令接口分为联机用户接口和脱机用户接口。

15. A

操作系统通过系统调用向用户程序提供服务，文件 I/O 需要在内核态运行。

16. D

广义指令就是系统调用命令，而命令解释器属于命令接口，shell 是命令解析器，它也属于命令接口。系统中的缓存全部由操作系统管理，对用户是透明的，操作系统不提供管理系统缓存的系统调用。

17. C

引入多道程序设计后，程序的执行就失去了封闭性和顺序性。程序执行因为共享资源及相互协调的原因产生了竞争，相互制约。考虑到竞争的公平性，程序的执行是断续的。顺序性是单道程序设计的基本特征。

18. D

操作系统是用来管理资源的程序，用户程序也是在操作系统的管理下完成的。配置了操作系统的机器与裸机相比，资源利用率大大提高。操作系统不能直接用来编程，D 错误。

19. D

系统开机后，操作系统的程序会被自动加载到内存中的系统区，这段区域是 RAM。部分未复习计算机组成原理的读者可能对此题的答案并不熟悉，但熟悉了计算机组成原理中的各类存储介质后，相信选对这道题并不难。

二、综合应用题

1. 解答：

库函数是语言或应用程序的一部分，可以运行在用户空间中。而系统调用是操作系统的一部分，是内核为用户提供的程序接口，运行在内核空间中，而且许多库函数都会使用系统调用实现功能。未使用系统调用的库函数，其执行效率通常要比系统调用的高。因为使用系统调用时，需要上下文的切换及状态的转换（由用户态转向核心态）。

1.2 操作系统的发展与分类

1.2.1 手工操作阶段（此阶段无操作系统）

用户在计算机上算题的所有工作都要人工干预，如程序的装入、运行、结果的输出等。随着计算机硬件的发展，人机矛盾（速度和资源利用）越来越大，必须寻求新的解决办法。

手工操作阶段有两个突出的缺点：① 用户独占全机，虽然不会出现因资源已被其他用户占用而等待的现象，但资源利用率低。② CPU 等待手工操作，CPU 的利用不充分。

唯一的解决办法就是用高速的机器代替相对较慢的手工操作来对作业进行控制。

1.2.2 批处理阶段（操作系统开始出现）

为了解决人机矛盾及 CPU 和 I/O 设备之间速度不匹配的矛盾，出现了批处理系统。按发展历程又分为单道批处理系统、多道批处理系统（多道程序设计技术出现以后）。

1. 单道批处理系统

系统对作业的处理是成批进行的，但内存中始终保持一道作业。单道批处理系统是在解决人机矛盾及 CPU 和 I/O 设备速率不匹配的矛盾中形成的。单道批处理系统的主要特征如下：

- 1) 自动性。在顺利的情况下，磁带上的一批作业能自动地逐个运行，而无须人工干预。
- 2) 顺序性。磁带上的各道作业顺序地进入内存，各道作业的完成顺序与它们进入内存的顺序在正常情况下应完全相同，亦即先调入内存的作业先完成。
- 3) 单道性。内存中仅有一道程序运行，即监督程序每次从磁带上只调入一道程序进入内存运行，当该程序完成或发生异常情况时，才换入其后继程序进入内存运行。

此时面临的问题是：每次主机内存中仅存放一道作业，每当它在运行期间（注意这里是“运

行时”而不是“完成后”)发出输入/输出请求后,高速的CPU便处于等待低速的I/O完成的状态。为了进一步提高资源的利用率和系统的吞吐量,引入了多道程序技术。

2. 多道批处理系统

多道程序设计技术允许多个程序同时进入内存并允许它们在CPU中交替地运行,这些程序共享系统中的各种硬/软件资源。当一道程序因I/O请求而暂停运行时,CPU便立即转去运行另一道程序。它不采用某些机制来提高某一技术方面的瓶颈问题,而让系统的各个组成部分都尽量去“忙”,因此切换任务所花费的时间很少,可实现系统各部件之间的并行工作,使其整体在单位时间内的效率翻倍。

当然,多道批处理系统的设计和实现要比单道系统复杂很多,因为要充分利用各种资源,就要涉及各种资源的调度问题。

多道程序设计的特点是多道、宏观上并行、微观上串行。

- 1) 多道。计算机内存中同时存放多道相互独立的程序。
- 2) 宏观上并行。同时进入系统的多道程序都处于运行过程中,即它们先后开始各自的运行,但都未运行完毕。
- 3) 微观上串行。内存中的多道程序轮流占有CPU,交替执行。

多道程序设计技术的实现需要解决下列问题:

- 1) 如何分配处理器。
- 2) 多道程序的内存分配问题。
- 3) I/O设备如何分配。
- 4) 如何组织和存放大量的程序和数据,以方便用户使用并保证其安全性与一致性。

在批处理系统中采用多道程序设计技术就形成了多道批处理操作系统。该系统把用户提交的作业成批地送入计算机内存,然后由作业调度程序自动地选择作业运行。

优点:资源利用率高,多道程序共享计算机资源,从而使各种资源得到充分利用;系统吞吐量大,CPU和其他资源保持“忙碌”状态。缺点:用户响应的时间较长;不提供人机交互能力,用户既不能了解自己的程序的运行情况,又不能控制计算机。

1.2.3 分时操作系统

所谓分时技术,是指把处理器的运行时间分成很短的时间片,按时间片轮流把处理器分配给各联机作业使用。若某个作业在分配给它的时间片内不能完成其计算,则该作业暂时停止运行,把处理器让给其他作业使用,等待下一轮再继续运行。由于计算机速度很快,作业运行轮转得也很快,因此给每个用户的感觉就像是自己独占一台计算机。

分时操作系统是指多个用户通过终端同时共享一台主机,这些终端连接在主机上,用户可以同时与主机进行交互操作而互不干扰。因此,实现分时系统最关键的问题是如何使用户能与自己的作业进行交互,即当用户在自己的终端上键入命令时,系统应能及时接收并及时处理该命令,再将结果返回用户。分时系统也是支持多道程序设计的系统,但它不同于多道批处理系统。多道批处理是实现作业自动控制而无须人工干预的系统,而分时系统是实现人机交互的系统,这使得分时系统具有与批处理系统不同的特征。分时系统的主要特征如下:

- 1) 同时性。同时性也称多路性,允许多个终端用户同时使用一台计算机,即一台计算机与若干台终端相连接,终端上的这些用户可以同时或基本同时使用计算机。
- 2) 交互性。用户能够方便地与系统进行人机对话,即用户通过终端采用人机对话的方式直接控制程序运行,与同程序进行交互。

3) 独立性。系统中多个用户可以彼此独立地进行操作，互不干扰，单个用户感觉不到别人也在使用这台计算机，好像只有自己单独使用这台计算机一样。

4) 及时性。用户请求能在很短时间内获得响应。分时系统采用时间片轮转方式使一台计算机同时为多个终端服务，使用户能够对系统的及时响应感到满意。

虽然分时操作系统较好地解决了人机交互问题，但在一些应用场合，需要系统能对外部的信息在规定的时间（比时间片的时间还短）内做出处理（比如飞机订票系统或导弹制导系统），因此，实时操作系统应运而生。

1.2.4 实时操作系统

为了能在某个时间限制内完成某些紧急任务而不需要时间片排队，诞生了实时操作系统。这里的时间限制可以分为两种情况：若某个动作必须绝对地在规定的时刻（或规定的时间范围）发生，则称为硬实时系统，如飞行器的飞行自动控制系统，这类系统必须提供绝对保证，让某个特定的动作在规定的时间内完成。若能够接受偶尔违反时间规定且不会引起任何永久性的损害，则称为软实时系统，如飞机订票系统、银行管理系统。

在实时操作系统的控制下，计算机系统接收到外部信号后及时进行处理，并在严格的时限内处理完接收的事件。实时操作系统的主要特点是及时性和可靠性。

1.2.5 网络操作系统和分布式计算机系统

网络操作系统把计算机网络中的各台计算机有机地结合起来，提供一种统一、经济而有效的使用各台计算机的方法，实现各台计算机之间数据的互相传送。网络操作系统最主要的特点是网络中各种资源的共享及各台计算机之间的通信。

分布式计算机系统是由多台计算机组成并满足下列条件的系统：系统中任意两台计算机通过通信方式交换信息；系统中的每台计算机都具有同等的地位，即没有主机也没有从机；每台计算机上的资源为所有用户共享；系统中的任意台计算机都可以构成一个子系统，并且还能重构；任何工作都可以分布在几台计算机上，由它们并行工作、协同完成。用于管理分布式计算机系统的操作系统称为分布式计算机系统。该系统的主要特点是：分布性和并行性。分布式操作系统与网络操作系统的本质不同是，分布式操作系统中的若干计算机相互协同完成同一任务。

1.2.6 个人计算机操作系统

个人计算机操作系统是目前使用最广泛的操作系统，它广泛应用于文字处理、电子表格、游戏中，常见的有 Windows、Linux 和 Macintosh 等。操作系统的发展历程如图 1.1 所示。

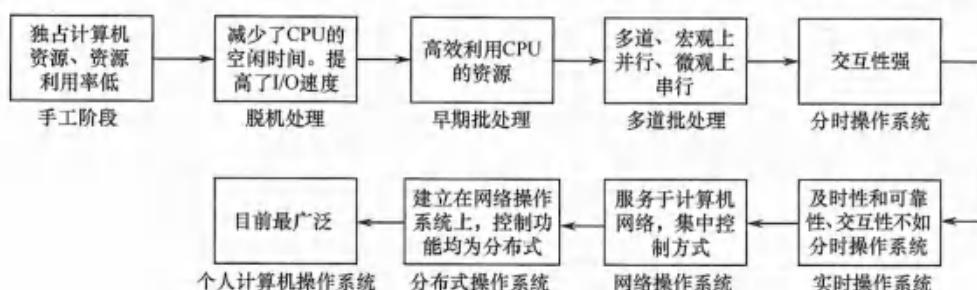


图 1.1 操作系统的发展历程

此外，还有嵌入式操作系统、服务器操作系统、智能手机操作系统等。

1.2.7 本节习题精选

一、单项选择题

1. 提高单机资源利用率的关键技术是()。

A. 脱机技术	B. 虚拟技术
C. 交换技术	D. 多道程序设计技术
2. 批处理系统的主要缺点是()。

A. 系统吞吐量小	B. CPU利用率不高	C. 资源利用率低	D. 无交互能力
-----------	-------------	-----------	----------
3. 下列选项中,不属于多道程序设计的基本特征的是()。

A. 制约性	B. 间断性	C. 顺序性	D. 共享性
--------	--------	--------	--------
4. 操作系统的基本类型主要有()。

A. 批处理操作系统、分时操作系统和多任务系统	B. 批处理操作系统、分时操作系统和实时操作系统
C. 单用户系统、多用户系统和批处理操作系统	D. 实时操作系统、分时操作系统和多用户系统
5. 【2016统考真题】下列关于批处理系统的叙述中,正确的是()。

I. 批处理系统允许多个用户与计算机直接交互	II. 批处理系统分为单道批处理系统和多道批处理系统		
III. 中断技术使得多道批处理系统和I/O设备可与CPU并行工作			
A. 仅I、III	B. 仅II	C. 仅I、II	D. 仅I、III
6. 【2017统考真题】与单道程序系统相比,多道程序系统的优点是()。

I. CPU利用率高	II. 系统开销小		
III. 系统吞吐量大	IV. I/O设备利用率高		
A. 仅I、III	B. 仅I、IV	C. 仅II、III	D. 仅I、III、IV
7. 实时操作系统必须在()内处理来自外部的事件。

A. 一个机器周期	B. 被控制对象规定时间
C. 周转时间	D. 时间片
8. 实时系统的进程调度,通常采用()算法。

A. 先来先服务	B. 时间片轮转
C. 抢占式的优先级高者优先	D. 高响应比优先
9. ()不是设计实时操作系统的主要追求目标。

A. 安全可靠	B. 资源利用率	C. 及时响应	D. 快速处理
---------	----------	---------	---------
10. 下列()应用工作最好采用实时操作系统平台。

I. 航空订票	II. 办公自动化	III. 机床控制	
IV. AutoCAD	V. 工资管理系统	VI. 股票交易系统	
A. I、II和III	B. I、III和IV	C. I、V和IV	D. I、III和VI
11. 分时系统的一个重要性能是系统的响应时间,对操作系统的()因素进行改进有利于改善系统的响应时间。

A. 加大时间片	B. 采用静态页式管理
C. 优先级+非抢占式调度算法	D. 代码可重入
12. 分时系统追求的目标是()。

- A. 充分利用 I/O 设备 B. 比较快速响应用户
 C. 提高系统吞吐率 D. 充分利用内存
13. 在分时系统中，时间片一定时，() 响应时间越长。
 A. 内存越多 B. 内存越少 C. 用户数越多 D. 用户数越少
14. 在分时系统中，为使多个进程能够及时与系统交互，最关键的问题是能在短时间内，使所有就绪进程都能运行。当就绪进程数为 100 时，为保证响应时间不超过 2s，此时的时间片最大应为 ()。
 A. 10ms B. 20ms C. 50ms D. 100ms
15. 操作系统有多种类型。允许多个用户以交互的方式使用计算机的操作系统，称为 (); 允许多个用户将若干作业提交给计算机系统集中处理的操作系统，称为 (); 在 () 的控制下，计算机系统能及时处理由过程控制反馈的数据，并及时做出响应；在 IBM-PC 中，操作系统称为 ()。
 A. 批处理系统 B. 分时操作系统
 C. 实时操作系统 D. 微型计算机操作系统
16. 【2018 统考真题】下列关于多任务操作系统的叙述中，正确的是 ()。
 I. 具有并发和并行的特点
 II. 需要实现对共享资源的保护
 III. 需要运行在多 CPU 的硬件平台上
 A. 仅 I B. 仅 II C. 仅 I、II D. I、II、III

二、综合应用题

- 批处理操作系统、分时操作系统和实时操作系统各有什么特点？
- 有两个程序，程序 A 依次使用 CPU 计 10s、设备甲计 5s、CPU 计 5s、设备乙计 10s、CPU 计 10s；程序 B 依次使用设备甲计 10s、CPU 计 10s、设备乙计 5s、CPU 计 5s、设备乙计 10s。在单道程序环境下先执行程序 A 再执行程序 B，CPU 的利用率是多少？在多道程序环境下，CPU 利用率是多少？
- 设某计算机系统有一个 CPU、一台输入设备、一台打印机。现有两个进程同时进入就绪态，且进程 A 先得到 CPU 运行，进程 B 后运行。进程 A 的运行轨迹为：计算 50ms，打印信息 100ms，再计算 50ms，打印信息 100ms，结束。进程 B 的运行轨迹为：计算 50ms，输入数据 80ms，再计算 100ms，结束。画出它们的时序关系图（可用甘特图），并说明：
 1) 开始运行后，CPU 有无空闲等待？若有，在哪段时间内等待？计算 CPU 的利用率。
 2) 进程 A 运行时有无等待现象？若有，在何时发生等待现象？
 3) 进程 B 运行时有无等待现象？若有，在何时发生等待现象？

1.2.8 答案与解析

一、单项选择题

1. D

脱机技术用于解决独占设备问题。虚拟技术与交换技术以多道程序设计技术为前提。多道程序设计技术由于同时在主存中运行多个程序，在一个程序等待时，可以去执行其他程序，因此提高了系统资源的利用率。

2. D

批处理系统中，作业执行时用户无法干预其运行，只能通过事先编制作业控制说明书来间接

干预，缺少交互能力，也因此才有了分时操作系统的出现。

3. C

多道程序的运行环境比单道程序的运行环境更加复杂。引入多道程序后，程序的执行就失去了封闭性和顺序性。程序执行因为共享资源及相互协同的原因产生了竞争，相互制约。

考虑到竞争的公平性，程序的执行是断续的。

4. B

操作系统的基本类型主要有批处理操作系统、分时操作系统和实时操作系统。

5. A

批处理系统中，作业执行时用户无法干预其运行，只能通过事先编制作业控制说明书来间接干预，缺少交互能力，也因此才发展出分时操作系统，I 错误。批处理系统按发展历程又分为单道批处理系统、多道批处理系统，II 正确。多道程序设计技术允许同时把多个程序放入内存，并允许它们交替在 CPU 中运行，它们共享系统中的各种硬/软件资源，当一道程序因 I/O 请求而暂停运行时，CPU 便立即转去运行另一道程序，即多道批处理系统的 I/O 设备可与 CPU 并行工作，这都是借助于中断技术实现的，III 正确。

6. D

多道程序系统通过组织作业（编码或数据）使 CPU 总有一个作业可执行，从而提高了 CPU 的利用率、系统吞吐量和 I/O 设备利用率，I、III、IV 是优点。但系统要付出额外的开销来组织作业和切换作业，II 错误。所以选 D。

7. B

实时系统要求能实时处理外部事件，即在规定的时间内完成对外部事件的处理。

8. C

实时系统必须能足够及时地处理某些紧急的外部事件，因此普遍用高优先级，并用“可抢占”来确保实时处理。

9. B

实时性和可靠性是实时操作系统最重要的两个目标，而安全可靠体现了可靠性，快速处理和及时响应体现了实时性。资源利用率不是实时操作系统的主要目标，即为了保证快速处理高优先级任务，允许“浪费”一些系统资源。

10. D

实时操作系统主要应用在需要对外界输入立即做出反应的场合，不能有拖延，否则会产生严重后果。本题的选项中，航空订票系统需要实时处理票务，因为票额数据库的数量直接反映了航班的可订机位。机床控制也要实时，不然会出差错。股票交易行情随时在变，若不能实时交易会出现时间差，使交易出现偏差。

11. C

采用优先级 + 非抢占式调度算法，既可让重要的作业/进程通过高优先级尽快获得系统响应，又可保证次要的作用/进程在非抢占式调度下不会迟迟得不到系统响应，这样兼顾的设计有利于改善系统的响应时间。加大时间片会延迟系统响应时间；静态页式管理和代码可重入与系统响应时间无关。

12. B

要求快速响应用户是导致分时系统出现的重要原因。

13. C

分时系统中，当时间片固定时，用户数越多，每个用户分到的时间片就越少，响应时间自然

就变长，选 C。注意，分时系统的响应时间 T 的比例关系可表达为 $T \approx QN$ ，其中 Q 是时间片，而 N 是用户数。

14. B

响应时间不超过 2s，即在 2s 内必须响应所有进程。所以时间片最大为 $2\text{s}/100 = 20\text{ms}$ 。

15. B、A、C、D

这是操作系统发展过程中的几种主要类型。

16. C

多任务操作系统可在同一时间内运行多个应用程序，因此 I 正确。多个任务必须互斥地访问共享资源，为达到这一目标必须对共享资源进行必要的保护，因此 II 正确。现代操作系统都是多任务的（主要特点是并发和并行），并不一定需要运行在多 CPU 的硬件上，单个 CPU 也可满足要求，III 错误。综上所述，I、II 正确，III 错误，因此选 C。

二、综合应用题

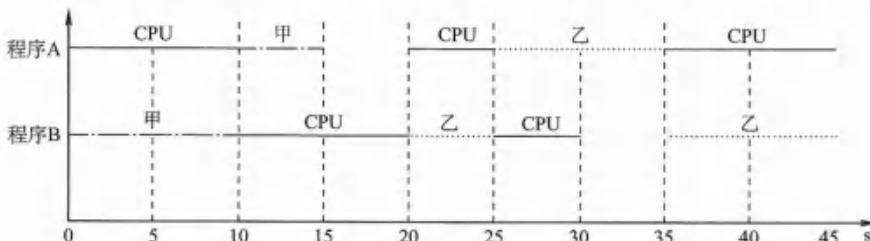
1. 解答：

- 批处理操作系统的用户脱机使用计算机，作业是成批处理的，系统内多道程序并发执行，交互能力差。
- 分时操作系统可让多个用户同时使用计算机，人机交互性较强，具有每个用户独立使用计算机的独占性，系统响应及时。
- 实时操作系统能对控制对象做出及时反应，可靠性高，响应及时，但资源利用率低。

2. 解答：

如下图所示，单道环境下，CPU 的运行时间为 $(10 + 5 + 10)\text{s} + (10 + 5)\text{s} = 40\text{s}$ ，两个程序运行的总时间为 $40\text{s} + 40\text{s} = 80\text{s}$ ，因此利用率是 $40/80 = 50\%$ 。

多道环境下，CPU 运行时间为 40s，两个程序运行总时间为 45s，因此利用率为 $40/45 = 88.9\%$ 。



注意：此图为甘特图，甘特图又称横道图，它以图示的方式通过活动列表和时间刻度形象地表示任意特定项目的活动顺序与持续时间。

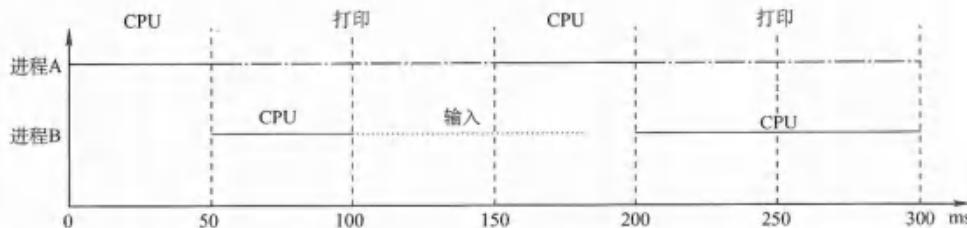
以后遇到此类题目，即给出几个不同的程序，每个程序以各个任务时间片给出时，一定要用甘特图来求解，因为其直观、快捷。为节省读者研究甘特图画法的时间，下面给出既定的步骤，读者可按下列步骤快速、正确地画出甘特图。

- ① 横坐标上标出合适的时间间隔，纵坐标上的点是程序的名字。
- ② 过横坐标上每个标出的时间点，向上作垂直于横坐标的虚线。
- ③ 用几种不同的线（推荐用“直线”“波浪线”“虚线”三种，较易区分）代表对不同资源的占用，按照题目给出的任务时间片，平行于横坐标把不同程序对应的线段分别画出来。

画图时要注意，如处理器、打印设备等资源是不能让两个程序同时使用的，有一个程序正在使用时，其他程序的请求只能排队。

3. 解答：

进程运行情况如下图所示。



- 1) CPU 在 100~150ms 时间段内空闲，利用率为 $250/300 = 83.3\%$ 。
- 2) 进程 A 为无等待现象。
- 3) 进程 B 为有等待现象，0~50ms, 180~200ms。

1.3 操作系统的运行环境

1.3.1 操作系统的运行机制^①

计算机系统中，通常 CPU 执行两种不同性质的程序：一种是操作系统内核程序；另一种是用户自编程序（即系统外层的应用程序，或简称“应用程序”）。对操作系统而言，这两种程序的作用不同，前者是后者的管理者，因此“管理程序”（即内核程序）要执行一些特权指令，而“被管理程序”（即用户自编程序）出于安全考虑不能执行这些指令。所谓特权指令，是指计算机中不允许用户直接使用的指令，如 I/O 指令、置中断指令，存取用于内存保护的寄存器、送程序状态字到程序状态字寄存器等的指令。在具体实现上，将 CPU 的状态划分为用户态（目态）和核心态（又称管态、内核态）。可以理解为 CPU 内部有一个小开关，当小开关为 1 时，CPU 处于核心态，此时 CPU 可以执行特权指令；当小开关为 0 时，CPU 处于用户态，此时 CPU 只能执行非特权指令。用户自编程序运行在用户态，操作系统内核程序运行在核心态。

在软件工程思想和结构化程序设计方法影响下诞生的现代操作系统，几乎都是层次式的结构。操作系统的各项功能分别被设置在不同的层次上。一些与硬件关联较紧密的模块，如时钟管理、中断处理、设备驱动等处于最低层。其次是运行频率较高的程序，如进程管理、存储器管理和设备管理等。这两部分内容构成了操作系统的内核。这部分内容的指令操作工作在核心态。

内核是计算机上配置的底层软件，是计算机功能的延伸。不同系统对内核的定义稍有区别，大多数操作系统的内核包括 4 方面的内容。

1. 时钟管理

在计算机的各种部件中，时钟是最关键的设备。时钟的第一功能是计时，操作系统需要通过时钟管理，向用户提供标准的系统时间。另外，通过时钟中断的管理，可以实现进程的切换。例如，在分时操作系统中采用时间片轮转调度，在实时系统中按截止时间控制运行，在批处理系统中通过时钟管理来衡量一个作业的运行程度等。因此，系统管理的方方面面无不依赖于时钟。

^① 初学者需要弄清楚一个问题，即计算机“指令”和高级语言的“代码”是不同的。我们一般所说的“编写代码”指的是用高级语言（如 C、Java 等）来编写程序。但 CPU 看不懂这些高级语言程序的含义，为了让这些程序能够顺利执行，就需要把它们“翻译”成 CPU 能懂的机器语言，即一条条“指令”（这个“翻译”的过程称为“编译”）。所谓执行程序，其实就是 CPU 根据一条条指令的指示来执行一个个具体的操作。

2. 中断机制

引入中断技术的初衷是提高多道程序运行环境中 CPU 的利用率，而且主要是针对外部设备的。后来逐步得到发展，形成了多种类型，成为操作系统各项操作的基础。例如，键盘或鼠标信息的输入、进程的管理和调度、系统功能的调用、设备驱动、文件访问等，无不依赖于中断机制。可以说，现代操作系统是靠中断驱动的软件。

中断机制中，只有一小部分功能属于内核，它们负责保护和恢复中断现场的信息，转移控制权到相关的处理程序。这样可以减少中断的处理时间，提高系统的并行处理能力。

3. 原语

按层次结构设计的操作系统，底层必然是一些可被调用的公用小程序，它们各自完成一个规定操作。它们的特点如下：

- 1) 处于操作系统的最低层，是最接近硬件的部分。
- 2) 这些程序的运行具有原子性，其操作只能一气呵成（主要从系统安全性和便于管理考虑）。
- 3) 这些程序的运行时间都较短，而且调用频繁。

通常把具有这些特点的程序称为原语（Atomic Operation）。定义原语的直接方法是关闭中断，让其所有动作不可分割地完成后再打开中断。

系统中的设备驱动、CPU 切换、进程通信等功能中的部分操作都可定义为原语，使它们成为内核的组成部分。

4. 系统控制的数据结构及处理

系统中用来登记状态信息的数据结构很多，如作业控制块、进程控制块（PCB）、设备控制块、各类链表、消息队列、缓冲区、空闲区登记表、内存分配表等。为了实现有效的管理，系统需要一些基本的操作，常见的操作有以下 3 种：

- 1) 进程管理。进程状态管理、进程调度和分派、创建与撤销进程控制块等。
- 2) 存储器管理。存储器的空间分配和回收、内存信息保护程序、代码对换程序等。
- 3) 设备管理。缓冲区管理、设备分配和回收等。

从上述内容可以了解，核心态指令实际上包括系统调用类指令和一些针对时钟、中断和原语的操作指令。

1.3.2 中断和异常的概念^①

在操作系统中引入核心态和用户态这两种工作状态后，就需要考虑这两种状态之间如何切换。操作系统内核工作在核心态，而用户程序工作在用户态。系统不允许用户程序实现核心态的功能，而它们又必须使用这些功能。因此，需要在核心态建立一些“门”，以便实现从用户态进入核心态。在实际操作系统中，CPU 运行上层程序时唯一能进入这些“门”的途径就是通过中断或异常。发生中断或异常时，运行用户态的 CPU 会立即进入核心态，这是通过硬件实现的（例如，用一个特殊寄存器的一位来表示 CPU 所处的工作状态，0 表示核心态，1 表示用户态。若要进入核心态，则只需将该位置 0 即可）。中断是操作系统中非常重要的一个概念，对一个运行在计算机上的实用操作系统而言，缺少了中断机制，将是不可想象的。原因是，操作系统的发展过程大体上就是一个想方设法不断提高资源利用率的过程，而提高资源利用率就需要在程序并未使用某种资源时，把它对那种资源的占有权释放，而这一行为就需要通过中断实现。

^① 建议结合《计算机组成原理考研复习指导》第 7 章学习，那里的讲解更详细。

1. 中断和异常的定义

中断 (Interruption) 也称外中断，指来自 CPU 执行指令以外的事件的发生，如设备发出的 I/O 结束中断，表示设备输入/输出处理已经完成，希望处理机能够向设备发下一个输入/输出请求，同时让完成输入/输出后的程序继续运行。时钟中断，表示一个固定的时间片已到，让处理机处理计时、启动定时运行的任务等。这一类中断通常是与当前指令执行无关的事件，即它们与当前处理机运行的程序无关。

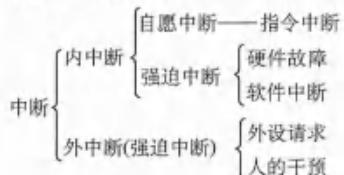


图 1.2 内中断和外中断的联系与区别

异常 (Exception) 也称内中断、例外或陷入 (trap)，指源自 CPU 执行指令内部的事件，如程序的非法操作码、地址越界、算术溢出、虚存系统的缺页及专门的陷入指令等引起的事件。对异常的处理一般要依赖于当前程序的运行现场，而且异常不能被屏蔽，一旦出现应立即处理。关于内中断和外中断的联系与区别如图 1.2 所示。

2. 中断处理的过程

不同计算机的中断（指外中断）处理过程各具特色，就其多数而论，中断处理流程如图 1.3 所示。各阶段处理流程的描述如下：

- 1) 关中断。CPU 响应中断后，首先要保护程序的现场状态，在保护现场的过程中，CPU 不应响应更高级中断源的中断请求。否则，若现场保存不完整，在中断服务程序结束后，也就不能正确地恢复并继续执行现行程序。
- 2) 保存断点。为保证中断服务程序执行完毕后能正确地返回到原来的程序，必须将原来的程序的断点（即程序计数器 PC）保存起来。
- 3) 中断服务程序寻址。其实质是取出中断服务程序的入口地址送入程序计数器 PC。
- 4) 保存现场和屏蔽字。进入中断服务程序后，首先要保存现场，现场信息一般是指程序状态字寄存器 PSWR 和某些通用寄存器的内容。
- 5) 开中断。允许更高级中断请求得到响应。
- 6) 执行中断服务程序。这是中断请求的目的。
- 7) 关中断。保证在恢复现场和屏蔽字时不被中断。
- 8) 恢复现场和屏蔽字。将现场和屏蔽字恢复到原来的状态。
- 9) 开中断、中断返回。中断服务程序的最后一条指令通常是一条中断返回指令，使其返回到原程序的断点处，以便继续执行原程序。

其中，1~3 步是在 CPU 进入中断周期后，由硬件自动（中断隐指令）完成的；4~9 步由中断服务程序完成。恢复现场是指在中断返回前，必须将寄存器的内容恢复到中断处理前的状态，这部分工作由中断服务程序完成。中断返回由中断服务程序的最后一条中断返回指令完成。

1.3.3 系统调用

所谓系统调用，是指用户在程序中调用操作系统所提供的一些子功能，系统调用可视为特殊



的公共子程序。系统中的各种共享资源都由操作系统统一掌管，因此在用户程序中，凡是与资源有关的操作（如存储分配、进行 I/O 传输及管理文件等），都必须通过系统调用方式向操作系统提出服务请求，并由操作系统代为完成。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。这些系统调用按功能大致可分为如下几类。

- 设备管理。完成设备的请求或释放，以及设备启动等功能。
- 文件管理。完成文件的读、写、创建及删除等功能。
- 进程控制。完成进程的创建、撤销、阻塞及唤醒等功能。
- 进程通信。完成进程之间的消息传递或信号传递等功能。
- 内存管理。完成内存的分配、回收以及获取作业占用内存区大小及始址等功能。

显然，系统调用相关功能涉及系统资源管理、进程管理之类的操作，对整个系统的影响非常大，因此必定需要使用某些特权指令才能完成，所以系统调用的处理需要由操作系统内核程序负责完成，要运行在核心态。用户程序可以执行陷入指令（又称访管指令或 trap 指令）来发起系统调用，请求操作系统提供服务。可以这么理解，用户程序执行“陷入指令”，相当于把 CPU 的使用权主动交给操作系统内核程序（CPU 状态会从用户态进入核心态），之后操作系统内核程序再对系统调用请求做出相应处理。处理完成后，操作系统内核程序又会把 CPU 的使用权还给用户程序（即 CPU 状态会从核心态回到用户态）。这么设计的目的是：用户程序不能直接执行对系统影响非常大的操作，必须通过系统调用的方式请求操作系统代为执行，以便保证系统的稳定性和安全性，防止用户程序随意更改或访问重要的系统资源，影响其他进程的运行。

这样，操作系统的运行环境就可以理解为：用户通过操作系统运行上层程序（如系统提供的命令解释程序或用户自编程序），而这个上层程序的运行依赖于操作系统的底层管理程序提供服务支持，当需要管理程序服务时，系统则通过硬件中断机制进入核心态，运行管理程序；也可能是程序运行出现异常情况，被动地需要管理程序的服务，这时就通过异常处理来进入核心态。管理程序运行结束时，用户程序需要继续运行，此时通过相应的保存的程序现场退出中断处理程序或异常处理程序，返回断点处继续执行，如图 1.4 所示。

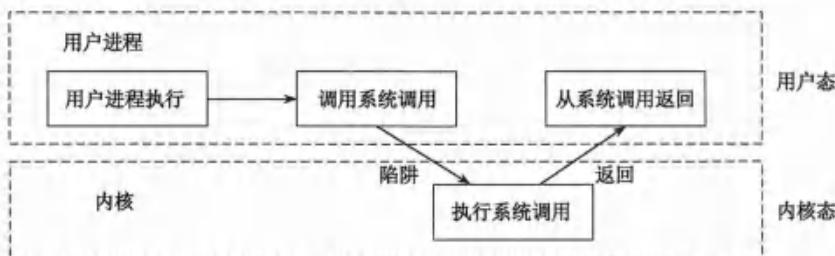


图 1.4 系统调用执行过程

在操作系统这一层面上，我们关心的是系统核心态和用户态的软件实现与切换，对于硬件层面的具体理解，可以结合“计算机组成原理”课程中有关中断的内容进行学习。

下面列举一些由用户态转向核心态的例子：

- 1) 用户程序要求操作系统的服务，即系统调用。
- 2) 发生一次中断。
- 3) 用户程序中产生了一个错误状态。
- 4) 用户程序中企图执行一条特权指令。
- 5) 从核心态转向用户态由一条指令实现，这条指令也是特权命令，一般是中断返回指令。

注意：由用户态进入核心态，不仅状态需要切换，而且所用的堆栈也可能需要由用户堆栈切换为系统堆栈，但这个系统堆栈也是属于该进程的。

若程序的运行由用户态转到核心态，则会用到访管指令，访管指令是在用户态使用的，所以它不可能是特权指令。

1.3.4 本节习题精选

一、单项选择题

1. 下列关于操作系统的说法中，错误的是（ ）。
 - I. 在通用操作系统管理下的计算机上运行程序，需要向操作系统预订运行时间
 - II. 在通用操作系统管理下的计算机上运行程序，需要确定起始地址，并从这个地址开始执行
 - III. 操作系统需要提供高级程序设计语言的编译器
 - IV. 管理计算机系统资源是操作系统关心的主要问题

A. I、III B. II、III
C. I、II、III、IV D. 以上答案都正确
2. 下列说法中，正确的是（ ）。
 - I. 批处理的主要缺点是需要大量内存
 - II. 当计算机提供了核心态和用户态时，输入/输出指令必须在核心态下执行
 - III. 操作系统中采用多道程序设计技术的最主要原因是提高CPU和外部设备的可靠性
 - IV. 操作系统中，通道技术是一种硬件技术

A. I、II B. I、III C. II、IV D. II、III、IV
3. 下列关于系统调用的说法中，正确的是（ ）。
 - I. 用户程序设计时，使用系统调用命令，该命令经过编译后，形成若干参数和陷入（trap）指令
 - II. 用户程序设计时，使用系统调用命令，该命令经过编译后，形成若干参数和屏蔽中断指令
 - III. 系统调用功能是操作系统向用户程序提供的接口
 - IV. 用户及其应用程序和应用系统是通过系统调用提供的支持和服务来使用系统资源完成其操作的

A. I、III B. II、IV C. I、III、IV D. II、III、IV
4. （ ）是操作系统必须提供的功能。
 - A. 图形用户界面（GUI）
 - B. 为进程提供系统调用命令
 - C. 中断处理
 - D. 编译源程序
5. 用户程序在用户态下要使用特权指令引起的中断属于（ ）。
 - A. 硬件故障中断
 - B. 程序中断
 - C. 外部中断
 - D. 访管中断
6. 处理器执行的指令被分为两类，其中有一类称为特权指令，它只允许（ ）使用。
 - A. 操作员
 - B. 联机用户
 - C. 目标程序
 - D. 操作系统
7. 下列操作系统的各个功能组成部分中，（ ）可不需要硬件的支持。
 - A. 进程调度
 - B. 时钟管理
 - C. 地址映射
 - D. 中断系统
8. 在中断发生后，进入中断处理的程序属于（ ）。
 - A. 用户程序
 - B. 可能是应用程序，也可能是操作系统程序

- C. 操作系统程序
D. 既不是应用程序，又不是操作系统程序
9. 计算机区分核心态和用户态指令后，从核心态到用户态的转换是由操作系统程序执行后完成的，而用户态到核心态的转换则是由（ ）完成的。
A. 硬件 B. 核心态程序 C. 用户程序 D. 中断处理程序
10. 【2011 统考真题】下列选项中，在用户态执行的是（ ）。
A. 命令解释程序 B. 缺页处理程序
C. 进程调度程序 D. 时钟中断处理程序
11. 【2012 统考真题】下列选项中，不可能在用户态发生的事件是（ ）。
A. 系统调用 B. 外部中断 C. 进程切换 D. 缺页
12. 只能在核心态下运行的指令是（ ）。
A. 读时钟指令 B. 置时钟指令 C. 取数指令 D. 寄存器清零
13. “访管”指令（ ）使用。
A. 仅在用户态下 B. 仅在核心态下 C. 在规定时间内 D. 在调度时间内
14. 当 CPU 执行操作系统代码时，处理器处于（ ）。
A. 自由态 B. 用户态 C. 核心态 D. 就绪态
15. 在操作系统中，只能在核心态下执行的指令是（ ）。
A. 读时钟 B. 取数 C. 广义指令 D. 寄存器清“0”
16. 下列选项中，必须在核心态下执行的指令是（ ）。
A. 从内存中取数 B. 将运算结果装入内存
C. 算术运算 D. 输入/输出
17. CPU 处于核心态时，它可以执行的指令是（ ）。
A. 只有特权指令 B. 只有非特权指令
C. 只有“访管”指令 D. 除“访管”指令的全部指令
18. 【2013 统考真题】下列选项中，会导致用户进程从用户态切换到内核态的操作是（ ）。
I. 整数除以零 II. sin() 函数调用 III. read 系统调用
A. 仅 I、II B. 仅 I、III C. 仅 II、III D. I、II 和 III
19. （ ）程序可执行特权指令。
A. 同组用户 B. 操作系统 C. 特权用户 D. 一般用户
20. 【2012 统考真题】中断处理和子程序调用都需要压栈以保护现场，中断处理一定会保存而子程序调用不需要保存其内容的是（ ）。
A. 程序计数器 B. 程序状态字寄存器
C. 通用数据寄存器 D. 通用地址寄存器
21. 【2014 统考真题】下列指令中，不能在用户态执行的是（ ）。
A. trap 指令 B. 跳转指令 C. 压栈指令 D. 关中断指令
22. 【2015 统考真题】内部异常（内中断）可分为故障（fault）、陷阱（trap）和终止（abort）三类。下列有关内部异常的叙述中，错误的是（ ）。
A. 内部异常的产生与当前执行指令相关
B. 内部异常的检测由 CPU 内部逻辑实现
C. 内部异常的响应发生在指令执行过程中
D. 内部异常处理后返回到发生异常的指令继续执行
23. 【2016 统考真题】异常是指令执行过程中在处理器内部发生的特殊事件，中断是来自处

- 理器外部的请求事件。下列关于中断或异常情况的叙述中，错误的是（ ）。
- “访存时缺页”属于中断
 - “整数除以 0”属于异常
 - “DMA 传送结束”属于中断
 - “存储保护错”属于异常
24. 【2015 统考真题】处理外部中断时，应该由操作系统保存的是（ ）。
- 程序计数器（PC）的内容
 - 通用寄存器的内容
 - 块表（TLB）中的内容
 - Cache 中的内容
25. 【2015 统考真题】假定下列指令已装入指令寄存器，则执行时不可能导致 CPU 从用户态变为内核态（系统态）的是（ ）。
- DIV R0, R1 ; (R0)/(R1)→R0
 - INT n ; 产生软中断
 - NOT R0 ; 寄存器 R0 的内容取非
 - MOV R0, addr ; 把地址 addr 处的内存数据放入寄存器 R0
26. 【2017 统考真题】执行系统调用的过程包括如下主要操作：
- ① 返回用户态
 - ② 执行陷入（trap）指令
 - ③ 传递系统调用参数
 - ④ 执行相应的服务程序
- 正确的执行顺序是（ ）。
- ②→③→①→④
 - ②→④→③→①
 - ③→②→④→①
 - ③→④→②→①
27. 【2018 统考真题】定时器产生时钟中断后，由时钟中断服务程序更新的部分内容是（ ）。
- I. 内核中时钟变量的值
 - II. 当前进程占用 CPU 的时间
 - III. 当前进程在时间片内的剩余执行时间
- 仅 I、II
 - 仅 II、III
 - 仅 I、III
 - I、II、III
28. 【2019 统考真题】下列关于系统调用的叙述中，正确的是（ ）。
- I. 在执行系统调用服务程序的过程中，CPU 处于内核态
 - II. 操作系统通过提供系统调用避免用户程序直接访问外设
 - III. 不同的操作系统为应用程序提供了统一的系统调用接口
 - IV. 系统调用是操作系统内核为应用程序提供服务的接口
- 仅 I、IV
 - 仅 II、III
 - 仅 I、II、IV
 - 仅 I、III、IV
29. 【2020 统考真题】下列与中断相关的操作中，由操作系统完成的是（ ）。
- I. 保存被中断程序的中断点
 - II. 提供中断服务
 - III. 初始化中断向量表
 - IV. 保存中断屏蔽字
- 仅 I、II
 - 仅 I、II、IV
 - 仅 III、IV
 - 仅 II、III、IV

二、综合应用题

- 处理器为什么要区分核心态和用户态两种操作方式？在什么情况下进行两种方式的切换？
- 为什么说直到出现中断和通道技术后，多道程序概念才变得有用？

1.3.5 答案与解析

一、单项选择题

1. A

I: 通用操作系统使用时间片轮转调度算法，用户运行程序并不需要预先预订运行时间，因此

I 项错误；II：操作系统执行程序时，必须从起始地址开始执行，因此 II 项正确；III：编译器是操作系统的上层软件，不是操作系统需要提供的功能，因此 III 项错误；IV：操作系统是计算机资源的管理者，管理计算机系统资源是操作系统关心的主要问题，因此 IV 项正确。综合分析，I 和 III 是错误项，因此选 A。

2. C

I 错误：批处理的主要缺点是缺少交互性。批处理系统的主要缺点是常考点，读者对此要非常敏感。II 正确：输入/输出指令需要中断操作，中断必须在核心态下执行。III 错误：多道性是为了提高系统利用率和吞吐量而提出的。IV 正确：I/O 通道实际上是一种特殊的处理器，它具有执行 I/O 指令的能力，并通过执行通道程序来控制 I/O 操作。综上分析，II、IV 正确。

3. C

I 正确：系统调用需要触发 trap 指令，如基于 x86 的 Linux 系统，该指令为 int 0x80 或 sysenter。II 是干扰项，程序设计无法形成屏蔽中断指令。III 正确：系统调用的概念。IV 正确：操作系统是一层接口，对上层提供服务，对下层进行抽象。它通过系统调用向其上层的用户、应用程序和应用系统提供对系统资源的使用。

4. C

中断是操作系统必须提供的功能，因为计算机的各种错误都需要中断处理，核心态与用户态切换也需要中断处理。

5. D

因操作系统不允许用户直接执行某些“危险性高”的指令，因此用户态运行这些指令的结果会转成操作系统的核心态去运行。这个过程就是访管中断。

6. D

内核可以执行处理器能执行的任何指令，用户程序只能执行除特权指令外的指令。所以特权指令只能由内核即操作系统使用。

7. A

中断系统和地址映射显然都需要硬件支持，因为中断指令和地址映射中的重定位都是离不开硬件支持的。而时钟管理中，重置时钟等是由硬件直接完成的。进程调度由调度算法决定 CPU 使用权，由操作系统实现，无须硬件的支持。

8. C

进入中断处理的程序在核心态执行，是操作系统程序。

9. A

计算机通过硬件中断机制完成由用户态到核心态的转换。B 显然不正确，核心态程序只有在操作系统进入核心态后才可以执行。D 中的中断处理程序一般也在核心态执行，因此无法完成“转换成核心态”这一任务。若由用户程序将操作系统由用户态转换到核心态，则用户程序中就可使用核心态指令，这就会威胁到计算机的安全，所以 C 不正确。

计算机通过硬件完成操作系统由用户态到核心态的转换，这是通过中断机制来实现的。发生中断事件时（有可能是用户程序发出的系统调用），触发中断，硬件中断机制将计算机状态置为核心态。

10. A

缺页处理和时钟中断都属于中断，在核心态执行；进程调度是操作系统内核进程，无须用户干预，在核心态执行；命令解释程序属于命令接口，是 4 个选项中唯一能面对用户的，它在用户态执行。

11. C

本题的关键是对“在用户态发生”（与上题的“执行”区分）的理解。对于 A，系统调用是操作系统提供给用户程序的接口，系统调用发生在用户态，被调用程序在核心态下执行。对于 B，外部中断是用户态到核心态的“门”，也发生在用户态，在核心态完成中断过程。对于 C，进程切换属于系统调用执行过程中的事件，只能发生在核心态；对于 D，缺页产生后，在用户态发生缺页中断，然后进入核心态执行缺页中断服务程序。

12. B

若在用户态下执行“置时钟指令”，则一个用户进程可在时间片还未到之前把时钟改回去，从而导致时间片永远不会用完，进而导致该用户进程一直占用 CPU，这显然不合理。

13. A

“访管”指令仅在用户态下使用，执行“访管”指令将用户态转变为核心态。

14. C

运行操作系统代码的状态为核心态。

15. C

广义指令即系统调用命令，它必然工作在核心态，所以答案为 C。要注意区分“调用”和“执行”，广义指令的调用可能发生在用户态，调用广义指令的那条指令不一定是特权指令，但广义指令存在于核心态中，所以执行一定在核心态。

16. D

输入/输出指令涉及中断操作，而中断处理是由系统内核负责的，工作在核心态。而 A、B、C 选项均可通过使用汇编语言编程来实现，因此它们可在用户态下执行。当然，读者也可用前面提到的“水杯”例子思考如何排除 A、B、C。操作系统管理内存时，管理的是内存中的数据放在哪里、哪里可以放数据、哪里不可以放数据（内存保护）、哪里空闲等问题，而内存中的数据是什么、怎么读和写，都不是核心态关心的。就好像操作系统管理的是杯子摆在哪里、哪些杯子中的水可以喝、哪些杯子中的水不能喝，而杯子中是水还是饮料、你是拿起杯子喝还是把吸管插进去吸，都不是操作系统关心的问题。“杯子”的例子可以帮助我们准确理解操作系统的任务，后续章节中的很多问题采用这个例子进行对比，就会十分清晰。

17. D

访管指令在用户态下使用，是用户程序“自愿进管”的手段，用户态下不能执行特权指令。在核心态下，CPU 可以执行指令系统中的任何指令。

18. B

需要在系统内核态执行的操作是整数除零操作（需要中断处理）和 read 系统调用函数，sin() 函数调用是在用户态下进行的。

19. B

特权指令是指仅能由操作系统使用的指令，选 B。

20. B

子程序调用只需保存程序断点，即该指令的下一条指令的地址；中断调用子程序不仅要保存断点（PC 的内容），还要保存程序状态字寄存器（PSW）的内容。在中断处理中，最重要的两个寄存器是 PC 和 PSWR。

21. D

trap 指令、跳转指令和压栈指令均可以在用户态执行，其中 trap 指令负责由用户态转换为内核态。关中断指令为特权指令，必须在核心态才能执行，选 D。注意，在操作系统中，关中断指

令是权限非常大的指令，因为中断是现代操作系统正常运行的核心保障之一，能把它关掉，说明执行这条指令的一定是权限非常大的机构（管态）。

22. D

内部异常是指来自 CPU 内部产生的中断，如电源掉电、地址非法、校验错、页面失效、非法指令、地址越界和除数为零等，以上都是在指令的执行过程中产生的，因此 A 正确。内部异常的检测是由 CPU 自身完成的，不必通过外部的某个信号通知 CPU，B 正确。内部异常不能被屏蔽，一旦出现应立即处理，C 正确。对于 D，对于非法指令、除数为零等异常，无法通过异常处理程序恢复故障，因此不能回到原断点执行，必须终止进程的执行，因此错误。

23. A

中断是指来自 CPU 执行指令以外事件的发生，如设备发出的 I/O 结束中断，表示设备输入/输出处理已经完成，希望处理机能够向设备发出下一个输入/输出请求，同时让完成输入/输出后的程序继续运行。时钟中断，表示一个固定的时间片已到，让处理机处理计时、启动定时运行的任务等。这一类中断通常是与当前程序运行无关的事件，即它们与当前处理机运行的程序无关。异常也称内中断、例外或陷入（trap），指源自 CPU 执行指令内部的事件，如程序的非法操作码、地址越界、算术溢出、虚存系统的缺页及专门的陷入指令等引起的事件。A 错误。

24. B

外部中断处理过程，PC 值由中断隐指令自动保存，而通用寄存器内容由操作系统保存。

25. C

考虑到部分指令可能出现异常（导致中断），从而转到核心态。指令 A 有除零异常的可能，指令 B 为中断指令，指令 D 有缺页异常的可能，指令 C 不会发生异常。

26. C

执行系统调用的过程如下：正在运行的进程先传递系统调用参数，然后由陷入（trap）指令负责将用户态转换为内核态，并将返回地址压入堆栈以备后用，接下来 CPU 执行相应的内核态服务程序，最后返回用户态。所以选项 C 正确。

27. D

时钟中断的主要工作是处理和时间有关的信息及决定是否执行调度程序。和时间有关的所有信息包括系统时间、进程的时间片、延时、使用 CPU 的时间、各种定时器，因此 I、II、III 均正确。

28. C

用户可以在用户态调用操作系统的服务，但执行具体的系统调用服务程序是处于内核态的，I 正确；设备管理属于操作系统的职能之一，包括对输入/输出设备的分配、初始化、维护等，用户程序需要通过系统调用使用操作系统的设备管理服务，II 正确；操作系统不同，底层逻辑、实现方式均不相同，为应用程序提供的系统调用接口也不同，III 错误；系统调用是用户在程序中调用操作系统提供的子功能，IV 正确。

29. D

当 CPU 检测到中断信号后，由硬件自动保存被中断程序的断点（即程序计数器 PC），I 错误。之后，硬件找到该中断信号对应的中断向量，中断向量指明中断服务程序入口地址（各中断向量统一存放在中断向量表中，该表由操作系统初始化，III 正确）。接下来开始执行中断服务程序，保存 PSW、保存中断屏蔽字、保存各通用寄存器的值，并提供与中断信号对应的中断服务，中断服务程序属于操作系统内核，II 和 IV 正确。

二、综合应用题

1. 解答：

区分执行态的主要目的是保护系统程序。用户态到核心态的转换发生在中断产生时，而核心态到用户态的转换则发生在中断返回用户程序时。

2. 解答：

多道程序并发执行是指有的程序正在 CPU 上执行，而另一些程序正在 I/O 设备上进行传输，即通过 CPU 操作与外设传输在时间上的重叠必须有中断和通道技术的支持，原因如下：

- 1) 通道是一种控制一台或多台外部设备的硬件机构，它一旦被启动就独立于 CPU 运行，因而做到了输入/输出操作与 CPU 并行工作。但早期 CPU 与通道的联络方法是由 CPU 向通道发出询问指令来了解通道工作是否完成的。若未完成，则主机就循环询问直到通道工作结束为止。因此，这种询问方式是无法真正做到 CPU 与 I/O 设备并行工作的。
- 2) 在硬件上引入了中断技术。所谓中断，就是在输入/输出结束时，或硬件发生某种故障时，由相应的硬件（即中断机构）向 CPU 发出信号，这时 CPU 立即停下工作而转向处理中断请求，待处理完中断后再继续原来的工作。

因此，通道技术和中断技术结合起来就可实现 CPU 与 I/O 设备并行工作，即 CPU 启动通道传输数据后便去执行其他程序的计算工作，而通道则进行输入/输出操作；当通道工作结束时，再通过中断机构向 CPU 发出中断请求，CPU 则暂停正在执行的操作，对出现的中断进行处理，处理完后再继续原来的工作。这样，就真正做到了 CPU 与 I/O 设备并行工作。此时，多道程序的概念才变为现实。

1.4 操作系统的体系结构

1.4.1 大内核和微内核

操作系统的体系结构是一个开放的问题。如上文所述，操作系统在核心态为应用程序提供公共的服务，那么操作系统在核心态应该提供什么服务、怎样提供服务？有关这一问题的回答形成了两种主要的体系结构：大内核和微内核。

大内核系统将操作系统的主要功能模块都作为一个紧密联系的整体运行在核心态，从而为应用提供高性能的系统服务。因为各管理模块之间共享信息，能有效利用相互之间的有效特性，所以具有无可比拟的性能优势。

但随着体系结构和应用需求的不断发展，需要操作系统提供的服务越来越多，而且接口形式越来越复杂，操作系统的设计规模急剧增长，操作系统也面临着“软件危机”困境。为此，操作系统设计人员试图按照复杂性、时间常数、抽象级别等因素，将操作系统内核分成基本进程管理、虚存、I/O 与设备管理、IPC、文件系统等几个层次，继而定义层次之间的服务结构，提高操作系统内核设计上的模块化。但是，由于层次之间的交互关系错综复杂，定义清晰的层次间接口非常困难，复杂的交互关系也使得层次之间的界限极其模糊。

为解决操作系统的内核代码难以维护的问题，提出了微内核的体系结构。它将内核中最基本的功能（如进程管理等）保留在内核，而将那些不需要在核心态执行的功能移到用户态执行，从而降低了内核的设计复杂性。那些移出内核的操作系统代码根据分层的原则被划分成若干服务程序，它们的执行相互独立，交互则都借助于微内核进行通信。

微内核结构有效地分离了内核与服务、服务与服务，使得它们之间的接口更加清晰，维护的代价大大降低，各部分可以独立地优化和演进，从而保证了操作系统的可靠性。

微内核结构的最大问题是性能问题，因为需要频繁地在核心态和用户态之间进行切换，操作系统的执行开销偏大。因此有的操作系统将那些频繁使用的系统服务又移回内核，从而保证系统性能。但相当多的实验数据表明，体系结构不是引起性能下降的主要因素，体系结构带来的性能提升足以弥补切换开销带来的缺陷。为减少切换开销，也有人提出将系统服务作为运行库链接到用户程序的一种解决方案，这样的体系结构称为库操作系统。

1.4.2 本节习题精选

单项选择题

1. 相对于传统操作系统结构，采用微内核结构设计和实现操作系统具有诸多好处，下列（ ）是微内核结构的特点。

- I. 使系统更高效 II. 添加系统服务时，不必修改内核
 - III. 微内核结构没有单一内核稳定 IV. 使系统更可靠
- A. I、III、IV B. I、II、IV C. II、IV D. I、IV

1.4.3 答案与解析

单项选择题

1. 答案：C

微内核结构将操作系统的很多服务移动到内核以外（如文件系统），且服务之间使用进程间通信机制进行信息交换，这种通过进程间通信机制进行的信息交换影响了系统的效率，所以 I 错。由于内核的服务变少，且一般来说内核的服务越少内核越稳定，所以 III 错。而 II、IV 正是微内核结构的优点。

1.5 本章疑难点

1. 并行性与并发性的区别和联系

并行性和并发性是既相似又有区别的两个概念。并行性是指两个或多个事件在同一时刻发生，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，并发性是指在一段时间内，宏观上有多个程序同时运行，但在单处理器系统中每个时刻却仅能有一道程序执行，因此微观上这些程序只能分时地交替执行。若在计算机系统中有多个处理器，则这些可以并发执行的程序便被分配到多个处理器上，实现并行执行，即利用每个处理器来处理一个可并发执行的程序。

2. 特权指令与非特权指令

所谓特权指令，是指有特殊权限的指令，由于这类指令的权限最大，使用不当将导致整个系统崩溃，如清内存、置时钟、分配系统资源、修改虚存的段表或页表、修改用户的访问权限等。若所有程序都能使用这些指令，则系统一天死机 n 次就不足为奇。为保证系统安全，这类指令只能用于操作系统或其他系统软件，不直接提供给用户使用。因此，特权指令必须在核心态执行。实际上，CPU 在核心态下可以执行指令系统的全集。形象地说，特权指令是那些儿童不宜的东西，

而非特权指令是老少皆宜的东西。

为了防止用户程序中使用特权指令，用户态下只能使用非特权指令，核心态下可以使用全部指令。在用户态下使用特权指令时，将产生中断以阻止用户使用特权指令。所以把用户程序放在用户态下运行，而操作系统中必须使用特权指令的那部分程序在核心态下运行，保证了计算机系统的安全可靠。从用户态转换为核心态的唯一途径是中断或异常。

3. 访管指令与访管中断

访管指令是一条可以在用户态下执行的指令。在用户程序中，因要求操作系统提供服务而有意识地使用访管指令，从而产生一个中断事件（自愿中断），将操作系统转换为核心态，称为访管中断。访管中断由访管指令产生，程序员使用访管指令向操作系统请求服务。

为什么要在程序中引入访管指令呢？这是因为用户程序只能在用户态下运行。若用户程序想要完成在用户态下无法完成的工作，该怎么办？解决这个问题要靠访管指令。访管指令本身不是特权指令，其基本功能是让程序拥有“自愿进管”的手段，从而引起访管中断。

处于用户态的用户程序使用访管指令时，系统根据访管指令的操作数执行访管中断处理程序，访管中断处理程序将按系统调用的操作数和参数转到相应的例行子程序。完成服务功能后，退出中断，返回到用户程序断点继续执行。

第2章

进程管理

【考纲内容】

(一) 进程与线程

进程的概念；进程的状态与转换

进程控制；进程组织

进程通信；线程概念与多线程模型

(二) 处理机调度

调度的基本概念；调度时机、切换与过程

调度的基本准则；调度方式；典型调度算法

(三) 进程同步

进程同步的基本概念

实现临界区互斥的基本方法

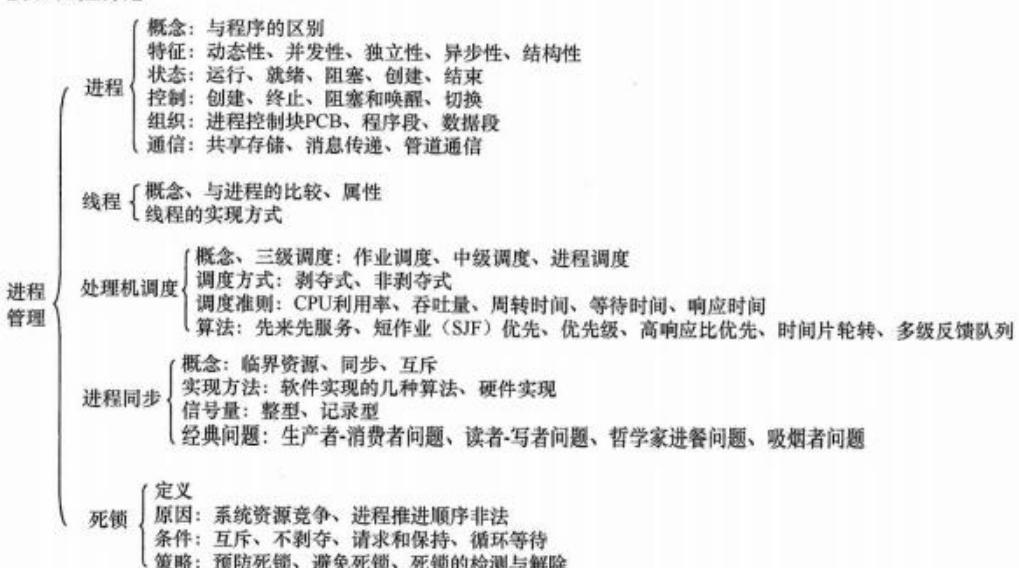
信号量；管程；经典同步问题

(四) 死锁

死锁的概念；死锁处理策略

死锁预防；死锁避免；死锁的检测和解除

【知识框架】



【复习提示】

进程管理是操作系统的核心，也是每年必考的重点。其中，进程的概念、进程调度、信号量

机制实现同步和互斥、进程死锁等更是重中之重，必须深入掌握。需要注意的是，除选择题外，本章还容易出综合题，其中信号量机制实现同步和互斥、进程调度算法和银行家算法都是可能出现的综合题考点，如利用信号量进行进程同步就在往年的统考中频繁出现。

2.1 进程与线程

在学习本节时，请读者思考以下问题：

- 1) 为什么要引入进程？
- 2) 什么是进程？进程由什么组成？
- 3) 进程是如何解决问题的？

希望读者带着上述问题去学习本节内容，并在学习的过程中多思考，从而更深入地理解本节内容。进程本身是一个比较抽象的概念，它不是实物，看不见、摸不着，初学者在理解进程概念时存在一定困难，在介绍完进程的相关知识后，我们会用比较直观的例子帮助大家理解。

2.1.1 进程的概念和特征

1. 进程的概念

在多道程序环境下，允许多个程序并发执行，此时它们将失去封闭性，并具有间断性及不可再现性的特征。为此引入了进程（Process）的概念，以便更好地描述和控制程序的并发执行，实现操作系统的并发性和共享性（最基本的两个特性）。

为了使参与并发执行的程序（含数据）能独立地运行，必须为之配置一个专门的数据结构，称为进程控制块（Process Control Block, PCB）。系统利用 PCB 来描述进程的基本情况和运行状态，进而控制和管理进程。相应地，由程序段、相关数据段和 PCB 三部分构成了进程映像（进程实体）。所谓创建进程，实质上是创建进程映像中的 PCB；而撤销进程，实质上是撤销进程的 PCB。值得注意的是，进程映像是静态的，进程则是动态的。

注意：PCB 是进程存在的唯一标志！

从不同的角度，进程可以有不同的定义，比较典型的定义有：

- 1) 进程是程序的一次执行过程。
- 2) 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。
- 3) 进程是具有独立功能的程序在一个数据集合上运行的过程，它是系统进行资源分配和调度的一个独立单位。

引入进程实体的概念后，我们可以把传统操作系统中的进程定义为：“进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位。”

读者要准确理解这里说的系统资源。它指处理机、存储器和其他设备服务于某个进程的“时间”，例如把处理机资源理解为处理机的时间片才是准确的。因为进程是这些资源分配和调度的独立单位，即“时间片”分配的独立单位，这就决定了进程一定是一个动态的、过程性的概念。

2. 进程的特征

进程是由多道程序的并发执行而引出的，它和程序是两个截然不同的概念。进程的基本特征是对比单个程序的顺序执行提出的，也是对进程管理提出的基本要求。

- 1) 动态性。进程是程序的一次执行，它有着创建、活动、暂停、终止等过程，具有一定的

- 生命周期，是动态地产生、变化和消亡的。动态性是进程最基本的特征。
- 2) 并发性。指多个进程实体同时存于内存中，能在一段时间内同时运行。并发性是进程的重要特征，同时也是操作系统的重要特征。引入进程的目的就是使程序能与其他进程的程序并发执行，以提高资源利用率。
 - 3) 独立性。指进程实体是一个能独立运行、独立获得资源和独立接受调度的基本单位。凡未建立 PCB 的程序，都不能作为一个独立的单位参与运行。
 - 4) 异步性。由于进程的相互制约，使得进程具有执行的间断性，即进程按各自独立的、不可预知的速度向前推进。异步性会导致执行结果的不可再现性，为此在操作系统中必须配置相应的进程同步机制。
 - 5) 结构性。每个进程都配置一个 PCB 对其进行描述。从结构上看，进程实体是由程序段、数据段和进程控制块三部分组成的。

通常不会直接考查进程有什么特性，所以读者对上面的 5 个特性不求记忆，只求理解。

2.1.2 进程的状态与转换

进程在其生命周期内，由于系统中各进程之间的相互制约关系及系统的运行环境的变化，使得进程的状态也在不断地发生变化（一个进程会经历若干不同状态）。通常进程有以下 5 种状态，前 3 种是进程的基本状态。

- 1) 运行态。进程正在处理机上运行。在单处理机环境下，每个时刻最多只有一个进程处于运行态。
- 2) 就绪态。进程获得了除处理机外的一切所需资源，一旦得到处理机，便可立即运行。系统中处于就绪状态的进程可能有多个，通常将它们排成一个队列，称为就绪队列。
- 3) 阻塞态，又称等待态。进程正在等待某一事件而暂停运行，如等待某资源为可用（不包括处理机）或等待输入/输出完成。即使处理机空闲，该进程也不能运行。
- 4) 创建态。进程正在被创建，尚未转到就绪态。创建进程通常需要多个步骤：首先申请一个空白的 PCB，并向 PCB 中填写一些控制和管理进程的信息；然后由系统为该进程分配运行时所必需的资源；最后把该进程转入就绪态。
- 5) 结束态。进程正从系统中消失，可能是进程正常结束或其他原因中断退出运行。进程需要结束运行时，系统首先必须将该进程置为结束态，然后进一步处理资源释放和回收等工作。

注意区别就绪态和等待态：就绪态是指进程仅缺少处理机，只要获得处理机资源就立即运行；而等待态是指进程需要其他资源（除了处理机）或等待某一事件。之所以把处理机和其他资源划分开，是因为在分时系统的时间片轮转机制中，每个进程分到的时间片是若干毫秒。也就是说，进程得到处理机的时间很短且非常频繁，进程在运行过程中实际上是频繁地转换到就绪态的；而其他资源（如外设）的使用和分配或某一事件的发生（如 I/O 操作的完成）对应的时间相对来说很长，进程转换到等待态的次数也相对较少。这样来看，就绪态和等待态是进程生命周期中两个完全不同的状态，显然需要加以区分。

图 2.1 说明了 5 种进程状态的转换，而 3 种基本状态之间的转换如下：

- 就绪态 → 运行态：处于就绪态的进程被调度后，获得处理机资源（分派处理机时间片），于是进程由就绪态转换为运行态。
- 运行态 → 就绪态：处于运行态的进程在时间片用完后，不得不让出处理机，从而进程由运行态转换为就绪态。此外，在可剥夺的操作系统中，当有更高优先级的进程就绪时，调度

程序将正在执行的进程转换为就绪态，让更高优先级的进程执行。

- 运行态→阻塞态：进程请求某一资源（如外设）的使用和分配或等待某一事件的发生（如 I/O 操作的完成）时，它就从运行态转换为阻塞态。进程以系统调用的形式请求操作系统提供服务，这是一种特殊的、由运行用户态程序调用操作系统内核过程的形式。
- 阻塞态→就绪态：进程等待的事件到来时，如 I/O 操作结束或中断结束时，中断处理程序必须把相应进程的状态由阻塞态转换为就绪态。



图 2.1 5 种进程状态的转换

需要注意的是，一个进程从运行态变成阻塞态是主动的行为，而从阻塞态变成就绪态是被动的行为，需要其他相关进程的协助。

2.1.3 进程控制

进程控制的主要功能是对系统中的所有进程实施有效的管理，它具有创建新进程、撤销已有进程、实现进程状态转换等功能。在操作系统中，一般把进程控制用的程序段称为原语，原语的特点是执行期间不允许中断，它是一个不可分割的基本单位。

1. 进程的创建

允许一个进程创建另一个进程。此时创建者称为父进程，被创建的进程称为子进程。子进程可以继承父进程所拥有的资源。当子进程被撤销时，应将其从父进程那里获得的资源归还给父进程。此外，在撤销父进程时，必须同时撤销其所有的子进程。

在操作系统中，终端用户登录系统、作业调度、系统提供服务、用户程序的应用请求等都会引起进程的创建。操作系统创建一个新进程的过程如下（创建原语）：

- 1) 为新进程分配一个唯一的进程标识号，并申请一个空白的 PCB (PCB 是有限的)。若 PCB 申请失败，则创建失败。
- 2) 为进程分配资源，为新进程的程序和数据及用户栈分配必要的内存空间（在 PCB 中体现）。注意，若资源不足（如内存空间），则并不是创建失败，而是处于阻塞态，等待内存资源。
- 3) 初始化 PCB，主要包括初始化标志信息、初始化处理机状态信息和初始化处理机控制信息，以及设置进程的优先级等。
- 4) 若进程就绪队列能够接纳新进程，则将新进程插入就绪队列，等待被调度运行。

2. 进程的终止

引起进程终止的事件主要有：① 正常结束，表示进程的任务已完成并准备退出运行。② 异常结束，表示进程在运行时，发生了某种异常事件，使程序无法继续运行，如存储区越界、保护错、非法指令、特权指令错、运行超时、算术运算错、I/O 故障等。③ 外界干预，指进程应外界的请求而终止运行，如操作员或操作系统干预、父进程请求和父进程终止。

操作系统终止进程的过程如下（撤销原语）：

- 1) 根据被终止进程的标识符，检索 PCB，从中读出该进程的状态。

- 2) 若被终止进程处于执行状态, 立即终止该进程的执行, 将处理机资源分配给其他进程。
- 3) 若该进程还有子孙进程, 则应将其所有子孙进程终止。
- 4) 将该进程所拥有的全部资源, 或归还给其父进程, 或归还给操作系统。
- 5) 将该 PCB 从所在队列(链表)中删除。

3. 进程的阻塞和唤醒

正在执行的进程, 由于期待的某些事件未发生, 如请求系统资源失败、等待某种操作的完成、新数据尚未到达或无新工作可做等, 由系统自动执行阻塞原语(Block), 使自己由运行态变为阻塞态。可见, 进程的阻塞是进程自身的一种主动行为, 也因此只有处于运行态的进程(获得 CPU), 才可能将其转为阻塞态。阻塞原语的执行过程如下:

- 1) 找到将要被阻塞进程的标识号对应的 PCB。
- 2) 若该进程为运行态, 则保护其现场, 将其状态转为阻塞态, 停止运行。
- 3) 把该 PCB 插入相应事件的等待队列, 将处理机资源调度给其他就绪进程。

当被阻塞进程所期待的事件出现时, 如它所启动的 I/O 操作已完成或其所期待的数据已到达, 由有关进程(比如, 释放该 I/O 设备的进程, 或提供数据的进程)调用唤醒原语(Wakeup), 将等待该事件的进程唤醒。唤醒原语的执行过程如下:

- 1) 在该事件的等待队列中找到相应进程的 PCB。
- 2) 将其从等待队列中移出, 并置其状态为就绪态。
- 3) 把该 PCB 插入就绪队列, 等待调度程序调度。

需要注意的是, Block 原语和 Wakeup 原语是一对作用刚好相反的原语, 必须成对使用。Block 原语是由被阻塞进程自我调用实现的, 而 Wakeup 原语则是由一个与被唤醒进程合作或被其他相关的进程调用实现的。

4. 进程切换

对于通常的进程而言, 其创建、撤销及要求由系统设备完成的 I/O 操作, 都是利用系统调用而进入内核, 再由内核中的相应处理程序予以完成的。进程切换同样是在内核的支持下实现的, 因此可以说, 任何进程都是在操作系统内核的支持下运行的, 是与内核紧密相关的。

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, 在这个过程中, 进程的运行环境产生了实质性的变化。进程切换的过程如下:

- 1) 保存处理机上下文, 包括程序计数器和其他寄存器。
- 2) 更新 PCB 信息。
- 3) 把进程的 PCB 移入相应的队列, 如就绪、在某事件阻塞等队列。
- 4) 选择另一个进程执行, 并更新其 PCB。
- 5) 更新内存管理的数据结构。
- 6) 恢复处理机上下文。

注意, 进程切换与处理机模式切换是不同的, 模式切换时, 处理机逻辑上可能还在同一进程中运行。若进程因中断或异常进入核心态运行, 执行完后又回到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 CPU 现场, 而无须改变当前进程的环境信息。但若要切换进程, 当前运行进程改变了, 则当前进程的环境信息也需要改变。

注意: “调度”和“切换”的区别。调度是指决定资源分配给哪个进程的行为, 是一种决策行为; 切换是指实际分配的行为, 是执行行为。一般来说, 先有资源的调度, 然后才有进程的切换。

2.1.4 进程的组织

进程是一个独立的运行单位，也是操作系统进行资源分配和调度的基本单位。它由以下三部分组成，其中最核心的是进程控制块（PCB）。

1. 进程控制块

进程创建时，操作系统为它新建一个 PCB，该结构之后常驻内存，任意时刻都可以存取，并在进程结束时删除。PCB 是进程实体的一部分，是进程存在的唯一标志。

进程执行时，系统通过其 PCB 了解进程的现行状态信息，以便对其进行控制和管理；进程结束时，系统收回其 PCB，该进程随之消亡。操作系统通过 PCB 表来管理和控制进程。

当操作系统欲调度某进程运行时，要从该进程的 PCB 中查出其现行状态及优先级；在调度到某进程后，要根据其 PCB 中所保存的处理机状态信息，设置该进程恢复运行的现场，并根据其 PCB 中的程序和数据的内存始址，找到其程序和数据；进程在运行过程中，当需要和与之合作的进程实现同步、通信或访问文件时，也需要访问 PCB；当进程由于某种原因而暂停运行时，又需将其断点的处理机环境保存在 PCB 中。可见，在进程的整个生命周期中，系统总是通过 PCB 对进程进行控制的，亦即系统唯有通过进程的 PCB 才能感知到该进程的存在。

表 2.1 是一个 PCB 的实例。PCB 主要包括进程描述信息、进程控制和管理信息、资源分配清单和处理机相关信息等。各部分的主要说明如下：

表 2.1 PCB 通常包含的内容

进程描述信息	进程控制和管理信息	资源分配清单	处理机相关信息
进程标识符（PID）	进程当前状态	代码段指针	通用寄存器值
用户标识符（UID）	进程优先级	数据段指针	地址寄存器值
	代码运行入口地址	堆栈段指针	控制寄存器值
	程序的外存地址	文件描述符	标志寄存器值
	进入内存时间	键盘	状态字
	处理机占用时间	鼠标	
	信号量使用		

- 1) 进程描述信息。进程标识符：标志各个进程，每个进程都有一个唯一的标识号。用户标识符：进程归属的用户，用户标识符主要为共享和保护服务。
- 2) 进程控制和管理信息。进程当前状态：描述进程的状态信息，作为处理机分配调度的依据。进程优先级：描述进程抢占处理机的优先级，优先级高的进程可优先获得处理机。
- 3) 资源分配清单，用于说明有关内存地址空间或虚拟地址空间的状况，所打开文件的列表和所使用的输入/输出设备信息。
- 4) 处理机相关信息，主要指处理机中各寄存器的值，当进程被切换时，处理机状态信息都必须保存在相应的 PCB 中，以便在该进程重新执行时，能从断点继续执行。

在一个系统中，通常存在着许多进程的 PCB，有的处于就绪态，有的处于阻塞态，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 PCB 用适当的方法组织起来。目前，常用的组织方式有链接方式和索引方式两种。链接方式将同一状态的 PCB 链接成一个队列，不同状态对应不同的队列，也可把处于阻塞态的进程的 PCB，根据其阻塞原因的不同，排成多个阻塞队列。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 PCB，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。

2. 程序段

程序段就是能被进程调度程序调度到 CPU 执行的程序代码段。注意，程序可被多个进程共享，即多个进程可以运行同一个程序。

3. 数据段

一个进程的数据段，可以是进程对应的程序加工处理的原始数据，也可以是程序执行时产生的中间或最终结果。

2.1.5 进程的通信

进程通信是指进程之间的信息交换。PV 操作是低级通信方式，高级通信方式是指以较高的效率传输大量数据的通信方式。高级通信方法主要有以下三类。

1. 共享存储

在通信的进程之间存在一块可直接访问的共享空间，通过对这片共享空间进行写/读操作实现进程之间的信息交换，如图 2.2 所示。在对共享空间进行写/读操作时，需要使用同步互斥工具（如 P 操作、V 操作），对共享空间的写/读进行控制。共享存储又分为两种：低级方式的共享是基于数据结构的共享；高级方式的共享则是基于存储区的共享。操作系统只负责为通信进程提供可共享使用的存储空间和同步互斥工具，而数据交换则由用户自己安排读/写指令完成。

注意，用户进程空间一般都是独立的，进程运行期间一般不能访问其他进程的空间，要想让两个用户进程共享空间，必须通过特殊的系统调用实现，而进程内的线程是自然共享进程空间的。

简单理解就是，甲和乙中间有一个大布袋，甲和乙交换物品是通过大布袋进行的，甲把物品放在大布袋里，乙拿走。但乙不能直接到甲的手中拿东西，甲也不能直接到乙的手中拿东西。

2. 消息传递

在消息传递系统中，进程间的数据交换是以格式化的消息（Message）为单位的。若通信的进程之间不存在可直接访问的共享空间，则必须利用操作系统提供的消息传递方法实现进程通信。进程通过系统提供的发送消息和接收消息两个原语进行数据交换。

- 1) 直接通信方式。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，如图 2.3 所示。

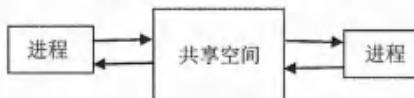


图 2.2 共享存储



图 2.3 消息传递

- 2) 间接通信方式。发送进程把消息发送到某个中间实体，接收进程从中间实体取得消息。这种中间实体一般称为信箱，这种通信方式又称信箱通信方式。该通信方式广泛应用于计算机网络中，相应的通信系统称为电子邮件系统。

简单理解就是，甲要告诉乙某些事情，就要写信，然后通过邮差送给乙。直接通信就是邮差把信直接送到乙的手上；间接通信就是乙家门口有一个邮箱，邮差把信放到邮箱里。

3. 管道通信

管道通信是消息传递的一种特殊方式（见图 2.4）。所谓“管道”，是指用于连接一个读进程和一个写进程以实现它们之间的通信的一个共享文件，又名 pipe 文件。向管道（共享文件）提供

输入的发送进程（即写进程），以字符流形式将大量的数据送入（写）管道；而接收管道输出的接收进程（即读进程）则从管道中接收（读）数据。为了协调双方的通信，管道机制必须提供以下三方面的协调能力：互斥、同步和确定对方的存在。

下面以 Linux 中的管道为例进行说明。在 Linux 中，管道是一种使用非常频繁的通信机制。从本质上说，管道也是一种文件，但它又和一般的文件有所不同，管道可以克服使用文件进行通信的两个问题，具体表现如下：

- 1) 限制管道的大小。实际上，管道是一个固定大小的缓冲区。在 Linux 中，该缓冲区的大小为 4KB，这使得它的大小不像文件那样不加检验地增长。使用单个固定缓冲区也会带来问题，比如在写管道时可能变满，这种情况发生时，随后对管道的 write() 调用将默认地被阻塞，等待某些数据被读取，以便腾出足够的空间供 write() 调用写。
- 2) 读进程也可能工作得比写进程快。当所有当前进程数据已被读取时，管道变空。当这种情况发生时，一个随后的 read() 调用将默认地被阻塞，等待某些数据被写入，这解决了 read() 调用返回文件结束的问题。

注意：从管道读数据是一次性操作，数据一旦被读取，它就从管道中被抛弃，释放空间以便写更多的数据。管道只能采用半双工通信，即某一时刻只能单向传输。要实现父子进程双方互动通信，需要定义两个管道。

管道可以理解为共享存储的优化和发展，因为在共享存储中，若某进程要访问共享存储空间，则必须没有其他进程在该共享存储空间中进行写操作，否则访问行为就会被阻塞。而管道通信中，存储空间进化成了缓冲区，缓冲区只允许一边写入、另一边读出，因此只要缓冲区中有数据，进程就能从缓冲区中读出，而不必担心会因为其他进程在其中进行写操作而遭到阻塞，因为写进程会先把缓冲区写满，然后才让读进程读，当缓冲区中还有数据时，写进程不会往缓冲区写数据。当然，这也决定了管道通信必然是半双工通信。

2.1.6 线程概念和多线程模型

1. 线程的基本概念

引入进程的目的是更好地使多道程序并发执行，提高资源利用率和系统吞吐量；而引入线程的目的则是减小程序在并发执行时所付出的时空开销，提高操作系统的并发性能。

线程最直接的理解就是“轻量级进程”，它是一个基本的 CPU 执行单元，也是程序执行流的最小单元，由线程 ID、程序计数器、寄存器集合和堆栈组成。线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其他线程共享进程所拥有的全部资源。一个线程可以创建和撤销另一个线程，同一进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。线程也有就绪、阻塞和运行三种基本状态。

引入线程后，进程的内涵发生了改变，进程只作为除 CPU 外的系统资源的分配单元，而线程则作为处理机的分配单元。由于一个进程内部有多个线程，若线程的切换发生在同一个进程内部，则只需要很少的时空开销。

2. 线程与进程的比较

- 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。在引入线程的

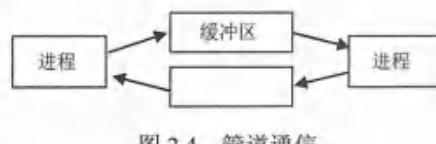


图 2.4 管道通信

操作系统中，线程是独立调度的基本单位，进程是拥有资源的基本单位。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，会引起进程切换。

- 2) 拥有资源。不论是传统操作系统还是设有线程的操作系统，进程都是拥有资源的基本单位，而线程不拥有系统资源（也有一点儿必不可少的资源），但线程可以访问其隶属进程的系统资源。要知道，若线程也是拥有资源的单位，则切换线程就需要较大的时空开销，线程这个概念的提出就没有意义。
- 3) 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，提高了系统的吞吐量。
- 4) 系统开销。由于创建或撤销进程时，系统都要为之分配或回收资源，如内存空间、I/O 设备等，因此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，涉及当前执行进程 CPU 环境的保存及新调度到进程 CPU 环境的设置，而线程切换时只需保存和设置少量寄存器内容，开销很小。此外，由于同一进程内的多个线程共享进程的地址空间，因此这些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。
- 5) 地址空间和其他资源（如打开的文件）。进程的地址空间之间互相独立，同一进程的各线程间共享进程的资源，某进程内的线程对于其他进程不可见。
- 6) 通信方面。进程间通信（IPC）需要进程同步和互斥手段的辅助，以保证数据的一致性，而线程间可以直接读/写进程数据段（如全局变量）来进行通信。

3. 线程的属性

多线程操作系统把线程作为独立运行（或调度）的基本单位，此时的进程已不再是一个基本的可执行实体，但它仍具有与执行相关状态。所谓进程处于“执行”状态，实际上是指该进程中的某线程正在执行。线程的主要属性如下：

- 1) 线程是一个轻型实体，它不拥有系统资源，但每个线程都应有一个唯一的标识符和一个线程控制块，线程控制块记录了线程执行的寄存器和栈等现场状态。
- 2) 不同的线程可以执行相同的程序，即同一个服务程序被不同的用户调用时，操作系统把它们建成不同的线程。
- 3) 同一进程中的各个线程共享该进程所拥有的资源。
- 4) 线程是处理机的独立调度单位，多个线程是可以并发执行的。在单 CPU 的计算机系统中，各线程可交替地占用 CPU；在多 CPU 的计算机系统中，各线程可同时占用不同的 CPU，若各个 CPU 同时为一个进程内的各线程服务，则可缩短进程的处理时间。
- 5) 一个线程被创建后，便开始了它的生命周期，直至终止。线程在生命周期内会经历阻塞态、就绪态和运行态等各种状态变化。

为什么线程的提出有利于提高系统并发性？可以这样来理解：由于有了线程，线程切换时，有可能会发生进程切换，也有可能不发生进程切换，平均而言每次切换所需的开销就变小了，因此能够让更多的线程参与并发，而不会影响到响应时间等问题。

4. 线程的实现方式

线程的实现可以分为两类：用户级线程（User-Level Thread,ULT）和内核级线程（Kernel-Level Thread, KLT）。内核级线程又称内核支持的线程。

在用户级线程中，有关线程管理（线程的创建、撤销和切换等）的所有工作都由应用程序完

成，内核意识不到线程的存在。应用程序可以通过使用线程库设计成多线程程序。通常，应用程序从单线程开始，在该线程中开始运行，在其运行的任何时刻，可以通过调用线程库中的派生例程创建一个在相同进程中运行的新线程。图 2.5(a)说明了用户级线程的实现方式。

在内核级线程中，线程管理的所有工作由内核完成，应用程序没有进行线程管理的代码，只有一个到内核级线程的编程接口。内核为进程及其内部的每个线程维护上下文信息，调度也在内核基于线程架构的基础上完成。图 2.5(b)说明了内核级线程的实现方式。

有些系统中使用组合方式的多线程实现。线程创建完全在用户空间中完成，线程的调度和同步也在应用程序中进行。一个应用程序中的多个用户级线程被映射到一些（小于等于用户级线程的数目）内核级线程上。图 2.5(c)说明了用户级与内核级的组合实现方式。

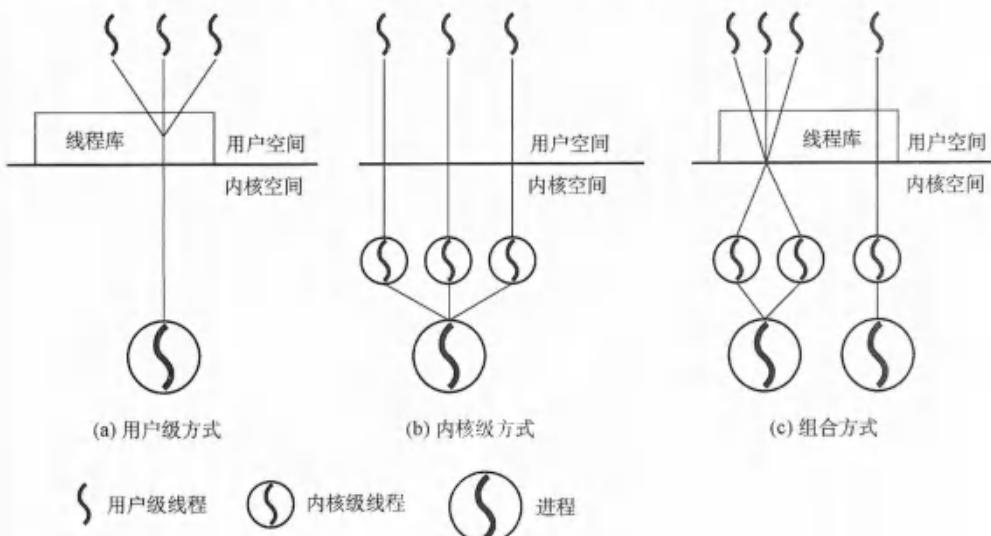


图 2.5 用户级和内核级线程

5. 多线程模型

有些系统同时支持用户线程和内核线程，由此产生了不同的多线程模型，即实现用户级线程和内核级线程的连接方式。

1) 多对一模型。将多个用户级线程映射到一个内核级线程，线程管理在用户空间完成。此模式中，用户级线程对操作系统不可见（即透明）。

优点：线程管理是在用户空间进行的，因而效率比较高。

缺点：一个线程在使用内核服务时被阻塞，整个进程都会被阻塞；多个线程不能并行地运行在多处理机上。

2) 一对一模型。将每个用户级线程映射到一个内核级线程。

优点：当一个线程被阻塞后，允许另一个线程继续执行，所以并发能力较强。

缺点：每创建一个用户级线程都需要创建一个内核级线程与其对应，这样创建线程的开销比较大，会影响到应用程序的性能。

3) 多对多模型。将 n 个用户级线程映射到 m 个内核级线程上，要求 $m \leq n$ 。

特点：多对多模型是多对一模型和一对一模型的折中，既克服了多对一模型并发度不高的缺点，又克服了一对一模型的一个用户进程占用太多内核级线程而开销太大的缺点。

此外，还拥有多对一模型和一对一模型各自的优点，可谓集两者之所长。

2.1.7 本节小结

本节开头提出的问题的参考答案如下。

1) 为什么要引入进程?

在多道程序同时运行的背景下,进程之间需要共享系统资源,因此会导致各程序在执行过程中出现相互制约的关系,程序的执行会表现出间断性的特征。这些特征都是在程序的执行过程中发生的,是动态的过程,而传统的程序本身是一组指令的集合,是一个静态的概念,无法描述程序在内存中的执行情况,即我们无法从程序的字面上看出它何时执行、何时停顿,也无法看出它与其他执行程序的关系,因此,程序这个静态概念已不能如实反映程序并发执行过程的特征。为了深刻描述程序动态执行过程的性质乃至更好地支持和管理多道程序的并发执行,人们引入了进程的概念。

2) 什么是进程?进程由什么组成?

进程是一个具有独立功能的程序关于某个数据集合的一次运行活动。它可以申请和拥有系统资源,是一个动态的概念,是一个活动的实体。它不只是程序的代码本身,还包括当前的活动,通过程序计数器的值和处理寄存器的内容来表示。

一个进程实体由程序段、相关数据段和 PCB 三部分构成,其中 PCB 是标志一个进程存在的唯一标识,程序段是进程运行的程序的代码,数据段则存储程序运行过程中相关的一些数据。

3) 进程是如何解决问题的?

进程把能够识别程序运行态的一些变量存放在 PCB 中,通过这些变量系统能够更好地了解进程的状况,并在适当时进行进程的切换,以避免一些资源的浪费,甚至划分为更小的调度单位一线程来提高系统的并发度。

本节主要介绍什么是进程,并围绕这个问题进行一些阐述和讨论,为下一节讨论的内容做铺垫,但之前未学过相关课程的读者可能会比较费解,到现在为止对进程这个概念还未形成比较清晰的认识。接下来,我们再用一个比较熟悉的概念来类比进程,以便大家能彻底理解本节的内容到底在讲什么,到底解决了什么问题。

我们用“人的生命历程”来类比进程。首先,人的生命历程一定是一个动态的、过程性的概念,要研究人的生命历程,先要介绍经历这个历程的主体是什么。主体当然是人,相当于经历进程的主体是进程映像,人有自己的身份,相当于进程映像里有 PCB;人生历程会经历好几种状态:出生的时候、弥留的时候、充满斗志的时候、发奋图强的时候及失落的时候,相当于进程有创建、撤销、就绪、运行、阻塞等状态,这几种状态会发生改变,人会充满斗志而转向发奋图强,发奋图强获得进步之后又会充满斗志预备下一次发奋图强,或者发奋图强后遇到阻碍会进入失落状态,然后在别人的开导之下又重新充满斗志。类比进程,会由就绪态转向运行态,运行态转向就绪态,或者运行态转向阻塞态,然后在别的进程帮助下返回就绪态。

若我们用“人生历程”这个过程的概念去类比进程,则对进程的理解就会更深一层。前面生活化的例子可以帮我们理解进程的实质,但它毕竟有不严谨的地方。一种较好的方式是,在类比进程和“人生历程”后,再看一遍前面较为严谨的书面阐述和讨论,这样对知识的掌握会更加准确而全面。

这里再给出一些学习计算机科学知识的建议。学习科学知识时,很多同学会陷入一个误区,即只注重对定理、公式的应用,而忽视对基础概念的理解。这是我们从小到大为了应付考试而培养出的一个毛病,因为熟练应用公式和定理对考试有立竿见影的效果。公式、定理的应用固然重要,但基础概念的理解能让我们透彻地理解一门学科,更利于我们产生兴趣,培养创造性思维。

2.1.8 本节习题精选

一、单项选择题

1. 一个进程映像是（ ）。

A. 由协处理器执行的一个程序	B. 一个独立的程序 + 数据集
C. PCB 结构与程序和数据的组合	D. 一个独立的程序
2. 下列关于线程的叙述中，正确的是（ ）。

A. 线程包含 CPU 现场，可以独立执行程序	B. 每个线程有自己独立的地址空间
C. 进程只能包含一个线程	D. 线程之间的通信必须使用系统调用函数
3. 进程之间交换数据不能通过（ ）途径进行。

A. 共享文件	B. 消息传递
C. 访问进程地址空间	D. 访问共享存储区
4. 进程与程序的根本区别是（ ）。

A. 静态和动态特点	B. 是不是被调入内存
C. 是不是具有就绪、运行和等待三种状态	D. 是不是占有处理器
5. 下面的叙述中，正确的是（ ）。

A. 进程获得处理器运行是通过调度得到的	B. 优先级是进程调度的重要依据，一旦确定不能改动
C. 在单处理器系统中，任何时刻都只有一个进程处于运行态	D. 进程申请处理器而得不到满足时，其状态变为阻塞态
6. 操作系统是根据（ ）来对并发执行的进程进行控制和管理的。

A. 进程的基本状态	B. 进程控制块
C. 多道程序设计	D. 进程的优先权
7. 在任何时刻，一个进程的状态变化（ ）引起另一个进程的状态变化。

A. 必定	B. 一定不	C. 不一定	D. 不可能
-------	--------	--------	--------
8. 在单处理器系统中，若同时存在 10 个进程，则处于就绪队列中的进程最多有（ ）个。

A. 1	B. 8	C. 9	D. 10
------	------	------	-------
9. 一个进程释放了一台打印机，它可能会改变（ ）的状态。

A. 自身进程	B. 输入/输出进程
C. 另一个等待打印机的进程	D. 所有等待打印机的进程
10. 系统进程所请求的一次 I/O 操作完成后，将使进程状态从（ ）。

A. 运行态变为就绪态	B. 运行态变为阻塞态
C. 就绪态变为运行态	D. 阻塞态变为就绪态
11. 一个进程的基本状态可以从其他两种基本状态转变过去，这个基本的状态一定是（ ）。

A. 执行状态	B. 阻塞态	C. 就绪态	D. 完成状态
---------	--------	--------	---------
12. 并发进程失去封闭性，是指（ ）。

A. 多个相对独立的进程以各自的速度向前推进	B. 并发进程的执行结果与速度无关
C. 并发进程执行时，在不同时刻发生的错误	

- D. 并发进程共享变量，其执行结果与速度有关
13. 通常用户进程被建立后，()。
- A. 便一直存在于系统中，直到被操作人员撤销
 - B. 随着进程运行的正常或不正常结束而撤销
 - C. 随着时间片轮转而撤销与建立
 - D. 随着进程的阻塞或者唤醒而撤销与建立
14. 进程在处理器上执行时，()。
- A. 进程之间是无关的，具有封闭特性
 - B. 进程之间都有交互性，相互依赖、相互制约，具有并发性
 - C. 具有并发性，即同时执行的特性
 - D. 进程之间可能是无关的，但也可能是有交互性的
15. 下面的说法中，正确的是()。
- A. 不论是系统支持的线程还是用户级线程，其切换都需要内核的支持
 - B. 线程是资源分配的单位，进程是调度和分派的单位
 - C. 不管系统中是否有线程，进程都是拥有资源的独立单位
 - D. 在引入线程的系统中，进程仍是资源调度和分派的基本单位
16. 在多对一的线程模型中，当一个多线程进程中的某个线程被阻塞后，()。
- A. 该进程的其他线程仍可继续运行
 - B. 整个进程都将阻塞
 - C. 该阻塞线程将被撤销
 - D. 该阻塞线程将永远不可能再执行
17. 用信箱实现进程间互通信息的通信机制要有两个通信原语，它们是()。
- A. 发送原语和执行原语
 - B. 就绪原语和执行原语
 - C. 发送原语和接收原语
 - D. 就绪原语和接收原语
18. 下列几种关于进程的叙述，()最不符合操作系统对进程的理解。
- A. 进程是在多程序环境中的完整程序
 - B. 进程可以由程序、数据和 PCB 描述
 - C. 线程(Thread)是一种特殊的进程
 - D. 进程是程序在一个数据集合上的运行过程，它是系统进行资源分配和调度的一个独立单元
19. 支持多道程序设计的操作系统在运行过程中，不断地选择新进程运行来实现 CPU 的共享，但其中()不是引起操作系统选择新进程的直接原因。
- A. 运行进程的时间片用完
 - B. 运行进程出错
 - C. 运行进程要等待某一事件发生
 - D. 有新进程进入就绪态
20. 若一个进程实体由 PCB、共享正文段、数据堆段和数据栈段组成，请指出下列 C 语言程序中的内容及相关数据结构各位于哪一段中。
- I. 全局赋值变量() II. 未赋值的局部变量()
 - III. 函数调用实参传递值() IV. 用 malloc()要求动态分配的存储区()
 - V. 常量值(如 1995、"string") () VI. 进程的优先级()
 - A. PCB B. 正文段 C. 堆段 D. 栈段
21. 同一程序经过多次创建，运行在不同的数据集上，形成了()的进程。
- A. 不同 B. 相同 C. 同步 D. 互斥
22. 系统动态 DLL 库中的系统线程，被不同的进程所调用，它们是()的线程。

- A. 不同 B. 相同
 C. 可能不同，也可能相同 D. 不能被调用
23. PCB 是进程存在的唯一标志，下列（）不属于 PCB。
 A. 进程 ID B. CPU 状态 C. 堆栈指针 D. 全局变量
24. 一个计算机系统中，进程的最大数目主要受到（）限制。
 A. 内存大小 B. 用户数目 C. 打开的文件数 D. 外部设备数量
25. 进程创建完成后会进入一个序列，这个序列称为（）。
 A. 阻塞队列 B. 挂起序列 C. 就绪队列 D. 运行队列
26. 在一个多道系统中，若就绪队列不空，就绪的进程数目越多，处理器的效率（）。
 A. 越高 B. 越低 C. 不变 D. 不确定
27. 在具有通道设备的单处理器系统中实现并发技术后，（）。
 A. 各进程在某一时刻并行运行，CPU 与 I/O 设备间并行工作
 B. 各进程在某一时间段内并行运行，CPU 与 I/O 设备间串行工作
 C. 各进程在某一时间段内并发运行，CPU 与 I/O 设备间并行工作
 D. 各进程在某一时刻并发运行，CPU 与 I/O 设备间串行工作
28. 进程自身决定（）。
 A. 从运行态到阻塞态 B. 从运行态到就绪态
 C. 从就绪态到运行态 D. 从阻塞态到就绪态
29. 对进程的管理和控制使用（）。
 A. 指令 B. 原语 C. 信号量 D. 信箱
30. 【2010 统考真题】下列选项中，导致创建新进程的操作是（）。
 I. 用户登录成功 II. 设备分配 III. 启动程序执行
 A. 仅 I 和 II B. 仅 II 和 III C. 仅 I 和 III D. I、II、III
31. 下面的叙述中，正确的是（）。
 A. 引入线程后，处理器只能在线程间切换
 B. 引入线程后，处理器仍在进程间切换
 C. 线程的切换，不会引起进程的切换
 D. 线程的切换，可能引起进程的切换
32. 下面的叙述中，正确的是（）。
 A. 线程是比进程更小的能独立运行的基本单位，可以脱离进程独立运行
 B. 引入线程可提高程序并发执行的程度，可进一步提高系统效率
 C. 线程的引入增加了程序执行时的时空开销
 D. 一个进程一定包含多个线程
33. 下面的叙述中，正确的是（）。
 A. 同一进程内的线程可并发执行，不同进程的线程只能串行执行
 B. 同一进程内的线程只能串行执行，不同进程的线程可并发执行
 C. 同一进程或不同进程内的线程都只能串行执行
 D. 同一进程或不同进程内的线程都可以并发执行
34. 【2014 统考真题】在支持多线程的系统中，进程 P 创建的若干线程不能共享的是（）。
 A. 进程 P 的代码段 B. 进程 P 中打开的文件
 C. 进程 P 的全局变量 D. 进程 P 中某线程的栈指针

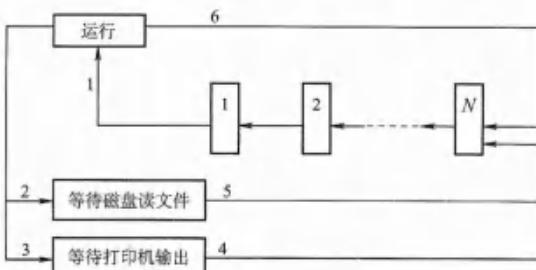
35. 在以下描述中，() 并不是多线程系统的特长。
- A. 利用线程并行地执行矩阵乘法运算
 - B. Web 服务器利用线程响应 HTTP 请求
 - C. 键盘驱动程序为每个正在运行的应用配备一个线程，用以响应该应用的键盘输入
 - D. 基于 GUI 的调试程序用不同的线程分别处理用户输入、计算和跟踪等操作
36. 【2012 统考真题】下列关于进程和线程的叙述中，正确的是 ()。
- A. 不管系统是否支持线程，进程都是资源分配的基本单位
 - B. 线程是资源分配的基本单位，进程是调度的基本单位
 - C. 系统级线程和用户级线程的切换都需要内核的支持
 - D. 同一进程中的各个线程拥有各自不同的地址空间
37. 在进程转换时，下列 () 转换是不可能发生的。
- A. 就绪态 → 运行态
 - B. 运行态 → 就绪态
 - C. 运行态 → 阻塞态
 - D. 阻塞态 → 运行态
38. 当 () 时，进程从执行状态转变为就绪态。
- A. 进程被调度程序选中
 - B. 时间片到
 - C. 等待某一事件
 - D. 等待的事件发生
39. 两个合作进程 (Cooperating Processes) 无法利用 () 交换数据。
- A. 文件系统
 - B. 共享内存
 - C. 高级语言程序设计中的全局变量
 - D. 消息传递系统
40. 以下可能导致一个进程从运行态变为就绪态的事件是 ()。
- A. 一次 I/O 操作结束
 - B. 运行进程需做 I/O 操作
 - C. 运行进程结束
 - D. 出现了比现在进程优先级更高的进程
41. () 必会引起进程切换。
- A. 一个进程创建后，进入就绪态
 - B. 一个进程从运行态变为就绪态
 - C. 一个进程从阻塞态变为就绪态
 - D. 以上答案都不对
42. 进程处于 () 时，它处于非阻塞态。
- A. 等待从键盘输入数据
 - B. 等待协作进程的一个信号
 - C. 等待操作系统分配 CPU 时间
 - D. 等待网络数据进入内存
43. 【2010 统考真题】下列选项中，降低进程优先级的合理时机是 ()。
- A. 进程时间片用完
 - B. 进程刚完成 I/O 操作，进入就绪队列
 - C. 进程长期处于就绪队列
 - D. 进程从就绪态转为运行态
44. 一个进程被唤醒，意味着 ()。
- A. 该进程可以重新竞争 CPU
 - B. 优先级变大
 - C. PCB 移动到就绪队列之首
 - D. 进程变为运行态
45. 进程创建时，不需要做的是 ()。
- A. 填写一个该进程的进程表项
 - B. 分配该进程适当的内存
 - C. 将该进程插入就绪队列
 - D. 为该进程分配 CPU
46. 计算机两个系统中两个协作进程之间不能用来进行进程间通信的是 ()。
- A. 数据库
 - B. 共享内存
 - C. 消息传递机制
 - D. 管道

47. 下列说法中，不正确的是（ ）。
- A. 一个进程可以创建一个或多个线程
 - B. 一个线程可以创建一个或多个线程
 - C. 一个线程可以创建一个或多个进程
 - D. 一个进程可以创建一个或多个进程
48. 【2014 统考真题】一个进程的读磁盘操作完成后，操作系统针对该进程必做的是（ ）。
- A. 修改进程状态为就绪态
 - B. 降低进程优先级
 - C. 给进程分配用户内存空间
 - D. 增加进程时间片大小
49. 【2014 统考真题】下列关于管道（Pipe）通信的叙述中，正确的是（ ）。
- A. 一个管道可实现双向数据传输
 - B. 管道的容量仅受磁盘容量大小限制
 - C. 进程对管道进行读操作和写操作都可能被阻塞
 - D. 一个管道只能有一个读进程或一个写进程对其操作
50. 【2015 统考真题】下列选项中，会导致进程从执行态变为就绪态的事件是（ ）。
- A. 执行 P(wait) 操作
 - B. 申请内存失败
 - C. 启动 I/O 设备
 - D. 被高优先级进程抢占
51. 【2018 统考真题】下列选项中，可能导致当前进程 P 阻塞的事件是（ ）。
- I. 进程 P 申请临界资源
 - II. 进程 P 从磁盘读数据
 - III. 系统将 CPU 分配给高优先权的进程
- A. 仅 I
 - B. 仅 II
 - C. 仅 I、II
 - D. I、II、III
52. 【2019 统考真题】下列选项中，可能会将进程唤醒的事件是（ ）。
- I. I/O 结束
 - II. 某进程退出临界区
 - III. 当前进程的时间片用完
- A. 仅 I
 - B. 仅 III
 - C. 仅 I、II
 - D. I、II、III
53. 【2019 统考真题】下列关于线程的描述中，错误的是（ ）。
- A. 内核级线程的调度由操作系统完成
 - B. 操作系统为每个用户级线程建立一个线程控制块
 - C. 用户级线程间的切换比内核级线程间的切换效率高
 - D. 用户级线程可以在不支持内核级线程的操作系统上实现
54. 【2020 统考真题】下列关于父进程与子进程的叙述中，错误的是（ ）。
- A. 父进程与子进程可以并发执行
 - B. 父进程与子进程共享虚拟地址空间
 - C. 父进程与子进程有不同的进程控制块
 - D. 父进程与子进程不能同时使用同一临界资源

二、综合应用题

1. 进程和程序之间可以形成一对一、一对多、多对一、多对多的关系，请分别举例说明在什么情况下会形成这样的关系。
2. 父进程创建子进程和主程序调用子程序有何不同？
3. 为什么进程之间的通信必须借助于操作系统内核功能？简单说明进程通信的几种主要方式。
4. 什么是多线程？多线程与多任务有什么区别？
5. 回答下列问题：
 - 1) 若系统中没有运行进程，是否一定没有就绪进程？为什么？
 - 2) 若系统中既没有运行进程，又没有就绪进程，系统中是否就没有进程？为什么？

- 3) 在采用优先级进程调度时, 运行进程是否一定是系统中优先级最高的进程?
6. 现代操作系统一般都提供多进程(或称多任务)运行环境, 回答以下问题:
- 1) 为支持多进程的并发执行, 系统必须建立哪些关于进程的数据结构?
 - 2) 为支持进程状态的变迁, 系统至少应提供哪些进程控制原语?
 - 3) 执行每个进程控制原语时, 进程状态发生什么变化? 相应的数据结构发生什么变化?
7. 某分时系统中的进程可能出现如下图所示的状态变化, 请回答下列问题:
- 1) 根据图示, 该系统应采用什么进程调度策略?
 - 2) 把图中每个状态变化的可能原因填写在下表中。



变化	原 因
1	
2	
3	
4	
5	
6	

2.1.9 答案与解析

一、单项选择题

1. C

进程映像是 PCB、程序段和数据的组合, 其中 PCB 是进程存在的唯一标志。

2. A

线程是处理机调度的基本单位, 当然可以独立执行程序, A 对; 线程没有自己独立的地址空间, 它共享其所属进程的空间, B 错; 进程可以创建多个线程, C 错; 与进程之间线程的通信可以直接通过它们共享的存储空间, D 错。

3. C

每个进程包含独立的地址空间, 进程各自的地址空间是私有的, 只能执行自己地址空间中的程序, 且只能访问自己地址空间中的数据, 相互访问会导致指针的越界错误(学完内存管理将会有更好的认识)。因此, 进程之间不能直接交换数据, 但可利用操作系统提供的共享文件、消息传递、共享存储区等进行通信。

4. A

动态性是进程最重要的特性, 以此来区分文件形式的静态程序。操作系统引入进程的概念, 是为了从变化的角度动态地分析和研究程序的执行。

5. A

选项 B 错在优先级分静态和动态两种, 动态优先级是根据运行情况而随时调整的。选项 C 错在系统发生死锁时有可能进程全部都处于阻塞态, 或无进程任务, CPU 空闲。选项 D 错在进程申请处理器得不到满足时就处于就绪态, 等待处理器的调度。

6. B

在进程的整个生命周期中, 系统总是通过其 PCB 对进程进行控制。也就是说, 系统是根据进程的 PCB 而非任何其他因素来感知到进程存在的, PCB 是进程存在的唯一标志。同时 PCB 常驻内存。A 和 D 选项的内容都包含在进程 PCB 中。

7. C

一个进程的状态变化可能会引起另一个进程的状态变化。例如，一个进程时间片用完，可能会引起另一个就绪进程的运行。同时，一个进程的状态变化也可能不会引起另一个进程的状态变化。例如，一个进程由阻塞态转变为就绪态就不会引起其他进程的状态变化。

8. C

不可能出现这样一种情况。单处理器系统的 10 个进程都处于就绪态，但 9 个处于就绪态、1 个正在运行是可能存在的。还要想到，可能 10 个进程都处于阻塞态。

9. C

由于打印机独占资源，当一个进程释放打印机后，另一个等待打印机的进程就可能从阻塞态转到就绪态。

当然，也存在一个进程执行完毕后由运行态转为结束态时释放打印机的情况，但这并不是由于释放打印机引起的，相反是因为运行完成才释放了打印机。

10. D

I/O 操作完成之前进程在等待结果，状态为阻塞态；完成后进程等待事件就绪，变为就绪态。

11. C

只有就绪态可以既由运行态转变过去又能由阻塞态转变过去。时间片到，运行态变为就绪态；当所需要资源到达时，进程由阻塞态转变为就绪态。

12. D

程序封闭性是指进程执行的结果只取决于进程本身，不受外界影响。也就是说，进程在执行过程中不管是不停顿地执行，还是走走停停，进程的执行速度都不会改变它的执行结果。失去封闭性后，不同速度下的执行结果不同。

13. B

进程有它的生命周期，不会一直存在于系统中，也不一定需要用户显式地撤销。进程在时间片结束时只是就绪，而不是撤销。阻塞和唤醒是进程生存期的中间状态。进程可在完成时撤销，或在出现内存错误等时撤销。

14. D

A 和 B 都说得太绝对，进程之间有可能具有相关性，也有可能是相互独立的。C 错在“同时”。

15. C

引入线程后，进程仍然是资源分配的单位。线程是处理器调度和分派的单位，线程本身不具有资源，它可以共享所属进程的全部资源，C 对，B、D 明显是错的。至于 A，可以这样来理解：假如有一个内核进程，它映射到用户级后有多个线程，那么这些线程之间的切换不需要在内核级切换进程，也就不需要内核的支持。

16. B

在多对一的线程模型中，用户级线程的“多”对操作系统透明，即操作系统并不知道用户有多少线程。因此该进程的一个线程被阻塞后，该进程就被阻塞，进程的其他线程当然也都被阻塞。

17. C

用信箱实现进程间互通信息的通信机制要有两个通信原语，它们是发送原语和接收原语。

18. A

进程是一个独立的运行单位，也是操作系统进行资源分配和调度的基本单位，它包括 PCB、程序和数据以及执行栈区，仅仅说进程是在多程序环境下的完整程序是不合适的，因为程序是静态的，它以文件形式存放于计算机硬盘内，而进程是动态的。

19. D

这道题实际上问的是，哪种情况下进程不从运行态转换成其他状态。

20. B、D、D、C、B、A

C 语言编写的程序在使用内存时一般分为三个段，它们一般是正文段（即代码和赋值数据段）、数据堆段和数据栈段。二进制代码和常量存放在正文段，动态分配的存储区在数据堆段，临时使用的变量在数据栈段。由此，我们可以确定全局赋值变量在正文段赋值数据段，未赋值的局部变量和实参传递在栈段，动态内存分配在堆段，常量在正文段，进程的优先级只能在 PCB 内。

21. A

一个进程是程序在一个数据集上的一次运行过程。运行于不同的数据集，将会形成不同的进程。

22. B

进程是暂时的，程序是永久的；进程是动态的，程序是静态的；进程至少由代码、数据和 PCB 组成，程序仅需代码和数据即可；程序代码经过多次创建可对应不同的进程，而同一个系统的进程（或线程）可以由系统调用的方法被不同的进程（或线程）多次使用。

23. D

进程实体主要是代码、数据和 PCB。因此，要清楚了解 PCB 内所含的数据结构内容，主要有四大类：进程标志信息、进程控制信息、进程资源信息、CPU 现场信息。由上述可知，全局变量与 PCB 无关，它只与用户代码有关。

24. A

进程创建需要占用系统内存来存放 PCB 的数据结构，所以一个系统能够创建的进程总数是有限的，进程的最大数目取决于系统内存的大小，它在系统安装时就已确定（若后期内存增加，系统能够创建的进程总数也应增加，但一般需要重新启动）。而用户数目、外设数量和文件等均与此无关。

25. C

我们先要考虑创建进程的过程，当该进程所需的资源分配完成只等 CPU 时，进程的状态为就绪态，因此所有的就绪 PCB 一般以链表方式链成一个序列，称为就绪队列。

26. C

由进程的状态图（见图 2.1）可以看出，进程的就绪数目越多，争夺 CPU 的进程就越多，但只要就绪队列不为空，CPU 就总是可以调度进程运行，保持繁忙。这与就绪进程的数目没有关系，除非就绪队列为空，此时 CPU 进入等待态，导致 CPU 的效率下降。

27. C

由于是单处理器，在某一时刻只有一个进程能获得处理器资源，所以是某一时间段内并发运行。此外，也正是因为 CPU 和 I/O 设备的并行运行，才使各进程能并发执行。

28. A

只有从运行态到阻塞态的转换是由进程自身决定的。从运行态到就绪态的转换是由于进程的时间片用完，“主动”调用程序转向就绪态。虽然从就绪态到运行态的转换同样是由调度程序决定的，但进程是“被动的”。从阻塞态到就绪态的转换是由协作进程决定的。

29. B

对进程的管理和控制功能是通过执行各种原语来实现的，如创建原语等。

30. C

I. 用户登录成功后，系统要为此创建一个用户管理的进程，包括用户桌面、环境等。所有用

户进程都会在该进程中创建和管理。II. 设备分配是通过在系统中设置相应的数据结构实现的，不需要创建进程，这是操作系统中 I/O 核心子系统的内容。III. 启动程序执行是引起创建进程的典型事件。

31. D

在同一进程中，线程的切换不会引起进程的切换。当从一个进程中的线程切换到另一个进程中的线程时，才会引起进程的切换，因此 A、B、C 错误。

32. B

线程是进程内一个相对独立的执行单元，但不能脱离进程单独运行，只能在进程中运行。引入线程是为了减少程序执行时的时空开销。一个进程可包含一个或多个线程。

33. D

在无线程的系统中，进程是资源调度和并发执行的基本单位。在引入线程的系统中，进程退化为资源分配的基本单位，而线程代替了进程被操作系统调度，因而线程可以并发执行。

34. D

进程中的线程共享进程内的全部资源，但进程中某线程的栈指针对其他线程是透明的，不能与其他线程共享。

35. C

整个系统只有一个键盘，而且键盘输入是人的操作，速度比较慢，完全可以使用一个线程来处理整个系统的键盘输入。

36. A

在引入线程后，进程依然是资源分配的基本单位，线程是调度的基本单位，同一进程中的各个线程共享进程的地址空间。在用户级线程中，有关线程管理的所有工作都由应用程序完成，无须内核的干预，内核意识不到线程的存在。

37. D

阻塞的进程在获得所需资源时只能由阻塞态转变为就绪态，并插入就绪队列，而不能直接转变为运行态。

38. B

当进程的时间片到时，进程由运行态转变为就绪态，等待下一个时间片的到来。

39. C

不同的进程拥有不同的代码段和数据段，全局变量是对同一进程而言的，在不同的进程中是不同的变量，没有任何联系，所以不能用于交换数据。此题也可用排除法做，A、B、D 均是课本上所讲的。管道是一种文件。

40. D

进程处于运行态时，它必须已获得所需的资源，在运行结束后就撤销。只有在时间片到或出现了比现在进程优先级更高的进程时才转变成就绪态。选项 A 使进程从阻塞态到就绪态，选项 B 使进程从运行态到阻塞态，选项 C 使进程撤销。

41. B

进程切换是指 CPU 调度不同的进程执行，当一个进程从运行态变为就绪态时，CPU 调度另一个进程执行，引起进程切换。

42. C

进程有三种基本状态，处于阻塞态的进程由于某个事件不满足而等待。这样的事件一般是 I/O 操作，如键盘等，或是因互斥或同步数据引起的等待，如等待信号或等待进入互斥临界区代码段

等，等待网络数据进入内存是为了进程同步。而等待 CPU 调度的进程处于就绪态，只有它是非阻塞态。

43. A

A 中进程时间片用完，可降低其优先级以让其他进程被调度进入执行状态。B 中进程刚完成 I/O，进入就绪队列等待被处理机调度，为了让其尽快处理 I/O 结果，因此应提高优先级。C 中进程长期处于就绪队列，为不至于产生饥饿现象，也应适当提高优先级。D 中进程的优先级不应该在此时降低，而应在时间片用完后再降低。

44. A

当一个进程被唤醒时，这个进程就进入了就绪态，等待进程调度而占有 CPU 运行。进程被唤醒在某种情形下优先级可以增大，但一般不会变为最大，而由固定的算法来计算。也不会在唤醒后位于就绪队列的队首，就绪队列是按照一定的规则赋予其位置的，如先来先服务，或者高优先级优先，或者短进程优先等，更不能直接占有处理器运行。

45. D

进程创建原语完成的工作是：向系统申请一个空闲 PCB，为被创建进程分配必要的资源，然后将其 PCB 初始化，并将此 PCB 插入就绪队列，最后返回一个进程标志号。当调度程序为进程分配 CPU 后，进程开始运行。所以进程创建的过程中不会包含分配 CPU 的过程，这不是进程创建者的工作，而是调度程序的工作。

46. A

进程间的通信主要有管道、消息传递、共享内存、文件映射和套接字等。数据库不能用于进程间通信。

47. C

进程可以创建进程或线程，线程也可以创建线程，但线程不能创建进程。

48. A

进程申请读磁盘操作时，因为要等待 I/O 操作完成，会把自身阻塞，此时进程变为阻塞态；I/O 操作完成后，进程得到了想要的资源，会从阻塞态转换到就绪态（这是操作系统的 behavior）。而降低进程优先级、分配用户内存空间和增加进程的时间片大小都不一定会发生，选 A。

49. C

管道实际上是一种固定大小的缓冲区，管道对于管道两端的进程而言，就是一个文件，但它不是普通的文件，它不属于某种文件系统，而是自立门户、单独构成的一种文件系统，并且只存在于内存中。它类似于通信中半双工信道的进程通信机制，一个管道可以实现双向的数据传输，而同一时刻只能最多有一个方向的传输，不能两个方向同时进行。管道的容量大小通常为内存上的一页，它的大小并不受磁盘容量大小的限制。当管道满时，进程在写管道会被阻塞，而当管道空时，进程在读管道会被阻塞，因此选 C。

50. D

P(wait)操作表示进程请求某一资源，A、B 和 C 都因为请求某一资源会进入阻塞态，而 D 只是被剥夺了处理机资源，进入就绪态，一旦得到处理机即可运行。

51. C

进程等待某资源为可用（不包括处理机）或等待输入/输出完成均会进入阻塞态，因此 I、II 正确；III 中情况发生时，进程进入就绪态，因此 III 错误，答案选 C。

52. C

当被阻塞进程等待的某资源（不包括处理机）可用时，进程将会被唤醒。I/O 结束后，等待

该 I/O 结束而被阻塞的有关进程会被唤醒, I 正确; 某进程退出临界区后, 之前因需要进入该临界区而被阻塞的有关进程会被唤醒, II 正确; 当前进程的时间片用完后进入就绪队列等待重新调度, 优先级最高的进程获得处理机资源从就绪态变成执行态, III 错误。

53. B

应用程序没有进行线程管理的代码, 只有一个到内核级线程的编程接口, 内核为进程及其内部的每个线程维护上下文信息, 调度也是在内核中由操作系统完成的, A 正确。在多线程模型中, 用户级线程和内核级线程的连接方式分为多对一、一对一、多对多, “操作系统为每个用户线程建立一个线程控制块” 属于一对一模型, B 错误。用户级线程的切换可以在用户空间完成, 内核级线程的切换需要操作系统帮助进行调度, 因此用户级线程的切换效率更高, C 正确。用户级线程的管理工作可以只在用户空间中进行, 因此可以在不支持内核级线程的操作系统上实现, D 正确。

54. B

父进程与子进程当然可以并发执行, A 正确。父进程可与子进程共享一部分资源, 但不能共享虚拟地址空间, 在创建子进程时, 会为子进程分配资源, 如虚拟地址空间等, B 错误。临界资源一次只能为一个进程所用, D 正确。进程控制块 (PCB) 是进程存在的唯一标志, 每个进程都有自己的 PCB, C 正确。

二、综合应用题

1. 分析:

从进程的概念、进程与程序之间的关系来考虑问题的解答。进程是程序的执行过程, 进程代表执行中的程序, 因此进程与程序的差别就隐含在“执行”之中。程序是静态的指令集合, 进程是程序的动态执行过程。静态的程序除占用磁盘空间外, 不需要其他系统资源, 只有执行中的进程才需要分配内存、CPU 等系统资源。

进程的定义说明了两点:

- 1) 进程与程序相关, 进程包含了程序。程序是进程的核心内容, 没有程序就没有进程。
- 2) 进程不仅仅是程序, 还包含程序在执行过程中使用的全部资源。没有资源, 程序就无法执行, 因此进程是程序执行的载体。

运行一个程序时, 操作系统首先要创建一个进程, 为进程分配内存等资源, 然后加入进程队列中执行。对单个进程在某个时刻而言, 一个进程只能执行一个程序, 进程与程序之间是一对一的关系。但对整个系统中的进程集合及进程的生命周期而言, 进程与程序之间可以形成一对一、多对一、一对多、多对多的关系。

解答:

执行一条命令或运行一个应用程序时, 进程和程序之间形成一对一的关系。进程在执行过程中可以加载执行不同的应用程序, 从而形成一对多的关系; 以不同的参数或数据多次执行同一个应用程序时, 形成多对一的关系; 并发地执行不同的应用程序时, 形成多对多的关系。

2. 解答:

父进程创建子进程后, 父进程与子进程同时执行 (并发)。主程序调用子程序后, 主程序暂停在调用点, 子程序开始执行, 直到子程序返回, 主程序才开始执行。

3. 分析:

在操作系统中, 进程是竞争和分配计算机系统资源的基本单位。每个进程都有自己的独立地址空间。为了保证多个进程能够彼此互不干扰地共享物理内存, 操作系统利用硬件地址机制对进程的地址空间进行了严格的保护, 限制每个进程只能访问自己的地址空间。

解答：

每个进程都有自己独立的地址空间。在操作系统和硬件的地址保护机制下，进程无法访问其他进程的地址空间，所以必须借助于操作系统的系统调用函数实现进程之间的通信。进程通信的主要方式有：

- 1) 共享内存区。通过系统调用创建共享内存区。多个进程可以（通过系统调用）连接同一个共享内存区，通过访问共享内存区实现进程之间的数据交换。使用共享内存区时需要利用信号量解决同步互斥问题。
- 2) 消息传递。通过发送/接收消息，系统调用实现进程之间的通信。当进程发送消息时，系统将消息从用户缓冲区复制到内核中的消息缓冲区，然后将消息缓冲区挂入消息队列。进程发送的消息保持在消息队列中，直到被另一进程接收。当进程接收消息时，系统从消息队列中解挂消息缓冲区，将消息从内核的消息缓冲区中复制到用户缓冲区，然后释放消息缓冲区。
- 3) 管道系统。管道是先进先出（FIFO）的信息流，允许多个进程向管道写入数据，允许多个进程从管道读出数据。在读/写过程中，操作系统保证数据的写入顺序和读出顺序是一致的。进程通过读/写管道文件或管道设备实现彼此之间的通信。
- 4) 共享文件。利用操作系统提供的文件共享功能实现进程之间的通信。这时，也需要信号量来解决文件共享操作中的同步和互斥问题。

4. 解答：

多线程是指在一个程序中可以定义多个线程并同时运行它们，每个线程可以执行不同的任务。

多线程与多任务的区别：多任务是针对操作系统而言的，代表操作系统可以同时执行的程序个数；多线程是针对一个程序而言的，代表一个程序可以同时执行的线程个数，而每个线程可以完成不同的任务。

5. 解答：

- 1) 是。若系统中未运行进程，则系统很快会选择一个就绪进程运行。只有就绪队列中无进程时，CPU 才可能处于空闲状态。
- 2) 不一定。因为系统中的所有进程可能都处于等待态，可能处于死锁状态，也有可能因为等待的事件未发生而进入循环等待态。
- 3) 不一定。因为高优先级的进程有可能正处在等待队列中，进程调度会从就绪队列中选择一个进程占用 CPU，这个被选中的进程可能优先级较低。

6. 解答：

- 1) 为支持多进程的并发执行，系统为每个进程建立了一个数据结构：进程控制块（PCB），用于进程的管理和控制。PCB 中记录了有关进程的一些描述信息和控制信息，包括进程标识符、进程当前的状态、优先级、进程放弃 CPU 时的现场信息，以及指示组成进程的程序和数据在存储器中存放位置的信息、资源使用信息、进程各种队列的连接指针和反映进程之间的隶属关系的信息等。
- 2) 在进程的整个生命周期中，会经历多种状态。进程控制的主要职能是对系统中所有进程实施有效的管理，它具有创建新进程、撤销已有进程、实现进程的状态转换等功能。在操作系统内核中，有一组程序专门用于完成对进程的控制，这些原语至少需要包括创建新进程原语、阻塞进程原语、唤醒进程原语、终止进程原语等操作。系统服务对用户开放，即用户可以通过相应的接口来使用它们。

3) 进程创建原语：从 PCB 集合中申请一个空白的 PCB，将调用者参数（如进程外部标识符、初始 CPU 状态、进程优先数、初始内存及申请资源清单等）添入该 PCB，设置记账数据。置新进程为“就绪”态。

终止进程原语：用于终止完成的进程，回收其所占资源。包括消去其资源描述块，消去进程的 PCB。

阻塞原语：将进程从运行态变为阻塞态。进程被插入等待事件的队列，同时修改 PCB 中相应的表项，如进程状态和等待队列指针等。

唤醒原语：将进程从阻塞态变为就绪态。进程从阻塞队列中移出，插入就绪队列，等待调度，同时修改 PCB 中相应的表项，如进程状态等。

7. 分析：

根据题意，首先由图进行分析，进程由运行态可以直接回到就绪队列的末尾，而且就绪队列中是先来先服务。那么，什么情况才能发生这样的变化呢？只有采用单一时间片轮转的调度系统，当分配的时间片用完时，才会发生上述情况。因此，该系统一定采用时间片轮转调度算法，采用时间片轮转算法的操作系统一般均为交互式操作系统。由图可知，进程被阻塞时，可以进入不同的阻塞队列，等待打印机输出结果和等待磁盘读取文件。所以，它是一个多阻塞队列的时间片轮转法的调度系统。

解答：

1) 根据题意，该系统采用的是时间片轮转法调度进程策略。

2) 可能的变化见下表。

变 化	原 因
1	进程被调度，获得 CPU，进入运行态
2	进程需要读文件，因 I/O 操作进入阻塞态
3	进程打印输出结果，因打印机未结束而阻塞
4	打印机打印结束，进程重新回归就绪态，并排在尾部
5	进程所需数据已从磁盘进入内存，进程回到就绪态
6	运行的进程因为时间片用完而让出 CPU，排到就绪队列尾部

2.2 处理机调度

在学习本节时，请读者思考以下问题：

- 为什么要进行处理机调度？
- 调度算法有哪几种？结合第 1 章学习的分时操作系统和实时操作系统，思考哪种调度算法比较适合这两种操作系统。

希望读者能够在学习调度算法前，先自己思考一些调度算法，在学习的过程中注意把自己的想法与这些经典的算法进行比对，并学会计算一些调度算法的周转时间。

2.2.1 调度的概念

1. 调度的基本概念

在多道程序系统中，进程的数量往往多于处理机的个数，因此进程争用处理机的情况在所难免。

免。处理机调度是对处理机进行分配，即从就绪队列中按照一定的算法（公平、高效）选择一个进程并将处理机分配给它运行，以实现进程并发地执行。

处理机调度是多道程序操作系统的基础，是操作系统设计的核心问题。

2. 调度的层次

一个作业从提交开始直到完成，往往要经历以下三级调度，如图 2.6 所示。

1) 作业调度。又称高级调度，其主要任务是按一定的原则从外存上处于后备状态的作业中挑选一个（或多个）作业，给它（们）分配内存、输入/输出设备等必要的资源，并建立相应的进程，以使它（们）获得竞争处理机的权利。简言之，作业调度就是内存与辅存之间的调度。对于每个作业只调入一次、调出一次。

多道批处理系统中大多配有作业调度，而其他系统中通常不需要配置作业调度。作业调度的执行频率较低，通常为几分钟一次。

2) 中级调度。又称内存调度，其作用是提高内存利用率和系统吞吐量。为此，应将那些暂时不能运行的进程调至外存等待，把此时的进程状态称为挂起态。当它们已具备运行条件且内存又稍有空闲时，由中级调度来决定把外存上的那些已具备运行条件的就绪进程，再重新调入内存，并修改其状态为就绪态，挂在就绪队列上等待。

3) 进程调度。又称低级调度，其主要任务是按照某种方法和策略从就绪队列中选取一个进程，将处理机分配给它。进程调度是操作系统中最基本的一种调度，在一般的操作系统中都必须配置进程调度。进程调度的频率很高，一般几十毫秒一次。

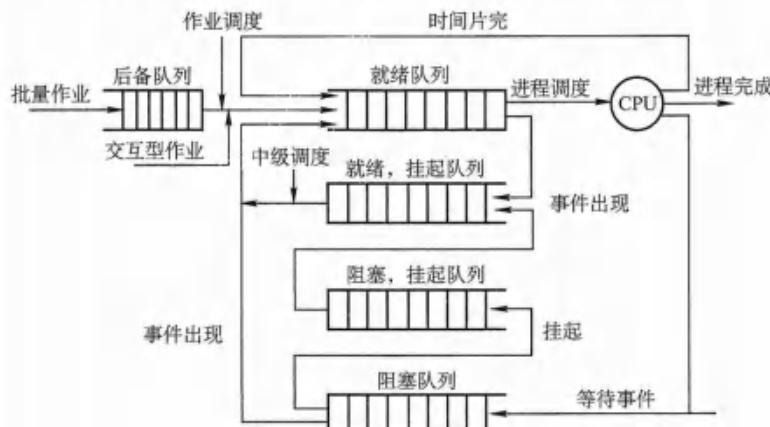


图 2.6 处理机的三级调度

3. 三级调度的联系

作业调度从外存的后备队列中选择一批作业进入内存，为它们建立进程，这些进程被送入就绪队列，进程调度从就绪队列中选出一个进程，并把其状态改为运行态，把 CPU 分配给它。中级调度是为了提高内存的利用率，系统将那些暂时不能运行的进程挂起来。当内存空间宽松时，通过中级调度选择具备运行条件的进程，将其唤醒。

- 1) 作业调度为进程活动做准备，进程调度使进程正常活动起来，中级调度将暂时不能运行的进程挂起，中级调度处于作业调度和进程调度之间。
- 2) 作业调度次数少，中级调度次数略多，进程调度频率最高。
- 3) 进程调度是最基本的，不可或缺。

2.2.2 调度的时机、切换与过程

进程调度和切换程序是操作系统内核程序。请求调度的事件发生后，才可能运行进程调度程序，调度了新的就绪进程后，才会进行进程间的切换。理论上这三件事情应该顺序执行，但在实际设计中，操作系统内核程序运行时，若某时发生了引起进程调度的因素，则不一定能够马上进行调度与切换。

现代操作系统中，不能进行进程的调度与切换的情况有以下几种：

- 1) 在处理中断的过程中。中断处理过程复杂，在实现上很难做到进程切换，而且中断处理是系统工作的一部分，逻辑上不属于某一进程，不应被剥夺处理机资源。
- 2) 进程在操作系统内核程序临界区中。进入临界区后，需要独占式地访问共享数据，理论上必须加锁，以防止其他并行程序进入，在解锁前不应切换到其他进程运行，以加快该共享数据的释放。
- 3) 其他需要完全屏蔽中断的原子操作过程中。如加锁、解锁、中断现场保护、恢复等原子操作。在原子过程中，连中断都要屏蔽，更不应该进行进程调度与切换。

若在上述过程中发生了引起调度的条件，则不能马上进行调度和切换，应置系统的请求调度标志，直到上述过程结束后才进行相应的调度与切换。

应该进行进程调度与切换的情况如下：

- 1) 发生引起调度条件且当前进程无法继续运行下去时，可以马上进行调度与切换。若操作系统只在这种情况下进行进程调度，则是非剥夺调度。
- 2) 中断处理结束或自陷处理结束后，返回被中断进程的用户态程序执行现场前，若置上请求调度标志，即可马上进行进程调度与切换。若操作系统支持这种情况下的运行调度程序，则实现了剥夺方式的调度。

进程切换往往在调度完成后立刻发生，它要求保存原进程当前切换点的现场信息，恢复被调度进程的现场信息。现场切换时，操作系统内核将原进程的现场信息推入当前进程的内核堆栈来保存它们，并更新堆栈指针。内核完成从新进程的内核栈中装入新进程的现场信息、更新当前运行进程空间指针、重设 PC 寄存器等相关工作之后，开始运行新的进程。

2.2.3 进程调度方式

所谓进程调度方式，是指当某个进程正在处理机上执行时，若有某个更为重要或紧迫的进程需要处理，即有优先权更高的进程进入就绪队列，此时应如何分配处理机。

通常有以下两种进程调度方式：

- 1) 非剥夺调度方式，又称非抢占方式。非剥夺调度方式是指当一个进程正在处理机上执行时，即使有某个更为重要或紧迫的进程进入就绪队列，仍然让正在执行的进程继续执行，直到该进程完成或发生某种事件而进入阻塞态时，才把处理机分配给更为重要或紧迫的进程。

在非剥夺调度方式下，一旦把 CPU 分配给一个进程，该进程就会保持 CPU 直到终止或转换到等待态。这种方式的优点是实现简单、系统开销小，适用于大多数的批处理系统，但它不能用于分时系统和大多数的实时系统。

- 2) 剥夺调度方式，又称抢占方式。剥夺调度方式是指当一个进程正在处理机上执行时，若有某个更为重要或紧迫的进程需要使用处理机，则立即暂停正在执行的进程，将处理机分配给这个更为重要或紧迫的进程。

采用剥夺式的调度，对提高系统吞吐率和响应效率都有明显的好处。但“剥夺”不是一种任意性行为，必须遵循一定的原则，主要有优先权、短进程优先和时间片原则等。

2.2.4 调度的基本准则

不同的调度算法具有不同的特性，在选择调度算法时，必须考虑算法的特性。为了比较处理机调度算法的性能，人们提出了很多评价准则，下面介绍其中主要的几种：

- 1) CPU 利用率。CPU 是计算机系统中最重要和昂贵的资源之一，所以应尽可能使 CPU 保持“忙”状态，使这一资源利用率最高。
- 2) 系统吞吐量。表示单位时间内 CPU 完成作业的数量。长作业需要消耗较长的处理机时间，因此会降低系统的吞吐量。而对于短作业，它们所需要消耗的处理机时间较短，因此能提高系统的吞吐量。调度算法和方式的不同，也会对系统的吞吐量产生较大的影响。
- 3) 周转时间。周转时间是指从作业提交到作业完成所经历的时间，是作业等待、在就绪队列中排队、在处理机上运行及进行输入/输出操作所花费时间的总和。

作业的周转时间可用公式表示如下：

$$\text{周转时间} = \text{作业完成时间} - \text{作业提交时间}$$

平均周转时间是指多个作业周转时间的平均值：

$$\text{平均周转时间} = (\text{作业 1 的周转时间} + \dots + \text{作业 } n \text{ 的周转时间})/n$$

带权周转时间是指作业周转时间与作业实际运行时间的比值：

$$\text{带权周转时间} = \frac{\text{作业周转时间}}{\text{作业实际运行时间}}$$

平均带权周转时间是指多个作业带权周转时间的平均值：

$$\text{平均带权周转时间} = (\text{作业 1 的带权周转时间} + \dots + \text{作业 } n \text{ 的带权周转时间})/n$$

- 4) 等待时间。等待时间指进程处于等处理机状态的时间之和，等待时间越长，用户满意度越低。处理机调度算法实际上并不影响作业执行或输入/输出操作的时间，只影响作业在就绪队列中等待所花的时间。因此，衡量一个调度算法的优劣，常常只需简单地考察等待时间。
- 5) 响应时间。响应时间指从用户提交请求到系统首次产生响应所用的时间。在交互式系统中，周转时间不可能是最好的评价准则，一般采用响应时间作为衡量调度算法的重要准则之一。从用户角度来看，调度策略应尽量降低响应时间，使响应时间处在用户能接受的范围之内。

要想得到一个满足所有用户和系统要求的算法几乎是不可能的。设计调度程序，一方面要满足特定系统用户的要求（如某些实时和交互进程的快速响应要求），另一方面要考虑系统整体效率（如减少整个系统的进程平均周转时间），同时还要考虑调度算法的开销。

2.2.5 典型的调度算法

操作系统中存在多种调度算法，有的调度算法适用于作业调度，有的调度算法适用于进程调度，有的调度算法两者都适用。下面介绍几种常用的调度算法。

1. 先来先服务（FCFS）调度算法

FCFS 调度算法是一种最简单的调度算法，它既可用于作业调度，又可用于进程调度。在作业调度中，算法每次从后备作业队列中选择最先进入该队列的一个或几个作业，将它们调入内存，分配必要的资源，创建进程并放入就绪队列。

在进程调度中，FCFS 调度算法每次从就绪队列中选择最先进入该队列的进程，将处理机分配给它，使之投入运行，直到完成或因某种原因而阻塞时才释放处理机。

下面通过一个实例来说明 FCFS 调度算法的性能。假设系统中有 4 个作业，它们的提交时间分别是 8, 8.4, 8.8, 9，运行时间依次是 2, 1, 0.5, 0.2，系统采用 FCFS 调度算法，这组作业的平均等待时间、平均周转时间和平均带权周转时间见表 2.2。

表 2.2 FCFS 调度算法的性能

作业号	提交时间	运行时间	开始时间	等待时间	完成时间	周转时间	带权周转时间
1	8	2	8	0	10	2	1
2	8.4	1	10	1.6	11	2.6	2.6
3	8.8	0.5	11	2.2	11.5	2.7	5.4
4	9	0.2	11.5	2.5	11.7	2.7	13.5

平均等待时间 $t = (0 + 1.6 + 2.2 + 2.5)/4 = 1.575$ ； 平均周转时间 $T = (2 + 2.6 + 2.7 + 2.7)/4 = 2.5$ ；

平均带权周转时间 $W = (1 + 2.6 + 5.4 + 13.5)/4 = 5.625$ 。

FCFS 调度算法属于不可剥夺算法。从表面上看，它对所有作业都是公平的，但若一个长作业先到达系统，就会使后面的许多短作业等待很长时间，因此它不能作为分时系统和实时系统的主要调度策略。但它常被结合在其他调度策略中使用。例如，在使用优先级作为调度策略的系统中，往往对多个具有相同优先级的进程按 FCFS 原则处理。

FCFS 调度算法的特点是算法简单，但效率低；对长作业比较有利，但对短作业不利（相对 SJF 和高响应比）；有利于 CPU 繁忙型作业，而不利于 I/O 繁忙型作业。

2. 短作业优先 (SJF) 调度算法

短作业（进程）优先调度算法是指对短作业（进程）优先调度的算法。短作业优先 (SJF) 调度算法从后备队列中选择一个或若干估计运行时间最短的作业，将它们调入内存运行；短进程优先 (SPF) 调度算法从就绪队列中选择一个估计运行时间最短的进程，将处理机分配给它，使之立即执行，直到完成或发生某事件而阻塞时，才释放处理机。

例如，考虑表 2.2 中给出的一组作业，若系统采用短作业优先调度算法，其平均等待时间、平均周转时间和平均带权周转时间见表 2.3。

表 2.3 SJF 调度算法的性能

作业号	提交时间	运行时间	开始时间	等待时间	完成时间	周转时间	带权周转时间
1	8	2	8	0	10	2	1
2	8.4	1	10.7	2.3	11.7	3.3	3.3
3	8.8	0.5	10.2	1.4	10.7	1.9	3.8
4	9	0.2	10	1	10.2	1.2	6

平均等待时间 $t = (0 + 2.3 + 1.4 + 1)/4 = 1.175$ ； 平均周转时间 $T = (2 + 3.3 + 1.9 + 1.2)/4 = 2.1$ ；

平均带权周转时间 $W = (1 + 3.3 + 3.8 + 6)/4 = 3.525$ 。

SJF 调度算法也存在不容忽视的缺点：

- 1) 该算法对长作业不利，由表 2.2 和表 2.3 可知，SJF 调度算法中长作业的周转时间会增加。更严重的是，若有一长作业进入系统的后备队列，由于调度程序总是优先调度那些（即使是后进来的）短作业，将导致长作业长期不被调度（“饥饿”现象，注意区分“死锁”，后者是系统环形等待，前者是调度策略问题）。

- 2) 该算法完全未考虑作业的紧迫程度，因而不能保证紧迫性作业会被及时处理。
 - 3) 由于作业的长短只是根据用户所提供的估计执行时间而定的，而用户又可能会有意或无意地缩短其作业的估计运行时间，致使该算法不一定能真正做到短作业优先调度。
- 注意，SJF 调度算法的平均等待时间、平均周转时间最少。

3. 优先级调度算法

优先级调度算法又称优先权调度算法，它既可用于作业调度，又可用于进程调度。该算法中的优先级用于描述作业运行的紧迫程度。

在作业调度中，优先级调度算法每次从后备作业队列中选择优先级最高的一个或几个作业，将它们调入内存，分配必要的资源，创建进程并放入就绪队列。在进程调度中，优先级调度算法每次从就绪队列中选择优先级最高的进程，将处理机分配给它，使之投入运行。

根据新的更高优先级进程能否抢占正在执行的进程，可将该调度算法分为如下两种：

- 1) 非剥夺式优先级调度算法。当一个进程正在处理机上运行时，即使有某个更为重要或紧迫的进程进入就绪队列，仍然让正在运行的进程继续运行，直到由于其自身的原因而主动让出处理机时（任务完成或等待事件），才把处理机分配给更为重要或紧迫的进程。
 - 2) 剥夺式优先级调度算法。当一个进程正在处理机上运行时，若有某个更为重要或紧迫的进程进入就绪队列，则立即暂停正在运行的进程，将处理机分配给更重要或紧迫的进程。而根据进程创建后其优先级是否可以改变，可以将进程优先级分为以下两种：
- 1) 静态优先级。优先级是在创建进程时确定的，且在进程的整个运行期间保持不变。确定静态优先级的主要依据有进程类型、进程对资源的要求、用户要求。
 - 2) 动态优先级。在进程运行过程中，根据进程情况的变化动态调整优先级。动态调整优先级的主要依据有进程占有 CPU 时间的长短、就绪进程等待 CPU 时间的长短。

一般来说，进程优先级的设置可以参照以下原则：

- 1) 系统进程 > 用户进程。系统进程作为系统的管理者，理应拥有更高的优先级。
- 2) 交互型进程 > 非交互型进程（或前台进程 > 后台进程）。大家平时在使用手机时，在前台运行的正在和你交互的进程应该更快速地响应你，因此自然需要被优先处理，即要有更高的优先级。
- 3) I/O 型进程 > 计算型进程。所谓 I/O 型进程，是指那些会频繁使用 I/O 设备的进程，而计算型进程是那些频繁使用 CPU 的进程（很少使用 I/O 设备）。我们知道，I/O 设备（如打印机）的处理速度要比 CPU 慢得多，因此若将 I/O 型进程的优先级设置得更高，就更有可能让 I/O 设备尽早开始工作，进而提升系统的整体效率。

4. 高响应比优先调度算法

高响应比优先调度算法主要用于作业调度，是对 FCFS 调度算法和 SJF 调度算法的一种综合平衡，同时考虑了每个作业的等待时间和估计的运行时间。在每次进行作业调度时，先计算后备作业队列中每个作业的响应比，从中选出响应比最高的作业投入运行。

响应比的变化规律可描述为

$$\text{响应比 } R_p = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}}$$

根据公式可知：

- 1) 作业的等待时间相同时，要求服务时间越短，响应比越高，有利于短作业。
- 2) 要求服务时间相同时，作业的响应比由其等待时间决定，等待时间越长，其响应比越高，

因而它实现的是先来先服务。

- 3) 对于长作业，作业的响应比可以随等待时间的增加而提高，等待时间足够长时，其响应比便可升到很高，从而也可获得处理机。因此，克服了饥饿状态，兼顾了长作业。

5. 时间片轮转调度算法

时间片轮转调度算法主要适用于分时系统。在这种算法中，系统将所有就绪进程按到达时间的先后次序排成一个队列，进程调度程序总是选择就绪队列中的第一个进程执行，即先来先服务的原则，但仅能运行一个时间片，如 100ms。在使用完一个时间片后，即使进程并未完成其运行，它也必须释放出（被剥夺）处理机给下一个就绪的进程，而被剥夺的进程返回到就绪队列的末尾重新排队，等候再次运行。

在时间片轮转调度算法中，时间片的大小对系统性能的影响很大。若时间片足够大，以至于所有进程都能在一个时间片内执行完毕，则时间片轮转调度算法就退化为先来先服务调度算法。若时间片很小，则处理机将在进程间过于频繁地切换，使处理机的开销增大，而真正用于运行用户进程的时间将减少。因此，时间片的大小应选择适当。

时间片的长短通常由以下因素确定：系统的响应时间、就绪队列中的进程数目和系统的处理能力。

6. 多级反馈队列调度算法（融合了前几种算法的优点）

多级反馈队列调度算法是时间片轮转调度算法和优先级调度算法的综合与发展，如图 2.7 所示。通过动态调整进程优先级和时间片大小，多级反馈队列调度算法可以兼顾多方面的系统目标。例如，为提高系统吞吐量和缩短平均周转时间而照顾短进程；为获得较好的 I/O 设备利用率和缩短响应时间而照顾 I/O 型进程；同时，也不必事先估计进程的执行时间。

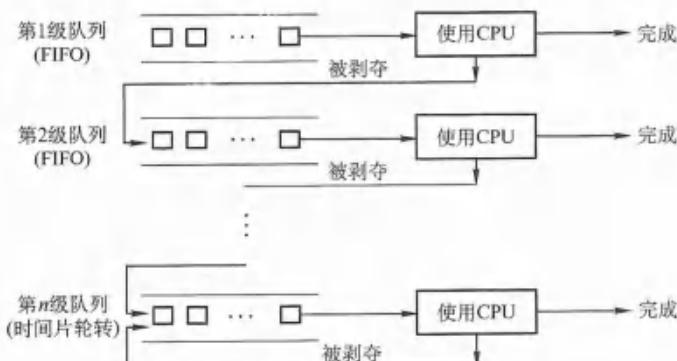


图 2.7 多级反馈队列调度算法

多级反馈队列调度算法的实现思想如下：

- 1) 设置多个就绪队列，并为各个队列赋予不同的优先级，第 1 级队列的优先级最高，第 2 级队列次之，其余队列的优先级逐次降低。
- 2) 赋予各个队列中进程执行时间片的大小各不相同。在优先级越高的队列中，每个进程的运行时间片越小。例如，第 2 级队列的时间片要比第 1 级队列的时间片长 1 倍……第 $i + 1$ 级队列的时间片要比第 i 级队列的时间片长 1 倍。
- 3) 一个新进程进入内存后，首先将它放入第 1 级队列的末尾，按 FCFS 原则排队等待调度。当轮到该进程执行时，如它能在该时间片内完成，便可准备撤离系统；若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 2 级队列的末尾，再同样按 FCFS 原则等

待调度执行；若它在第 2 级队列中运行一个时间片后仍未完成，再以同样的方法放入第 3 级队列……如此下去，当一个长进程从第 1 级队列依次降到第 n 级队列后，在第 n 级队列中便采用时间片轮转的方式运行。

- 4) 仅当第 1 级队列为空时，调度程序才调度第 2 级队列中的进程运行；仅当第 $1 \sim (i-1)$ 级队列均为空时，才会调度第 i 级队列中的进程运行。若处理机正在执行第 i 级队列中的某进程，这时又有新进程进入优先级较高的队列 [第 $1 \sim (i-1)$ 中的任何一个队列]，则此时新进程将抢占正在运行进程的处理机，即由调度程序把正在运行的进程放回第 i 级队列的末尾，把处理机分配给新到的更高优先级的进程。

多级反馈队列的优势有以下几点：

- 1) 终端型作业用户：短作业优先。
- 2) 短批处理作业用户：周转时间较短。
- 3) 长批处理作业用户：经过前面几个队列得到部分执行，不会长期得不到处理。

2.2.6 本节小结

本节开头提出的问题的参考答案如下。

- 1) 为什么要进行处理机调度？

若没有处理机调度，同意味着要等到当前运行的进程执行完毕后，下一个进程才能执行，而实际情况中，进程时常需要等待一些外部设备的输入，而外部设备的速度与处理机相比是非常缓慢的，若让处理机总是等待外部设备，则对处理机的资源是极大的浪费。而引进处理机调度后，可在运行进程等待外部设备时，把处理机调度给其他进程，从而提高处理机的利用率。用一句话简单的话说，就是为了合理地处理计算机的软/硬件资源。

- 2) 调度算法有哪几种？结合第 1 章学习的分时操作系统和实时操作系统，思考有没有哪种调度算法比较适合这两种操作系统。

本节介绍的调度算法有先来先服务调度算法、短作业优先调度算法、优先级调度算法、高响应比优先调度算法、时间片轮转调度算法、多级反馈队列调度算法 6 种。

先来先服务算法和短作业优先算法无法保证及时地接收和处理问题，因此无法保证在规定的时间间隔内响应每个用户的需求，也同样无法达到实时操作系统的及时性需求。优先级调度算法按照任务的优先级进行调度，对于更紧急的任务给予更高的优先级，适合实时操作系统。

高响应比优先调度算法、时间片轮转调度算法、多级反馈队列调度算法都能保证每个任务在一定时间内分配到时间片，并轮流占用 CPU，适合分时操作系统。

本节主要介绍了处理机调度的概念。操作系统主要管理处理机、内存、文件、设备几种资源，只要对资源的请求大于资源本身的数量，就会涉及调度。例如，在单处理机系统中，处理机只有一个，而请求服务的进程却有多个，所以就有处理机调度的概念出现。而出现调度的概念后，又有了一个问题，即如何调度、应该满足谁、应该让谁等待，这是调度算法所回答的问题；而应该满足谁、应该让谁等待，要遵循一定的准则，即调度的准则。调度这一概念贯穿于操作系统的始终，读者在接下来的学习中，将接触到几种资源的调度问题和相应的调度算法。将它们与处理机调度的内容相对比，将会发现它们有异曲同工之妙。

2.2.7 本节习题精选

一、单项选择题

1. 时间片轮转调度算法是为了（ ）。

- A. 多个用户能及时干预系统 B. 使系统变得高效
 C. 优先级较高的进程得到及时响应 D. 需要 CPU 时间最少的进程最先做
2. 在单处理器的多进程系统中，进程什么时候占用处理器及决定占用时间的长短是由（ ）决定的。
 A. 进程相应的代码长度 B. 进程总共需要运行的时间
 C. 进程特点和进程调度策略 D. 进程完成什么功能
3. （ ）有利于 CPU 繁忙型的作业，而不利于 I/O 繁忙型的作业。
 A. 时间片轮转调度算法 B. 先来先服务调度算法
 C. 短作业（进程）优先算法 D. 优先权调度算法
4. 下面有关选择进程调度算法的准则中，不正确的是（ ）。
 A. 尽快响应交互式用户的请求 B. 尽量提高处理器利用率
 C. 尽可能提高系统吞吐量 D. 适当增长进程就绪队列的等待时间
5. 设有 4 个作业同时到达，每个作业的执行时间为 2h，它们在一台处理器上按单道式运行，则平均周转时间为（ ）。
 A. 1h B. 5h C. 2.5h D. 8h
6. 若每个作业只能建立一个进程，为了照顾短作业用户，应采用（ ）；为了照顾紧急作业用户，应采用（ ）；为了能实现人机交互，应采用（ ）；而能使短作业、长作业和交互作业用户都满意，应采用（ ）。
 A. FCFS 调度算法 B. 短作业优先调度算法
 C. 时间片轮转调度算法 D. 多级反馈队列调度算法
 E. 剥夺式优先级调度算法
7. （ ）优先级是在创建进程时确定的，确定之后在整个运行期间不再改变。
 A. 先来先服务 B. 动态 C. 短作业 D. 静态
8. 现在有三个同时到达的作业 J₁, J₂ 和 J₃，它们的执行时间分别是 T₁, T₂, T₃，且 T₁ < T₂ < T₃。系统按单道方式运行且采用短作业优先调度算法，则平均周转时间是（ ）。
 A. T₁ + T₂ + T₃ B. (3T₁ + 2T₂ + T₃) / 3
 C. (T₁ + T₂ + T₃) / 3 D. (T₁ + 2T₂ + 3T₃) / 3
9. 设有三个作业，其运行时间分别是 2h, 5h, 3h，假定它们同时到达，并在同一台处理器上以单道方式运行，则平均周转时间最小的执行顺序是（ ）。
 A. J₁, J₂, J₃ B. J₃, J₂, J₁ C. J₂, J₁, J₃ D. J₁, J₃, J₂
10. 【2013 统考真题】某系统正在执行三个进程 P₁, P₂ 和 P₃，各进程的计算（CPU）时间和 I/O 时间比例如下表所示。

进程	计算时间	I/O 时间
P ₁	90%	10%
P ₂	50%	50%
P ₃	15%	85%

- 为提高系统资源利用率，合理的进程优先级设置应为（ ）。
- A. P₁ > P₂ > P₃ B. P₃ > P₂ > P₁ C. P₂ > P₁ = P₃ D. P₁ > P₂ = P₃
11. 采用时间片轮转调度算法分配 CPU 时，当处于运行态的进程用完一个时间片后，它的状态是（ ）状态。

- A. 阻塞 B. 运行 C. 就绪 D. 消亡
12. 一个作业 8:00 到达系统，估计运行时间为 1h。若 10:00 开始执行该作业，其响应比是（ ）。
- A. 2 B. 1 C. 3 D. 0.5
13. 关于优先权大小的论述中，正确的是（ ）。
- A. 计算型作业的优先权，应高于 I/O 型作业的优先权
 B. 用户进程的优先权，应高于系统进程的优先权
 C. 在动态优先权中，随着作业等待时间的增加，其优先权将随之下降
 D. 在动态优先权中，随着进程执行时间的增加，其优先权降低
14. 下列调度算法中，（ ）调度算法是绝对可抢占的。
- A. 先来先服务 B. 时间片轮转 C. 优先级 D. 短进程优先
15. 作业是用户提交的，进程是由系统自动生成的，除此之外，两者的区别是（ ）。
- A. 两者执行不同的程序段
 B. 前者以用户任务为单位，后者以操作系统控制为单位
 C. 前者是批处理的，后者是分时的
 D. 后者是可并发执行，前者则不同
16. 【2009 统考真题】下列进程调度算法中，综合考虑进程等待时间和执行时间的是（ ）。
- A. 时间片轮转调度算法 B. 短进程优先调度算法
 C. 先来先服务调度算法 D. 高响应比优先调度算法
17. 进程调度算法采用固定时间片轮转调度算法，当时间片过大时，就会使时间片轮转法算法转化为（ ）调度算法。
- A. 高响应比优先 B. 先来先服务
 C. 短进程优先 D. 以上选项都不对
18. 有以下的进程需要调度执行（见下表）：
- 1) 若用非抢占式短进程优先调度算法，问这 5 个进程的平均周转时间是多少？
 - 2) 若采用抢占式短进程优先调度算法，问这 5 个进程的平均周转时间是多少？
- | 进程名 | 到达时间 | 运行时间 |
|----------------|------|------|
| P ₁ | 0.0 | 9 |
| P ₂ | 0.4 | 4 |
| P ₃ | 1.0 | 1 |
| P ₄ | 5.5 | 4 |
| P ₅ | 7 | 2 |
19. 有 5 个批处理作业 A, B, C, D, E 几乎同时到达，其预计运行时间分别为 10, 6, 2, 4, 8，其优先级（由外部设定）分别为 3, 5, 2, 1, 4，这里 5 为最高优先级。以下各种调度算法中，平均周转时间为 14 的是（ ）调度算法。
- A. 时间片轮转（时间片为 1） B. 优先级调度
 C. 先来先服务（按照顺序 10, 6, 2, 4, 8） D. 短作业优先
20. 【2017 统考真题】假设 4 个作业到达系统的时刻和运行时间如下表所示。

作业	到达时刻 t	运行时间
J ₁	0	3
J ₂	1	3
J ₃	1	2
J ₄	3	1

- 系统在 $t = 2$ 时开始作业调度。若分别采用先来先服务和短作业优先调度算法，则选中的作业分别是（）。
- A. J_2, J_3 B. J_1, J_4 C. J_2, J_4 D. J_1, J_3
21. 【2012 统考真题】一个多道批处理系统中仅有 P_1 和 P_2 两个作业， P_2 比 P_1 晚 5ms 到达，它的计算和 I/O 操作顺序如下：
 P_1 : 计算 60ms, I/O 80ms, 计算 20ms
 P_2 : 计算 120ms, I/O 40ms, 计算 40ms
若不考虑调度和切换时间，则完成两个作业需要的时间最少是（）。
- A. 240ms B. 260ms C. 340ms D. 360ms
22. 【2016 统考真题】某单 CPU 系统中有输入和输出设备各 1 台，现有 3 个并发执行的作业，每个作业的输入、计算和输出时间均分别为 2ms, 3ms 和 4ms，且都按输入、计算和输出的顺序执行，则执行完 3 个作业需要的时间最少是（）。
- A. 15ms B. 17ms C. 22ms D. 27ms
23. 【2017 统考真题】下列有关基于时间片的进程调度的叙述中，错误的是（）。
- A. 时间片越短，进程切换的次数越多，系统开销越大
B. 当前进程的时间片用完后，该进程状态由执行态变为阻塞态
C. 时钟中断发生后，系统会修改当前进程在时间片内的剩余时间
D. 影响时间片大小的主要因素包括响应时间、系统开销和进程数量等
24. 分时操作系统通常采用（）调度算法来为用户服务。
- A. 时间片轮转 B. 先来先服务 C. 短作业优先 D. 优先级
25. 在进程调度算法中，对短进程不利的是（）。
- A. 短进程优先调度算法 B. 先来先服务调度算法
C. 高响应比优先调度算法 D. 多级反馈队列调度算法
26. 假设系统中所有进程同时到达，则使进程平均周转时间最短的是（）调度算法。
- A. 先来先服务 B. 短进程优先 C. 时间片轮转 D. 优先级
27. 下列说法中，正确的是（）。
- I. 分时系统的时间片固定，因此用户数越多，响应时间越长
II. UNIX 是一个强大的多用户、多任务操作系统，支持多种处理器架构，按照操作系统分类，属于分时操作系统
III. 中断向量地址是中断服务例行程序的入口地址
IV. 中断发生时，由硬件保护并更新程序计数器 (PC)，而不是由软件完成，主要是为了提高处理速度
- A. I、II B. II、III C. III、IV D. 仅 IV
28. 【2012 统考真题】若某单处理器多进程系统中有多个就绪态进程，则下列关于处理机调度的叙述中，错误的是（）。
- A. 在进程结束时能进行处理机调度
B. 创建新进程后能进行处理机调度
C. 在进程处于临界区时不能进行处理机调度
D. 在系统调用完成并返回用户态时能进行处理机调度
29. 【2011 统考真题】下列选项中，满足短作业优先且不会发生饥饿现象的是（）调度算法。
- A. 先来先服务 B. 高响应比优先

- C. 时间片轮转 D. 非抢占式短作业优先
30. 【2014 统考真题】下列调度算法中，不可能导致饥饿现象的是（ ）。
- A. 时间片轮转 B. 静态优先数调度
C. 非抢占式短任务优先 D. 抢占式短任务优先
31. 【2018 统考真题】某系统采用基于优先权的非抢占式进程调度策略，完成一次进程调度和进程切换的系统时间开销为 $1\mu s$ 。在 T 时刻就绪队列中有 3 个进程 P_1 、 P_2 和 P_3 ，其在就绪队列中的等待时间、需要的 CPU 时间和优先权如下表所示。

进程	等待时间	需要的 CPU 时间	优先权
P_1	$30\mu s$	$12\mu s$	10
P_2	$15\mu s$	$24\mu s$	30
P_3	$18\mu s$	$36\mu s$	20

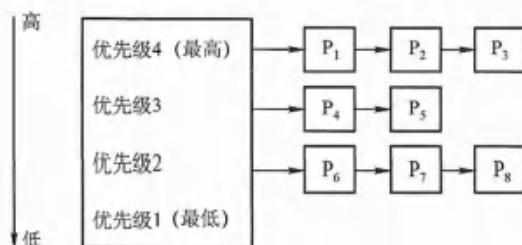
若优先权值大的进程优先获得 CPU，从 T 时刻起系统开始进程调度，则系统的平均周转时间为（ ）。

- A. $54\mu s$ C. $74\mu s$ D. $75\mu s$
32. 【2019 统考真题】系统采用二级反馈队列调度算法进行进程调度。就绪队列 Q_1 采用时间片轮转调度算法，时间片为 $10ms$ ；就绪队列 Q_2 采用短进程优先调度算法；系统优先调度 Q_1 队列中的进程，当 Q_1 为空时系统才会调度 Q_2 中的进程；新创建的进程首先进入 Q_1 ； Q_1 中的进程执行一个时间片后，若未结束，则转入 Q_2 。若当前 Q_1 、 Q_2 为空，系统依次创建进程 P_1 、 P_2 后即开始进程调度， P_1 、 P_2 需要的 CPU 时间分别为 $30ms$ 和 $20ms$ ，则进程 P_1 、 P_2 在系统中的平均等待时间为（ ）。
- A. $25ms$ C. $15ms$ D. $10ms$

33. 【2020 统考真题】下列与进程调度有关的因素中，在设计多级反馈队列调度算法时需要考虑的是（ ）。
- I. 就绪队列的数量 II. 就绪队列的优先级
III. 各就绪队列的调度算法 IV. 进程在就绪队列间的迁移条件
- A. 仅 I、II C. 仅 II、III、IV D. I、II、III 和 IV

二、综合应用题

- 为什么说多级反馈队列调度算法能较好地满足各类用户的需要？
- 将一组进程分为 4 类，如下图所示。各类进程之间采用优先级调度算法，而各类进程的内部采用时间片轮转调度算法。请简述 P_1 、 P_2 、 P_3 、 P_4 、 P_5 、 P_6 、 P_7 、 P_8 进程的调度过程。



- 设某计算机系统有一个 CPU、一台输入设备、一台打印机。现有两个进程同时进入就绪态，且进程 A 先得到 CPU 运行，进程 B 后运行。进程 A 的运行轨迹为：计算 $50ms$ ，打印信息 $100ms$ ，再计算 $50ms$ ，打印信息 $100ms$ ，结束。进程 B 的运行轨迹为：计算 $50ms$ ，

输入数据 80ms，再计算 100ms，结束。试画出它们的甘特图，并说明：

- 1) 开始运行后，CPU 有无空闲等待？若有，在哪段时间内等待？计算 CPU 的利用率。
- 2) 进程 A 运行时有无等待现象？若有，在什么时候发生等待现象？
- 3) 进程 B 运行时有无等待现象？若有，在什么时候发生等待现象？
4. 有一个 CPU 和两台外设 D_1, D_2 ，且在能够实现抢占式优先级调度算法的多道程序环境中，同时进入优先级由高到低的 P_1, P_2, P_3 三个作业，每个作业的处理顺序和使用资源的时间如下：

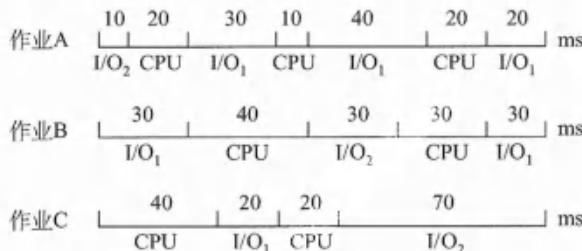
P_1 : D_2 (30ms), CPU (10ms), D_1 (30ms), CPU (10ms)

P_2 : D_1 (20ms), CPU (20ms), D_2 (40ms)

P_3 : CPU (30ms), D_1 (20ms)

假设忽略不计其他辅助操作的时间，每个作业的周转时间 T_1, T_2, T_3 分别为多少？CPU 和 D_1 的利用率各是多少？

5. 有三个作业 A, B, C，它们分别单独运行时的 CPU 和 I/O 占用时间如下图所示。



现在请考虑三个作业同时开始执行。系统中的资源有一个 CPU 和两台输入/输出设备 (I/O_1 和 I/O_2) 同时运行。三个作业的优先级为 A 最高、B 次之、C 最低，一旦低优先级的进程开始占用 CPU，高优先级进程也要等待到其结束后方可占用 CPU，请回答下面的问题：

- 1) 最早结束的作业是哪个？
- 2) 最后结束的作业是哪个？
- 3) 计算这段时间 CPU 的利用率（三个作业全部结束为止）。
6. 假定要在一台处理器上执行下表所示的作业，且假定这些作业在时刻 0 以 1, 2, 3, 4, 5 的顺序到达。说明分别使用 FCFS、RR (时间片=1)、SJF 及非剥夺式优先级调度算法时，这些作业的执行情况（优先级的高低顺序依次为 1 到 5）。

针对上述每种调度算法，给出平均周转时间和平均带权周转时间。

作业	执行时间	优先级
1	10	3
2	1	1
3	2	3
4	1	4
5	5	2

7. 有一个具有两道作业的批处理系统，作业调度采用短作业优先调度算法，进程调度采用抢占式优先级调度算法。作业的运行情况见下表，其中作业的优先数即进程的优先数，优先数越小，优先级越高。

作业名	到达时间	运行时间	优先数
1	8:00	40 分钟	5
2	8:20	30 分钟	3
3	8:30	50 分钟	4
4	8:50	20 分钟	6

- 1) 列出所有作业进入内存的时间及结束的时间(以分为单位)。
- 2) 计算平均周转时间。
8. 假设某计算机系统有 4 个进程, 各进程的预计运行时间和到达就绪队列的时刻见下表(相对时间, 单位为“时间配额”)。试用可抢占式短进程优先调度算法和时间片轮转调度算法进行调度(时间配额为 2)。分别计算各个进程的调度次序及平均周转时间。
- | 进程 | 到达就绪队列时刻 | 预计运行时间 |
|----------------|----------|--------|
| P ₁ | 0 | 8 |
| P ₂ | 1 | 4 |
| P ₃ | 2 | 9 |
| P ₄ | 3 | 5 |
9. 假设一个计算机系统具有如下性能特征: 处理一次中断平均需要 500μs, 一次进程调度平均需要花费 1ms, 进程的切换平均需要花费 2ms。若该计算机系统的定时器每秒发出 120 次时钟中断, 忽略其他 I/O 中断的影响, 请问:
- 1) 操作系统将百分之几的 CPU 时间分配给时钟中断处理程序?
- 2) 若系统采用时间片轮转调度算法, 24 个时钟中断为一个时间片, 操作系统每进行一次进程的切换, 需要花费百分之几的 CPU 时间?
- 3) 根据上述结果, 说明为了提高 CPU 的使用效率, 可以采用什么对策。
10. 【2016 统考真题】某进程调度程序采用基于优先数(priority)的调度策略, 即选择优先数最小的进程运行, 进程创建时由用户指定一个 nice 作为静态优先数。为了动态调整优先数, 引入运行时间 cpuTime 和等待时间 waitTime, 初值均为 0。进程处于执行态时, cpuTime 定时加 1, 且 waitTime 置 0; 进程处于就绪态时, cpuTime 置 0, waitTime 定时加 1。请回答下列问题:
- 1) 若调度程序只将 nice 的值作为进程的优先数, 即 priority = nice, 则可能会出现饥饿现象。为什么?
- 2) 使用 nice, cpuTime 和 waitTime 设计一种动态优先数计算方法, 以避免产生饥饿现象, 并说明 waitTime 的作用。
11. 设有 4 个作业 J₁, J₂, J₃, J₄, 它们的到达时间和计算时间见下表。若这 4 个作业在一台处理器上按单道方式运行, 采用高响应比优先调度算法, 试写出各作业的执行顺序、各作业的周转时间及平均周转时间。

作业	到达时间	计算时间
J ₁	8:00	2h
J ₂	8:30	40min
J ₃	9:00	25min
J ₄	9:30	30min

12. 在一个有两道作业的批处理系统中, 有一作业序列, 其到达时间及估计运行时间见下表。

系统作业采用最高响应比优先调度算法 [响应比 = (等待时间 + 估计运行时间)/估计运行时间]。进程的调度采用短进程优先的抢占式调度算法。

作业	到达时间/min	估计运行时间/min
J ₁	10:00	35
J ₂	10:10	30
J ₃	10:15	45
J ₄	10:20	20
J ₅	10:30	30

- 1) 列出各作业的执行时间 (即列出每个作业运行的时间片段, 如作业 i 的运行时间序列为 10:00—10:40, 11:00—11:20, 11:30—11:50 结束)。
- 2) 计算这批作业的平均周转时间。

2.2.8 答案与解析

一、单项选择题

1. A

时间片轮转的主要目的是, 使得多个交互的用户能够得到及时响应, 使得用户以为“独占”计算机的使用, 因此它并没有偏好, 也不会对特殊进程做特殊服务。时间片轮转增加了系统开销, 所以不会使得系统高效运转, 吞吐量和周转时间均不如批处理。但其较快速的响应时间使得用户能够与计算机进行交互, 改善了人机环境, 满足用户需求。

2. C

进程调度的时机与进程特点有关, 如进程是 CPU 繁忙型还是 I/O 繁忙型、自身的优先级等。但仅有这些特点是不够的, 能否得到调度还取决于进程调度策略, 若采用优先级调度算法, 则进程的优先级才起作用。至于占用处理器运行时间的长短, 则要看进程自身, 若进程是 I/O 繁忙型, 运行过程中要频繁访问 I/O 端口, 即可能会频繁放弃 CPU, 所以占用 CPU 的时间不会长, 一旦放弃 CPU, 则必须等待下次调度。若进程是 CPU 繁忙型, 则一旦占有 CPU, 就可能会运行很长时间, 但运行时间还取决于进程调度策略, 大部分情况下, 交互式系统为改善用户的响应时间, 大多数采用时间片轮转的算法, 这种算法在进程占用 CPU 达到一定时间后, 会强制将其换下, 以保证其他进程的 CPU 使用权。所以选择 C 选项。

3. B

先来先服务 (FCFS) 调度算法是一种最简单的调度算法, 在作业调度中采用该算法时, 每次调度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。

FCFS 调度算法比较有利于长作业, 而不利于短作业。所谓 CPU 繁忙型的作业, 是指该类作业需要大量的 CPU 时间进行计算, 而很少请求 I/O 操作。I/O 繁忙型的作业是指 CPU 处理时, 需频繁地请求 I/O 操作。所以 CPU 繁忙型作业更接近于长作业。答案选择 B 选项。

4. D

在选择进程调度算法时应考虑以下几个准则: ① 公平: 确保每个进程获得合理的 CPU 份额; ② 有效: 使 CPU 尽可能地忙碌; ③ 响应时间: 使交互用户的响应时间尽可能短; ④ 周转时间: 使批处理用户等待输出的时间尽可能短; ⑤ 吞吐量: 使单位时间处理的进程数尽可能最多。由此可见 D 选项不正确。

5. B

4 个作业的周转时间分别是 2h, 4h, 6h, 8h, 所以 4 个作业的总周转时间为 $2 + 4 + 6 + 8 = 20\text{h}$ 。此时, 平均周转时间 = 各个作业周转时间之和/作业数 = $20/4 = 5$ 小时。

6. B、E、C、D

照顾短作业用户, 选择短作业优先调度算法; 照顾紧急作业用户, 即选择优先级高的作业优先调度, 采用基于优先级的剥夺调度算法; 实现人机交互, 要保证每个作业都能在一定时间内轮到, 采用时间片轮转法; 使各种作业用户满意, 要处理多级反馈, 所以选择多级反馈队列调度算法。

7. D

优先级调度算法分静态和动态两种。静态优先级在进程创建时确定, 之后不再改变。

8. B

系统采用短作业优先调度算法, 作业的执行顺序为 J_1, J_2, J_3 , J_1 的周转时间为 T_1 , J_2 的周转时间为 $T_1 + T_2$, J_3 的周转时间为 $T_1 + T_2 + T_3$, 则平均周转时间为 $(T_1 + T_1 + T_2 + T_1 + T_2 + T_3)/3 = (3T_1 + 2T_2 + T_3)/3$ 。

9. D

在同一台处理器上以单道方式运行时, 要想获得最短的平均周转时间, 用短作业优先调度算法会有较好的效果。就本题目而言: A 选项的平均周转时间 $= (2 + 7 + 10)/3\text{h} = 19/3\text{h}$; B 选项的平均周转时间 $= (3 + 8 + 10)/3\text{h} = 7\text{h}$; C 选项的平均周转时间 $= (5 + 7 + 10)/3\text{h} = 22/3\text{h}$; D 选项的平均周转时间 $= (2 + 5 + 10)/3\text{h} = 17/3\text{h}$ 。

10. B

为了合理地设置进程优先级, 应综合考虑进程的 CPU 时间和 I/O 时间。对于优先级调度算法, 一般来说, I/O 型作业的优先权高于计算型作业的优先权, 这是由于 I/O 操作需要及时完成, 它没有办法长时间地保存所要输入/输出的数据, 所以考虑到系统资源利用率, 要选择 I/O 繁忙型作业有更高的优先级。

11. C

处于运行态的进程用完一个时间片后, 其状态会变为就绪态, 等待下一次处理器调度。进程执行完最后的语句并使用系统调用 exit 请求操作系统删除它或出现一些异常情况时, 进程才会终止。

12. C

$$\text{响应比} = \frac{\text{响应时间}}{\text{要求服务时间}} = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} = \frac{2+1}{1} = 3$$

13. D

优先级算法中, I/O 繁忙型作业要优于计算繁忙型作业, 系统进程的优先权应高于用户进程的优先权。作业的优先权与长作业、短作业或系统资源要求的多少没有必然的关系。在动态优先权中, 随着进程执行时间的增加其优先权随之降低, 随着作业等待时间的增加其优先权相应上升。

14. B

时间片轮转算法是按固定的时间配额来运行的, 时间一到, 不管是否完成, 当前的进程必须撤下, 调度新的进程, 因此它是由时间配额决定的、是绝对可抢占的。而优先级算法和短进程优先算法都可分为抢占式和不可抢占式。

15. B

作业是从用户角度出发的, 它由用户提交, 以用户任务为单位; 进程是从操作系统出发的, 由系统生成, 是操作系统的资源分配和独立运行的基本单位。

16. D

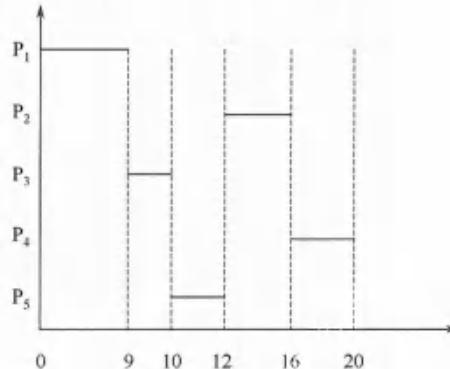
响应比 $R = (\text{等待时间} + \text{执行时间})/\text{执行时间}$ 。它综合考虑了每个进程的等待时间和执行时间，对于同时到达的长进程和短进程，短进程会优先执行，以提高系统吞吐量；而长进程的响应比可以随等待时间的增加而提高，不会产生进程无法调度的情况。

17. B

时间片轮转调度算法在实际运行中也按先后顺序使用时间片，时间片过大时，我们可以认为其大于进程需要的运行时间，即转变为先来先服务调度算法。

18. D

对于这种类型的题目，我们可以采用广义甘特图来求解，甘特图的画法在 1.2 节的习题中已经有所介绍。我们直接给出甘特图（见下图），以非抢占为例。



在 0 时刻，进程 P₁ 到达，于是处理器分配给 P₁，由于是不可抢占的，所以 P₁ 一旦获得处理器，就会运行直到结束；在 9 时刻，所有进程已经到达，根据短进程优先调度，会把处理器分配给 P₃，接下来就是 P₅；然后，由于 P₂、P₄ 的预计运行时间一样，所以在 P₂ 和 P₄ 之间用先来先服务调度，优先把处理器分配给 P₂，最后再分配给 P₄，完成任务。

周转时间 = 完成时间 - 作业到达时间，从图中显然可以得到各进程的完成时间，于是 P₁ 的周转时间是 $9 - 0 = 9$ ；P₂ 的周转时间是 $16 - 0.4 = 15.6$ ；P₃ 的周转时间是 $10 - 1 = 9$ ；P₄ 的周转时间是 $20 - 5.5 = 14.5$ ；P₅ 的周转时间是 $12 - 7 = 5$ ；平均周转时间为 $(9 + 15.6 + 9 + 14.5 + 5)/5 = 10.62$ 。

同理，抢占式的周转时间也可通过画甘特图求得，而且直观、不易出错。

抢占式的平均周转时间为 6.8。

甘特图在操作系统中有着广泛的应用，本节习题中会有不少这种类型的题目，若读者按照上面的方法求解，则解题时就可以做到胸有成竹。

19. D

这 5 个批处理作业采用短作业优先调度算法时，平均周转时间 = $[2 + (2 + 4) + (2 + 4 + 6) + (2 + 4 + 6 + 8) + (2 + 4 + 6 + 8 + 10)]/5 = 14$ 。

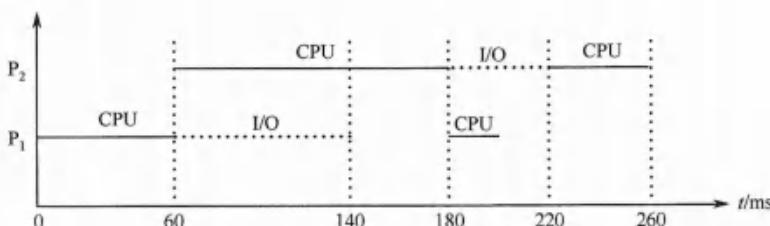
这道题主要考查读者对各种优先调度算法的认识。若按照 18 题中的方法求解，则可能要花费一定的时间，但这是值得的，因为可以起到熟练基本方法的效果。在考试中很少会遇到操作量和计算量如此大的题目，所以读者不用担心。

20. D

先来先服务调度算法是作业来得越早，优先级越高，因此会选择 J₁。短作业优先调度算法是作业运行时间越短，优先级越高，因此会选择 J₃。所以选项 D 正确。

21. B

由于 P₂ 比 P₁ 晚 5ms 到达，P₁ 先占用 CPU，作业运行的甘特图如下。



22. B

这类调度题目最好画图。因 CPU、输入设备、输出设备都只有一个，因此各操作步骤不能重叠，画出运行时的甘特图后，就能清楚地看到不同作业间的时序关系，如下图所示。

作业时间	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	输入		计算		输出												
2			输入		计算				输出								
3				输入				计算						输出			

23. B

进程切换带来系统开销，切换次数越多，开销越大，选项 A 正确。当前进程的时间片用完后，其状态由执行态变为就绪态，选项 B 错误。时钟中断是系统中特定的周期性时钟节拍，操作系统通过它来确定时间间隔，实现时间的延时和任务的超时，选项 C 正确。现代操作系统为了保证性能最优，通常根据响应时间、系统开销、进程数量、进程运行时间、进程切换开销等因素确定时间片大小，选项 D 正确。

24. A

分时系统需要同时满足多个用户的需要，因此把处理器时间轮流分配给多个用户作业使用，即采用时间片轮转调度算法。

25. B

先来先服务调度算法中，若一个长进程（作业）先到达系统，则会使后面的许多短进程（作业）等待很长的时间，因此对短进程（作业）不利。

26. B

短进程优先调度算法具有最短的平均周转时间。平均周转时间 = 各进程周转时间之和/进程数。因为每个进程的执行时间都是固定的，所以变化的是等待时间，只有短进程优先算法能最小化等待时间。

下表总结了几种常见进程调度算法的特点，读者要在理解的基础上掌握。

	先来先服务	短作业优先	高响应比优先	时间片轮转	多级反馈队列
能否是可抢占	否	能	能	能	队列内算法不一定
能否是不可抢占	能	能	能	否	队列内算法不一定
优点	公平，实现简单	平均等待时间最少，效率最高	兼顾长短作业	兼顾长短作业	兼顾长短作业，有较好的响应时间，可行性强
缺点	不利于短作业	长作业会饥饿，估计时间不易确定	计算响应比的开销大	平均等待时间较长，上下文切换浪费时间	无
适用于	无	作业调度，批处理系统	无	分时系统	相当通用
默认决策模式	非抢占	非抢占	非抢占	抢占	抢占

27. A

I 选项正确，分时系统中，响应时间与时间片和用户数成正比。II 选项正确。III 选项错误，中断向量本身是用于存放中断服务例行程序的入口地址，因此中断向量地址就应是该入口地址的地址。IV 选项错误，中断由硬件保护并完成，主要是为了保证系统运行可靠、正确。提高处理速度也是一个好处，但不是主要目的。综上分析，III、IV 选项错误。

28. C

选项 A、B、D 显然属于可以进行处理机调度的情况。对于选项 C，当进程处于临界区时，说明进程正在占用处理机，只要不破坏临界资源的使用规则，就不会影响处理机的调度。比如，通常访问的临界资源可能是慢速的外设（如打印机），若在进程访问打印机时，不能进行处理机调度，则系统的性能将非常差。

29. B

响应比 = (等待时间 + 执行时间)/执行时间。高响应比优先算法在等待时间相同的情况下，作业执行时间越短，响应比越高，满足短任务优先。随着长作业等待时间的增加，响应比会变大，执行机会也会增大，因此不会发生饥饿现象。先来先服务和时间片轮转不符合短任务优先，非抢占式短任务优先会产生饥饿现象。

30. A

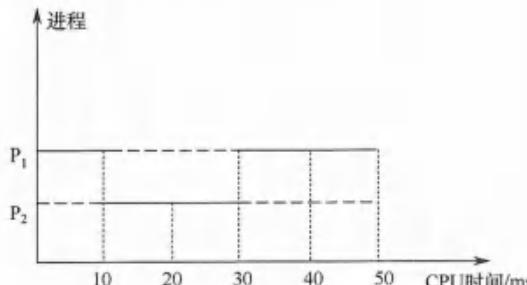
采用静态优先级调度且系统总是出现优先级高的任务时，优先级低的任务总是得不到处理机而产生饥饿现象；而短任务优先调度不管是抢占式的还是非抢占的，当系统总是出现新来的短任务时，长任务会总是得不到处理机，产生饥饿现象，因此选项 B、C、D 都错误。

31. D

由优先权可知，进程的执行顺序为 $P_2 \rightarrow P_3 \rightarrow P_1$ 。 P_2 的周转时间为 $1 + 15 + 24 = 40\mu s$ ； P_3 的周转时间为 $18 + 1 + 24 + 1 + 36 = 80\mu s$ ； P_1 的周转时间为 $30 + 1 + 24 + 1 + 36 + 1 + 12 = 105\mu s$ ；平均周转时间为 $(40 + 80 + 105)/3 = 225/3 = 75\mu s$ ，因此选 D。

32. C

进程 P_1 ， P_2 依次创建后进入队列 Q_1 ，根据时间片调度算法的规则，进程 P_1 ， P_2 将依次被分配 10ms 的 CPU 时间，两个进程分别执行完一个时间片后都会被转入队列 Q_2 ，就绪队列 Q_2 采用短进程优先调度算法，此时 P_1 还需要 20ms 的 CPU 时间， P_2 还需要 10ms 的 CPU 时间，所以 P_2 会被优先调度执行，10ms 后进程 P_2 执行完成，之后 P_1 再调度执行，再过 20ms 后 P_1 也执行完成。运行图表表述如下。



进程 P_1 ， P_2 的等待时间为图中的虚横线部分，平均等待时间 = $(P_1 \text{ 等待时间} + P_2 \text{ 等待时间})/2 = (20 + 10)/2 = 15$ ，因此答案选 C。

33. D

多级反馈队列调度算法需要综合考虑优先级数量、优先级之间的转换规则等，就绪队列的数量会影响长进程的最终完成时间，I 正确；就绪队列的优先级会影响进程执行的顺序，II 正确；各就绪队列的调度算法会影响各队列中进程的调度顺序，III 正确；进程在就绪队列中的迁移条件会影响各进程在各队列中的执行时间，IV 正确。

二、综合应用题

1. 解答：

多级反馈队列调度算法能较好地满足各种类型用户的需要。对终端型作业用户而言，由于它们提交的作业大多属于交互型作业，作业通常比较短小，系统只要能使这些作业在第 1 级队列所规定的时间片内完成，便可使终端型作业用户感到满意；对于短批处理作业用户而言，它们的作业开始时像终端型作业一样，若仅在第 1 级队列中执行一个时间片即可完成，便可获得与终端型作业一样的响应时间，对于稍长的作业，通常也只需要在第 2 级队列和第 3 级队列中各执行一个时间片即可完成，其周转时间仍然较短；对于长批处理作业用户而言，它们的长作业将依次在第 $1, 2, \dots, n$ 级队列中运行，然后按时间片轮转方式运行，用户不必担心其作业长期得不到处理。

2. 解答：

由题意可知，各类进程之间采用优先级调度算法，而同类进程内部采用时间片轮转调度算法。因此，系统首先对优先级为 4 的进程 P_1, P_2, P_3 采用时间片轮转调度算法运行；当 P_1, P_2, P_3 均运行结束或没有可运行的进程（即 P_1, P_2, P_3 都处于等待态；或其中部分进程已运行结束，其余进程处于等待态）时，对优先级为 3 的进程 P_4, P_5 采用时间片轮转调度算法运行。在此期间，若未结束的 P_1, P_2, P_3 有一个转为就绪态，则当前时间片用完后又回到优先级 4 进行调度。类似地，当 $P_1 \sim P_5$ 均运行结束或没有可运行进程（即 $P_1 \sim P_5$ 都处于等待态；或其中部分进程已运行结束，其余进程处于等待态）时，对优先级为 2 的进程 P_6, P_7, P_8 采用时间片轮转调度算法运行，一旦 $P_1 \sim P_5$ 中有一个转为就绪态，当前时间片用完后立即回到相应的优先级进行时间片轮转调度。

3. 解答：

这类实际的 CPU 和输入/输出设备调度的题目一定要画图，画出运行时的甘特图后就能清楚地看到不同进程间的时序关系，如下图所示。

	0	50	100	150	200	300
CPU	A	B	空闲	A	B	
输入设备	空闲		B		空闲	
打印机	空闲		A	空闲	A	

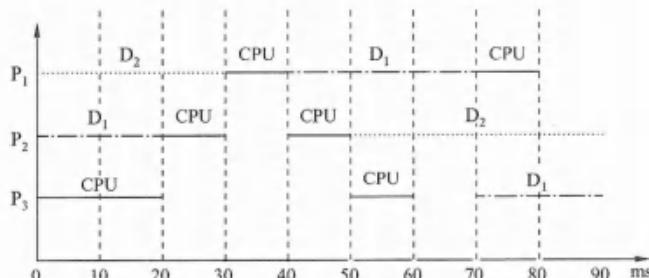
根据图中的进程时序关系：

- 1) 有，在 100~150ms 等待，利用率 = $[300 - (150 - 100)]/300 \times 100\% = 83.3\%$ 。
- 2) 无。
- 3) 有，在 0~50ms、180~200ms 时发生等待现象。

这里要提醒读者的是，甘特图的画法不止一种，上面用到的就是不常见的画法，但它起到的效果是一样的。若读者比较熟悉前面介绍的画法，则这道题用前面的画法也可轻松解决。

4. 解答：

抢占式优先级调度算法，三个作业执行的顺序如下图所示。



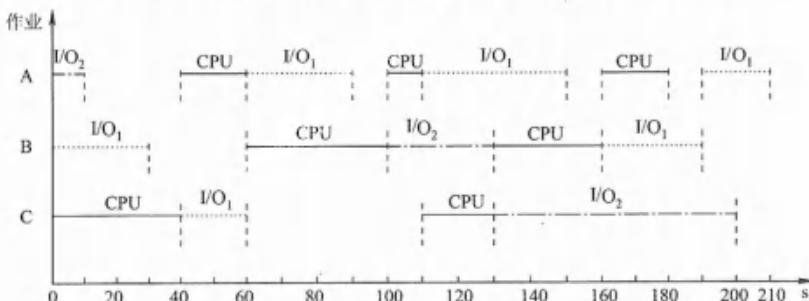
作业 P_1 的优先级最高, 周转时间等于运行时间, $T_1 = 80\text{ms}$; 作业 P_2 的等待时间为 10ms , 运行时间为 80ms , 周转时间 $T_2 = (10 + 80)\text{ms} = 90\text{ms}$; 作业 P_3 的等待时间为 40ms , 运行时间为 50ms , 因此周转时间 $T_3 = 90\text{ms}$ 。

三个作业从进入系统到全部运行结束, 时间为 90ms 。CPU 与外设都是独占设备, 运行时间为各作业的使用时间之和。CPU 运行时间为 $[(10 + 10) + 20 + 30]\text{ms} = 70\text{ms}$, D_1 为 $(30 + 20 + 20)\text{ms} = 70\text{ms}$, D_2 为 $(30 + 40)\text{ms} = 70\text{ms}$, 因此利用率均为 $70/90 = 77.8\%$ 。

5. 解答:

作业 A、B、C 的优先级依次递减, 采用不可抢占的优先级调度。

在时刻 40, 作业 C 释放 CPU, 优先级较高的作业 A 获得 CPU; 在时刻 60, 作业 A 释放 CPU, 优先级较高的作业 B 获得 CPU; 在时刻 100, 作业 B 释放 CPU, 优先级高的作业 A 获得 CPU; 在时刻 110, 作业 A 释放 CPU, 作业 C 获得 CPU; 在时刻 130, 作业 C 释放 CPU, 作业 B 获得 CPU; 在时刻 160, 作业 B 释放 CPU, 作业 A 获得 CPU。运行图如下所示。



- 1) 最早结束的是作业 B。
- 2) 最后结束的是作业 A。
- 3) 三个作业从开始到全部执行结束, 经历时间为 210s , 由于是单 CPU 系统, CPU 运行时间为各个作业的 CPU 运行时间之和, 即 $[(20 + 10 + 20) + (40 + 30) + (40 + 20)]\text{ms} = 180\text{ms}$ 。因此 CPU 的利用率为 $180/210 = 85.7\%$ 。

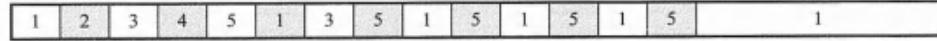
6. 解答:

- 1) 作业执行情况可以用如下的甘特图来表示。

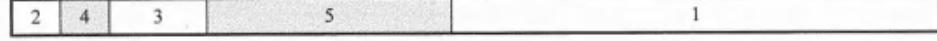
FCFS:



RR:



SJF:



优先级:



2) 各个作业对应于各个算法的周转时间和加权周转时间见下表。

算法	时间类型	P ₁	P ₂	P ₃	P ₄	P ₅	平均时间
	运行时间	10	1	2	1	5	3.8
FCFS	周转时间	10	11	13	14	19	13.4
	加权周转时间	1	11	6.5	14	3.8	7.26
RR	周转时间	19	2	7	4	14	9.2
	加权周转时间	1.9	2	3.5	4	2.8	2.84
SJF	周转时间	19	1	4	2	9	7
	加权周转时间	1.9	1	2	2	1.8	1.74
优先级	周转时间	16	1	18	19	6	12
	加权周转时间	1.6	1	9	19	1.2	6.36

所以，FCFS 的平均周转时间为 13.4，平均加权周转时间为 7.26。

RR 的平均周转时间为 9.2，平均加权周转时间为 2.84。

SJF 的平均周转时间为 7，平均加权周转时间为 1.74。

非剥夺式优先级调度算法的平均周转时间为 12，平均加权周转时间为 6.36。

注意：SJF 的平均周转时间肯定是最短的，计算完毕后可以利用这个性质进行检验。

7. 解答：

- 1) 具有两道作业的批处理系统，内存只存放两道作业，它们采用抢占式优先级调度算法竞争 CPU，而将作业调入内存采用的是短作业优先调度。8:00，作业 1 到来，此时内存和处理机空闲，作业 1 进入内存并占用处理机；8:20，作业 2 到来，内存仍有一个位置空闲，因此将作业 2 调入内存，又由于作业 2 的优先数高，相应的进程抢占处理机，在此期间 8:30 作业 3 到来，但内存此时已无空闲，因此等待。直至 8:50，作业 2 执行完毕，此时作业 3、4 竞争空出的一道内存空间，作业 4 的运行时间短，因此先调入，但它的优先数低于作业 1，因此作业 1 先执行。到 9:10 时，作业 1 执行完毕，再将作业 3 调入内存，且由于作业 3 的优先数高而占用 CPU。所有作业进入内存的时间及结束的时间见下表。

作业	到达时间	运行时间	优先数	进入内存时间	结束时间	周转时间
1	8:00	40min	5	8:00	9:10	70min
2	8:20	30min	3	8:20	8:50	30min
3	8:30	50min	4	9:10	10:00	90min
4	8:50	20min	6	8:50	10:20	90min

2) 平均周转时间为 $(70 + 30 + 90 + 90)/4 = 70\text{min}$ 。

8. 解答：

- 1) 按照可抢先式短进程优先调度算法，进程运行时间见下表。

进程	到达就绪队列时刻	预计执行时间	执行时间段	周转时间
P ₁	0	8	0~1; 10~17	17
P ₂	1	4	1~5	4
P ₃	2	9	17~26	24
P ₄	3	5	5~10	7

- 时刻 0, 进程 P₁到达并占用处理器运行。
- 时刻 1, 进程 P₂到达, 因其预计运行时间短, 因此抢夺处理器进入运行, P₁等待。
- 时刻 2, 进程 P₃到达, 因其预计运行时间长于正在运行的进程, 进入就绪队列等待。
- 时刻 3, 进程 P₄到达, 因其预计运行时间长于正在运行的进程, 进入就绪队列等待。
- 时刻 5, 进程 P₂运行结束, 调度器在就绪队列中选择短进程, P₄符合要求, 进入运行, 进程 P₁和进程 P₃则还在就绪队列等待。
- 时刻 10, 进程 P₄运行结束, 调度器在就绪队列中选择短进程, P₁符合要求, 再次进入运行, 而进程 P₃则还在就绪队列等待。
- 时刻 17, 进程 P₁运行结束, 只剩下进程 P₃, 调度其运行。
- 时刻 26, 进程 P₃运行结束。

$$\text{平均周转时间} = [(17 - 0) + (5 - 1) + (26 - 2) + (10 - 3)]/4 = 13.$$

2) 时间片轮转算法按就绪队列的 FCFS 进行轮转, 在时刻 2, P₁被挂到就绪队列队尾, 队列顺序为 P₂, P₃, P₁, 此时 P₄还未到达。按时间片轮转算法的进程时间分配见下表。

进程	到达就绪队列时刻	预计执行时间	执行时间段	周转时间
P ₁	0	8	0~2; 6~8; 14~16; 20~22	22
P ₂	1	4	2~4; 10~12	11
P ₃	2	9	4~6; 12~14; 18~20; 23~25; 25~26	24
P ₄	3	5	8~10; 16~18; 22~23	20

$$\text{平均周转时间} = ((22 - 0) + (12 - 1) + (26 - 2) + (23 - 3))/4 = 19.25.$$

9. 解答:

在时间片轮转调度算法中, 系统将所有就绪进程按到达时间的先后次序排成一个队列。进程调度程序总是选择队列中的第一个进程运行, 且仅能运行一个时间片。在使用完一个时间片后, 即使进程并未完成其运行, 也必须将处理器交给下一个进程。时间片轮转调度算法是绝对可抢先的算法, 由时钟中断来产生。

时间片的长短对计算机系统的影响很大。若时间片大到让一个进程足以完成其全部工作, 则这种算法就退化为先来先服务算法。若时间片很小, 则处理器在进程之间的转换工作会过于频繁, 处理器真正用于运行用户程序的时间将减少, 系统开销将增大。时间片的大小应能使分时用户得到好的响应时间, 同时也使系统具有较高的效率。

由题目给定条件可知:

- 1) 每秒产生 120 个时钟中断, 每次中断的时间为 $1/120 \approx 8.3\text{ms}$, 其中断处理耗时为 $500\mu\text{s}$, 那么其开销为 $500\mu\text{s}/8.3\text{ms} = 6\%$ 。
- 2) 每次进程切换需要 1 次调度、1 次切换, 所以需要耗时 $1\text{ms} + 2\text{ms} = 3\text{ms}$, 每 24 个时钟为一个时间片, $24 \times 8.3\text{ms} = 200\text{ms}$ 。一次切换所占 CPU 的时间比 $3\text{ms}/200\text{ms} = 1.5\%$ 。
- 3) 为提高 CPU 的效率, 一般情况下要尽量减少时钟中断的次数, 如由每秒 120 次降低到 100 次, 以延长中断的时间间隔。或将每个时间片的中断数量(时钟数)加大, 如由 24 个中断加大到 36 个。也可优化中断处理程序, 减少中断处理开销, 如将每次 $500\mu\text{s}$ 的时间降低到 $400\mu\text{s}$ 。若能这样, 则时钟中断和进程切换的总开销占 CPU 的时间比为 $(36 \times 400\mu\text{s} + 1\text{ms} + 2\text{ms})/(1/100 \times 36) \approx 4.8\%$ 。

10. 解答:

- 1) 由于采用了静态优先数, 当就绪队列中总有优先数较小的进程时, 优先数较大的进程一

直没有机会运行，因而会出现饥饿现象。

- 2) 优先数 priority 的计算公式为 $\text{priority} = \text{nice} + k_1 \times \text{cpuTime} - k_2 \times \text{waitTime}$ ，其中 $k_1 > 0$, $k_2 > 0$ ，用于分别调整 cpuTime 和 waitTime 在 priority 中所占的比例。waitTime 可使长时间等待的进程优先数减少，从而避免出现饥饿现象。

11. 解答：

作业的响应比可表示为

$$\text{响应比} = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}}$$

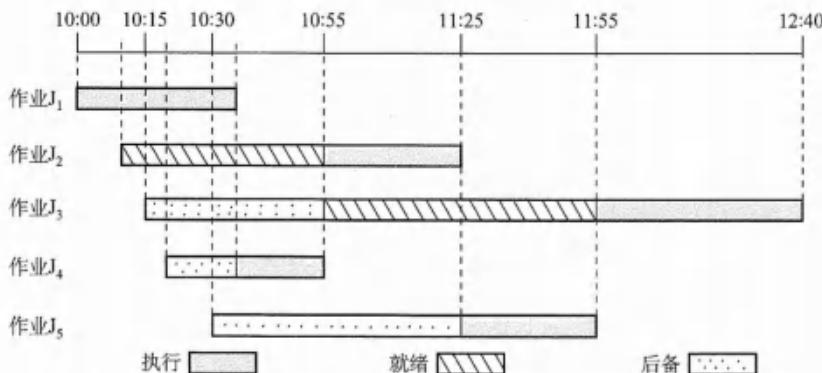
在时刻 8:00，系统中只有一个作业 J_1 ，因此系统将它投入运行。在 J_1 完成（即 10:00）时， J_2, J_3, J_4 的响应比分别为 $(90 + 40)/40, (60 + 25)/25, (30 + 30)/30$ ，即 3.25, 3.4, 2，因此应先将 J_3 投入运行。在 J_3 完成（即 10:25）时， J_2, J_4 的响应比分别为 $(115 + 40)/40, (55 + 30)/30$ ，即 3.875, 2.83，因此应先将 J_2 投入运行，待它运行完毕时（即 11:05），再将 J_4 投入运行， J_4 的结束时间为 11:35。

可见作业的执行次序为 J_1, J_3, J_2, J_4 ，各作业的运行情况见下表，它们的周转时间分别为 120min, 155min, 85min, 125min，平均周转时间为 121.25min。

作业号	提交时间	开始时间	执行时间	结束时间	周转时间
1	8:00	8:00	2h	10:00	120min
2	8:30	10:25	40min	11:05	155min
3	9:00	10:00	25min	10:25	85min
4	9:30	11:05	30min	11:35	125min

12. 解答：

上述 5 个作业的运行情况如下图所示。



在 10:00，因为只有 J_1 到达，因此将它调入内存，并将 CPU 调度给它。

在 10:10， J_2 到达，因此将 J_2 调入内存，但由于 J_1 只需再执行 25min，因此 J_1 继续执行。

虽然 J_3, J_4, J_5 分别在 10:15, 10:20 和 10:30 到达，但因当时内存中已存放了两道作业，因此不能马上将它们调入内存。

在 10:35, J_1 结束。此时 J_3, J_4, J_5 的响应比 [根据题意，响应比 = (等待时间 + 估计运行时间)/估计运行时间] 分别为 $65/45, 35/20, 35/30$ ，因此将 J_4 调入内存，并将 CPU 分配给内存中运行时间最短者，即 J_4 。

在 10:55, J_4 结束。此时 J_3, J_5 的响应比分别为 $85/45, 55/30$ ，因此将 J_3 调入内存，并将 CPU 分配给估计运行时间较短的 J_2 。

在 11:25, J_2 结束, 作业调度程序将 J_5 调入内存, 并将 CPU 分配给估计运行时间较短的 J_5 。在 11:55, J_5 结束, 将 CPU 分配给 J_3 。

在 12:40, J_3 结束。

通过上述分析, 可知:

- 1) 作业 1 的执行时间片段为 10:00—10:35 (结束)。
作业 2 的执行时间片段为 10:55—11:25 (结束)。
作业 3 的执行时间片段为 11:55—12:40 (结束)。
作业 4 的执行时间片段为 10:35—10:55 (结束)。
作业 5 的执行时间片段为 11:25—11:55 (结束)。
- 2) 它们的周转时间分别为 35min, 75min, 145min, 35min, 85min, 因此它们的平均周转时间为 75min。

2.3 进程同步

在学习本节时, 请读者思考以下问题:

- 1) 为什么要引入进程同步的概念?
- 2) 不同的进程之间会存在什么关系?
- 3) 当单纯用本节介绍的方法解决这些问题时会遇到什么新的问题吗?

用 PV 操作解决进程之间的同步互斥问题是这一节的重点, 考试已经多次考查过这一内容, 读者务必多加练习, 掌握好求解问题的方法。

2.3.1 进程同步的基本概念

在多道程序环境下, 进程是并发执行的, 不同进程之间存在着不同的相互制约关系。为了协调进程之间的相互制约关系, 引入了进程同步的概念。下面举一个简单的例子来帮大家理解这个概念。例如, 让系统计算 $1 + 2 \times 3$, 假设系统产生两个进程: 一个是加法进程, 一个是乘法进程。要让计算结果是正确的, 一定要让加法进程发生在乘法进程之后, 但实际上操作系统具有异步性, 若不加以制约, 加法进程发生在乘法进程之前是绝对有可能的, 因此要制定一定的机制去约束加法进程, 让它在乘法进程完成之后才发生, 而这种机制就是本节要讨论的内容。

1. 临界资源

虽然多个进程可以共享系统中的各种资源, 但其中许多资源一次只能为一个进程所用, 我们将一次仅允许一个进程使用的资源称为临界资源。许多物理设备都属于临界资源, 如打印机等。此外, 还有许多变量、数据等都可以被若干进程共享, 也属于临界资源。

对临界资源的访问, 必须互斥地进行, 在每个进程中, 访问临界资源的那段代码称为临界区。为了保证临界资源的正确使用, 可把临界资源的访问过程分成 4 个部分:

- 1) 进入区。为了进入临界区使用临界资源, 在进入区要检查可否进入临界区, 若能进入临界区, 则应设置正在访问临界区的标志, 以阻止其他进程同时进入临界区。
- 2) 临界区。进程中访问临界资源的那段代码, 又称临界段。
- 3) 退出区。将正在访问临界区的标志清除。
- 4) 剩余区。代码中的其余部分。

do {

```

entry section;           //进入区
critical section;        //临界区
exit section;            //退出区
remainder section;       //剩余区
) while(true)

```

2. 同步

同步亦称直接制约关系，是指为完成某种任务而建立的两个或多个进程，这些进程因为需要在某些位置上协调它们的工作次序而等待、传递信息所产生的制约关系。进程间的直接制约关系源于它们之间的相互合作。

例如，输入进程 A 通过单缓冲向进程 B 提供数据。当该缓冲区空时，进程 B 不能获得所需数据而阻塞，一旦进程 A 将数据送入缓冲区，进程 B 就被唤醒。反之，当缓冲区满时，进程 A 被阻塞，仅当进程 B 取走缓冲数据时，才唤醒进程 A。

3. 互斥

互斥也称间接制约关系。当一个进程进入临界区使用临界资源时，另一个进程必须等待，当占用临界资源的进程退出临界区后，另一进程才允许去访问此临界资源。

例如，在仅有一台打印机的系统中，有两个进程 A 和进程 B，若进程 A 需要打印时，系统已将打印机分配给进程 B，则进程 A 必须阻塞。一旦进程 B 将打印机释放，系统便将进程 A 唤醒，并将其由阻塞态变为就绪态。

为禁止两个进程同时进入临界区，同步机制应遵循以下准则：

- 1) 空闲让进。临界区空闲时，可以允许一个请求进入临界区的进程立即进入临界区。
- 2) 忙则等待。当已有进程进入临界区时，其他试图进入临界区的进程必须等待。
- 3) 有限等待。对请求访问的进程，应保证能在有限时间内进入临界区。
- 4) 让权等待。当进程不能进入临界区时，应立即释放处理器，防止进程忙等待。

2.3.2 实现临界区互斥的基本方法

1. 软件实现方法

在进入区设置并检查一些标志来标明是否有进程在临界区中，若已有进程在临界区，则在进入区通过循环检查进行等待，进程离开临界区后则在退出区修改标志。

1) 算法一：单标志法。该算法设置一个公用整型变量 turn，用于指示被允许进入临界区的进程编号，即若 turn = 0，则允许 P₀ 进程进入临界区。该算法可确保每次只允许一个进程进入临界区。但两个进程必须交替进入临界区，若某个进程不再进入临界区，则另一个进程也将无法进入临界区（违背“空闲让进”）。这样很容易造成资源利用不充分。

若 P₀ 顺利进入临界区并从临界区离开，则此时临界区是空闲的，但 P₁ 并没有进入临界区的打算，turn = 1 一直成立，P₀ 就无法再次进入临界区（一直被 while 死循环困住）。

P ₀ 进程:	P ₁ 进程:
while(turn!=0);	while(turn!=1);
critical section;	critical section;
turn=1;	turn=0;
remainder section;	remainder section;

2) 算法二：双标志法先检查。该算法的基本思想是在每个进程访问临界区资源之前，先查看临界资源是否正被访问，若正被访问，该进程需等待；否则，进程才进入自己的临界区。为此，设置一个数据 flag[i]，如第 i 个元素值为 FALSE，表示 P_i 进程未进入临界区，

值为 TRUE，表示 P_i 进程进入临界区。

P_i 进程：

```
while(flag[j]);      ①
flag[i]=TRUE;       ③
critical section;
flag[i]=FALSE;
remainder section;
```

P_j 进程：

```
while(flag[i]);      ② //进入区
flag[j]=TRUE;       ④ //进入区
critical section;   //临界区
flag[j]=FALSE;      //退出区
remainder section;  //剩余区
```

优点：不用交替进入，可连续使用；缺点： P_i 和 P_j 可能同时进入临界区。按序列①②③④执行时，会同时进入临界区（违背“忙则等待”）。即在检查对方的 flag 后和切换自己的 flag 前有一段时间，结果都检查通过。这里的问题出在检查和修改操作不能一次进行。

- 3) 算法三：双标志法后检查。算法二先检测对方的进程状态标志，再置自己的标志，由于在检测和放置中可插入另一个进程到达时的检测操作，会造成两个进程在分别检测后同时进入临界区。为此，算法三先将自己的标志设置为 TRUE，再检测对方的状态标志，若对方标志为 TRUE，则进程等待；否则进入临界区。

P_i 进程：

```
flag[i]=TRUE;
while(flag[j]);
critical section;
flag[i]=FALSE;
remainder section;
```

P_j 进程：

```
flag[j]=TRUE;          //进入区
while(flag[i]);        //进入区
critical section;     //临界区
flag[j]=FALSE;         //退出区
remainder section;    //剩余区
```

两个进程几乎同时都想进入临界区时，它们分别将自己的标志值 flag 设置为 TRUE，并且同时检测对方的状态（执行 while 语句），发现对方也要进入临界区时，双方互相谦让，结果谁也进不了临界区，从而导致“饥饿”现象。

- 4) 算法四：Peterson's Algorithm。为了防止两个进程为进入临界区而无限期等待，又设置了变量 turn，每个进程在先设置自己的标志后再设置 turn 标志。这时，再同时检测另一个进程状态标志和不允许进入标志，以便保证两个进程同时要求进入临界区时，只允许一个进程进入临界区。

P_i 进程：

```
flag[i]=TRUE;turn=j;
while(flag[j]&&turn==j);
critical section;
flag[i]=FALSE;
remainder section;
```

P_j 进程：

```
flag[j]=TRUE;turn=i;           //进入区
while(flag[i]&&turn==i);      //进入区
critical section;             //临界区
flag[j]=FALSE;                //退出区
remainder section;            //剩余区
```

具体如下：考虑进程 P_i ，一旦设置 $flag[i] = true$ ，就表示它想要进入临界区，同时 $turn = j$ ，此时若进程 P_j 已在临界区中，符合进程 P_i 中的 while 循环条件，则 P_i 不能进入临界区。若 P_j 不想要进入临界区，即 $flag[j] = false$ ，循环条件不符合，则 P_i 可以顺利进入，反之亦然。本算法的基本思想是算法一和算法三的结合。利用 flag 解决临界资源的互斥访问，而利用 turn 解决“饥饿”现象。

理解 Peterson's Algorithm 的最好方法就是手动模拟。

2. 硬件实现方法

理解本节介绍的硬件实现，对学习后面的信号量很有帮助。计算机提供了特殊的硬件指令，允许对一个字中的内容进行检测和修正，或对两个字的内容进行交换等。通过硬件支持实现临界段问题的方法称为低级方法，或称元方法。

(1) 中断屏蔽方法

当一个进程正在使用处理器执行它的临界区代码时，防止其他进程进入其临界区进行访问的最简方法是，禁止一切中断发生，或称之为屏蔽中断、关中断。因为 CPU 只在发生中断时引起进程切换，因此屏蔽中断能够保证当前运行的进程让临界区代码顺利地执行完，进而保证互斥的正确实现，然后执行开中断。其典型模式为

```
:
关中断;
临界区;
开中断;
:
```

这种方法限制了处理器交替执行程序的能力，因此执行的效率会明显降低。对内核来说，在它执行更新变量或列表的几条指令期间，关中断是很方便的，但将关中断的权力交给用户则很不明智，若一个进程关中断后不再开中断，则系统可能会因此终止。

(2) 硬件指令方法

TestAndSet 指令：这条指令是原子操作，即执行该代码时不允许被中断。其功能是读出指定标志后把该标志设置为真。指令的功能描述如下：

```
boolean TestAndSet(boolean *lock) {
    boolean old;
    old=*lock;
    *lock=true;
    return old;
}
```

可以为每个临界资源设置一个共享布尔变量 lock，表示资源的两种状态：true 表示正被占用，初值为 false。在进程访问临界资源之前，利用 TestAndSet 检查和修改标志 lock；若有进程在临界区，则重复检查，直到进程退出。利用该指令实现进程互斥的算法描述如下：

```
while TestAndSet(&lock);
进程的临界区代码段;
lock=false;
进程的其他代码;
```

Swap 指令：该指令的功能是交换两个字（字节）的内容。其功能描述如下：

```
Swap(boolean *a, boolean *b) {
    boolean temp;
    Temp=*a;
    *a==*b;
    *b=temp;
}
```

注意：以上对 TestAndSet 和 Swap 指令的描述仅是功能实现，而并非软件实现的定义。事实上，它们是由硬件逻辑直接实现的，不会被中断。

应为每个临界资源设置一个共享布尔变量 lock，初值为 false；在每个进程中再设置一个局部布尔变量 key，用于与 lock 交换信息。在进入临界区前，先利用 Swap 指令交换 lock 与 key 的内容，然后检查 key 的状态；有进程在临界区时，重复交换和检查过程，直到进程退出。利用 Swap 指令实现进程互斥的算法描述如下：

```
key=true;
while(key!=false)
    Swap(&lock, &key);
进程的临界区代码段;
```

```
lock=false;
进程的其他代码;
```

硬件方法的优点：适用于任意数目的进程，而不管是单处理机还是多处理机；简单、容易验证其正确性。可以支持进程内有多个临界区，只需为每个临界区设立一个布尔变量。

硬件方法的缺点：进程等待进入临界区时要耗费处理器时间，不能实现让权等待。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，从而导致“饥饿”现象。

无论是软件实现方法还是硬件实现方法，读者只需理解它的执行过程即可，关键是软件实现方法。实际练习和考试中很少让读者写出某种软件和硬件实现方法，因此读者并不需要默写或记忆。以上的代码实现与我们平时在编译器上写的代码意义不同，以上的代码实现是为了表述进程实现同步和互斥的过程，并不是说计算机内部实现同步互斥的就是这些代码。

2.3.3 信号量

信号量机制是一种功能较强的机制，可用来解决互斥与同步问题，它只能被两个标准的原语 `wait(S)` 和 `signal(S)` 访问，也可记为“P 操作”和“V 操作”。

原语是指完成某种功能且不被分割、不被中断执行的操作序列，通常可由硬件来实现。例如，前述的 `Test-and-Set` 和 `Swap` 指令就是由硬件实现的原子操作。原语功能的不被中断执行特性在单处理机上可由软件通过屏蔽中断方法实现。

原语之所以不能被中断执行，是因为原语对变量的操作过程若被打断，可能会去运行另一个对同一变量的操作过程，从而出现临界段问题。若能够找到一种解决临界段问题的元方法，就可以实现对共享变量操作的原子性。

1. 整型信号量

整型信号量被定义为一个用于表示资源数目的整型量 S，`wait` 和 `signal` 操作可描述为

```
wait(S) {
    while(S<=0);
    S=S-1;
}
signal(S) {
    S=S+1;
}
```

`wait` 操作中，只要信号量 $S \leq 0$ ，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。

2. 记录型信号量

记录型信号量是不存在“忙等”现象的进程同步机制。除需要一个用于代表资源数目的整型变量 `value` 外，再增加一个进程链表 `L`，用于链接所有等待该资源的进程。记录型信号量得名于采用了记录型的数据结构。记录型信号量可描述为

```
typedef struct{
    int value;
    struct process *L;
} semaphore;
```

相应的 `wait(S)` 和 `signal(S)` 的操作如下：

```
void wait(semaphore S){      //相当于申请资源
    S.value--;
    if(S.value<0){
        add this process to S.L;
```

```

        block(S.L);
    }
}

```

wait 操作, S.value--表示进程请求一个该类资源, 当 S.value < 0 时, 表示该类资源已分配完毕, 因此进程应调用 block 原语, 进行自我阻塞, 放弃处理机, 并插入该类资源的等待队列 S.L, 可见该机制遵循了“让权等待”的准则。

```

void signal(semaphore S){ //相当于释放资源
    S.value++;
    if(S.value<=0){
        remove a process P from S.L;
        wakeup(P);
    }
}

```

signal 操作, 表示进程释放一个资源, 使系统中可供分配的该类资源数增 1, 因此有 S.value ++。若加 1 后仍是 S.value ≤ 0, 则表示在 S.L 中仍有等待该资源的进程被阻塞, 因此还应调用 wakeup 原语, 将 S.L 中的第一个等待进程唤醒。

3. 利用信号量实现同步

信号量机制能用于解决进程间的各种同步问题。设 S 为实现进程 P₁, P₂ 同步的公共信号量, 初值为 0。进程 P₂ 中的语句 y 要使用进程 P₁ 中语句 x 的运行结果, 所以只有当语句 x 执行完成之后语句 y 才可以执行。其实现进程同步的算法如下:

```

semaphore S=0; //初始化信号量
P1(){
     $\sqrt{10000}$ 
    x; //语句 x
    V(S); //告诉进程 P2, 语句 x 已经完成
    ...
}
P2(){
    ...
    P(S); //检查语句 x 是否运行完成
    y; //检查无误, 运行 y 语句
    ...
}

```

若 P₂ 先执行到 P(S) 时, S 为 0, 执行 P 操作会把进程 P₂ 阻塞, 并放入阻塞队列; 当进程 P₁ 中的 x 执行完后, 执行 V 操作, 把 P₂ 从阻塞队列中放回就绪队列, 当 P₂ 得到处理机时, 就得以继续执行。

4. 利用信号量实现进程互斥

信号量机制也能很方便地解决进程互斥问题。设 S 为实现进程 P₁, P₂ 互斥的信号量, 由于每次只允许一个进程进入临界区, 所以 S 的初值应为 1(即可用资源数为 1)。只需把临界区置于 P(S) 和 V(S) 之间, 即可实现两个进程对临界资源的互斥访问。其算法如下:

```

semaphore S=1; //初始化信号量
P1(){
    ...
    P(S); //准备开始访问临界资源, 加锁
    进程 P1 的临界区;
    V(S); //访问结束, 解锁
}

```

```

    ...
}

P2() {
    ...
    P(S); //准备开始访问临界资源，加锁
    进程 P2 的临界区;
    V(S); //访问结束，解锁
    ...
}

```

当没有进程在临界区时，任意一个进程要进入临界区，就要执行 P 操作，把 S 的值减为 0，然后进入临界区；当有进程存在于临界区时，S 的值为 0，再有进程要进入临界区，执行 P 操作时将会被阻塞，直至在临界区中的进程退出，这样便实现了临界区的互斥。

互斥是不同进程对同一信号量进行 P, V 操作实现的，一个进程成功对信号量执行了 P 操作后进入临界区，并在退出临界区后，由该进程本身对该信号量执行 V 操作，表示当前没有进程进入临界区，可以让其他进程进入。

下面简单总结一下 PV 操作在同步互斥中的应用：在同步问题中，若某个行为要用到某种资源，则在这个行为前面 P 这种资源一下；若某个行为会提供某种资源，则在这个行为后面 V 这种资源一下。在互斥问题中，P, V 操作要紧夹使用互斥资源的那个行为，中间不能有其他冗余代码。

5. 利用信号量实现前驱关系

信号量也可用来描述程序之间或语句之间的前驱关系。图 2.8 给出了一个前驱图，其中 $S_1, S_2, S_3, \dots, S_6$ 是最简单的程序段（只有一条语句）。为使各程序段能正确执行，应设置若干初始值为“0”的信号量。例如，为保证 $S_1 \rightarrow S_2, S_1 \rightarrow S_3$ 的前驱关系，应分别设置信号量 a_1, a_2 。同样，为保证 $S_2 \rightarrow S_4, S_2 \rightarrow S_5, S_3 \rightarrow S_6, S_4 \rightarrow S_6, S_5 \rightarrow S_6$ ，应设置信号量 b_1, b_2, c, d, e 。

实现算法如下：

```

semaphore a1=a2=b1=b2=c=d=e=0; //初始化信号量
S1() {
    ...
    V(a1); V(a2); //S1 已经运行完成
}
S2() {
    P(a1); //检查 S1 是否运行完成
    ...
    V(b1); V(b2); //S2 已经运行完成
}
S3() {
    P(a2); //检查 S1 是否已经运行完成
    ...
    V(c); //S3 已经运行完成
}
S4() {
    P(b1); //检查 S2 是否已经运行完成
    ...
    V(d); //S4 已经运行完成
}
S5() {
    ...
}

```

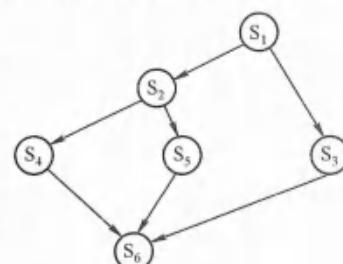


图 2.8 前驱图举例

```

P(b2); //检查 S2 是否已经运行完成
...
V(e); //S5 已经运行完成
}
S6() {
    P(c); //检查 S3 是否已经运行完成
    P(d); //检查 S4 是否已经运行完成
    P(e); //检查 S5 是否已经运行完成
    ...
}

```

6. 分析进程同步和互斥问题的方法步骤

- 1) 关系分析。找出问题中的进程数，并分析它们之间的同步和互斥关系。同步、互斥、前驱关系直接按照上面例子中的经典范式改写。
- 2) 整理思路。找出解决问题的关键点，并根据做过的题目找出求解的思路。根据进程的操作流程确定 P 操作、V 操作的大致顺序。
- 3) 设置信号量。根据上面的两步，设置需要的信号量，确定初值，完善整理。

这是一个比较直观的同步问题，以 S_2 为例，它是 S_1 的后继，所以要用到 S_1 的资源，在前面的简单总结中我们说过，在同步问题中，要用到某种资源，就要在行为（题中统一抽象成 L）前面 P 这种资源一下。 S_2 是 S_4, S_5 的前驱，给 S_4, S_5 提供资源，所以要在 L 行为后面 V 由 S_4 和 S_5 代表的资源一下。

2.3.4 管程

在信号量机制中，每个要访问临界资源的进程都必须自备同步的 PV 操作，大量分散的同步操作给系统管理带来了麻烦，且容易因同步操作不当而导致系统死锁。于是，便产生了一种新的进程同步工具——管程。管程的特性保证了进程互斥，无须程序员自己实现互斥，从而降低了死锁发生的可能性。同时管程提供了条件变量，可以让程序员灵活地实现进程同步。

1. 管程的定义

系统中的各种硬件资源和软件资源，均可用数据结构抽象地描述其资源特性，即用少量信息和对资源所执行的操作来表征该资源，而忽略它们的内部结构和实现细节。

利用共享数据结构抽象地表示系统中的共享资源，而把对该数据结构实施的操作定义为一组过程。进程对共享资源的申请、释放等操作，都通过这组过程来实现，这组过程还可以根据资源情况，或接受或阻塞进程的访问，确保每次仅有一个进程使用共享资源，这样就可以统一管理对共享资源的所有访问，实现进程互斥。这个代表共享资源的数据结构，以及由对该共享数据结构实施操作的一组过程所组成的资源管理程序，称为管程（monitor）。管程定义了一个数据结构和能为并发进程所执行（在该数据结构上）的一组操作，这组操作能同步进程和改变管程中的数据。

由上述定义可知，管程由 4 部分组成：

- ① 管程的名称；
- ② 局部于管程内部的共享结构数据说明；
- ③ 对该数据结构进行操作的一组过程（或函数）；
- ④ 对局部于管程内部的共享数据设置初始值的语句。

管程的定义描述举例如下：

```

monitor Demo { //①定义一个名称为“Demo”的管程
    //②定义共享数据结构，对应系统中的某种共享资源
}

```

```

共享数据结构 S;
//④对共享数据结构初始化的语句
init_code() {
    S=5;           //初始资源数等于 5
}
//③过程 1：申请一个资源
take_away() {
    对共享数据结构 x 的一系列处理;
    S--;           //可用资源数-1
    ...
}
//③过程 2：归还一个资源
give_back() {
    对共享数据结构 x 的一系列处理;
    S++;           //可用资源数+1
    ...
}
}
}

```

熟悉面向对象程序设计的读者看到管程的组成后，会立即联想到管程很像一个类（class）。

- 1) 管程把对共享资源的操作封装起来，管程内的共享数据结构只能被管程内的过程所访问。一个进程只有通过调用管程内的过程才能进入管程访问共享资源。对于上例，外部进程只能通过调用 take_away() 过程来申请一个资源；归还资源也一样。
- 2) 每次仅允许一个进程进入管程，从而实现进程互斥。若多个进程同时调用 take_away()，give_back()，则只有某个进程运行完它调用的过程后，下个进程才能开始运行它调用的过程。也就是说，各个进程只能串行执行管程内的过程，这一特性保证了进程“互斥”访问共享数据结构 S。

2. 条件变量

当一个进程进入管程后被阻塞，直到阻塞的原因解除时，在此期间，如果该进程不释放管程，那么其他进程无法进入管程。为此，将阻塞原因定义为条件变量 condition。通常，一个进程被阻塞的原因可以有多个，因此在管程中设置了多个条件变量。每个条件变量保存了一个等待队列，用于记录因该条件变量而阻塞的所有进程，对条件变量只能进行两种操作，即 wait 和 signal。

x.wait: 当 x 对应的条件不满足时，正在调用管程的进程调用 x.wait 将自己插入 x 条件的等待队列，并释放管程。此时其他进程可以使用该管程。

x.signal: x 对应的条件发生了变化，则调用 x.signal，唤醒一个因 x 条件而阻塞的进程。

下面给出条件变量的定义和使用：

```

monitor Demo{
    共享数据结构 S;
    condition x;           //定义一个条件变量 x
    init_code() { ... }
    take_away() {
        if(S<=0) x.wait();      //资源不够，在条件变量 x 上阻塞等待
        资源足够，分配资源，做一系列相应处理;
    }
    give_back() {
        归还资源，做一系列相应处理;
        if(有进程在等待) x.signal; //唤醒一个阻塞进程
    }
}

```

条件变量和信号量的比较：

相似点：条件变量的 wait/signal 操作类似于信号量的 P/V 操作，可以实现进程的阻塞/唤醒。

不同点：条件变量是“没有值”的，仅实现了“排队等待”功能；而信号量是“有值”的，信号量的值反映了剩余资源数，而在管程中，剩余资源数用共享数据结构记录。

2.3.5 经典同步问题

1. 生产者-消费者问题

问题描述：一组生产者进程和一组消费者进程共享一个初始为空、大小为 n 的缓冲区，只有缓冲区没满时，生产者才能把消息放入缓冲区，否则必须等待；只有缓冲区不空时，消费者才能从中取出消息，否则必须等待。由于缓冲区是临界资源，它只允许一个生产者放入消息，或一个消费者从中取出消息。

问题分析：

- 1) 关系分析。生产者和消费者对缓冲区互斥访问是互斥关系，同时生产者和消费者又是一个相互协作的关系，只有生产者生产之后，消费者才能消费，它们也是同步关系。
- 2) 整理思路。这里比较简单，只有生产者和消费者两个进程，正好是这两个进程存在着互斥关系和同步关系。那么需要解决的是互斥和同步 PV 操作的位置。
- 3) 信号量设置。信号量 mutex 作为互斥信号量，用于控制互斥访问缓冲池，互斥信号量初值为 1；信号量 full 用于记录当前缓冲池中的“满”缓冲区数，初值为 0。信号量 empty 用于记录当前缓冲池中的“空”缓冲区数，初值为 n 。

我们对同步互斥问题的介绍是一个循序渐进的过程。上面介绍了一个同步问题的例子和一个互斥问题的例子，下面来看生产者-消费者问题的例子是什么样的。

生产者-消费者进程的描述如下：

```

semaphore mutex=1;                                //临界区互斥信号量
semaphore empty=n;                               //空闲缓冲区
semaphore full=0;                                //缓冲区初始化为空
producer(){
    while(1){
        produce an item in nextp;                //生产数据
        P(empty); (要用什么, P一下)           //获取空缓冲区单元
        P(mutex); (互斥夹紧)                  //进入临界区
        add nextp to buffer; (行为)          //将数据放入缓冲区
        V(mutex); (互斥夹紧)                  //离开临界区, 释放互斥信号量
        V(full); (提供什么, V一下)          //满缓冲区数加1
    }
}
consumer(){                                         //消费者进程
    while(1){
        P(full);                                //获取满缓冲区单元
        P(mutex);                                //进入临界区
        remove an item from buffer;             //从缓冲区中取出数据
        V(mutex);                                //离开临界区, 释放互斥信号量
        V(empty);                                //空缓冲区数加1
        consume the item;                      //消费数据
    }
}

```

该类问题要注意对缓冲区大小为 n 的处理，当缓冲区中有空时，便可对 empty 变量执行 P 操作，一旦取走一个产品便要执行 V 操作以释放空闲区。对 empty 和 full 变量的 P 操作必须放在对 mutex 的 P 操作之前。若生产者进程先执行 P(mutex)，然后执行 P(empty)，消费者执行 P(mutex)，然后执行 P(full)，这样可不可以？答案是否定的。设想生产者进程已将缓冲区放满，消费者进程并没有取产品，即 empty = 0，当下次仍然是生产者进程运行时，它先执行 P(mutex) 封锁信号量，再执行 P(empty) 时将被阻塞，希望消费者取出产品后将其唤醒。轮到消费者进程运行时，它先执行 P(mutex)，然而由于生产者进程已经封锁 mutex 信号量，消费者进程也会被阻塞，这样一来生产者、消费者进程都将阻塞，都指望对方唤醒自己，因此陷入了无休止的等待。同理，若消费者进程已将缓冲区取空，即 full = 0，下次若还是消费者先运行，也会出现类似的死锁。不过生产者释放信号量时，mutex, full 先释放哪一个无所谓，消费者先释放 mutex 或 empty 都可以。

根据对同步互斥问题的简单总结，我们发现，其实生产者-消费者问题只是一个同步互斥问题的综合而已。

下面再看一个较为复杂的生产者-消费者问题。

问题描述：桌子上有两个盘子，每次只能向其中放入一个水果。爸爸专向盘子中放苹果，妈妈专向盘子中放橘子，儿子专等吃盘子中的橘子，女儿专等吃盘子中的苹果。只有盘子为空时，爸爸或妈妈才可向盘子中放一个水果；仅当盘子中有自己需要的水果时，儿子或女儿可以从盘子中取出。

问题分析：

- 1) **关系分析。**这里的关系要稍复杂一些。由每次只能向其中放入一只水果可知，爸爸和妈妈是互斥关系。爸爸和女儿、妈妈和儿子是同步关系，而且这两对进程必须连起来，儿子和女儿之间没有互斥和同步关系，因为他们是选择条件执行，不可能并发，如图 2.9 所示。
- 2) **整理思路。**这里有 4 个进程，实际上可抽象为两个生产者和两个消费者被连接到大小为 1 的缓冲区上。
- 3) **信号量设置。**首先将信号量 plate 设置互斥信号量，表示是否允许向盘子放入水果，初值为 1 表示允许放入，且只允许放入一个。信号量 apple 表示盘子中是否有苹果，初值为 0 表示盘子为空，不许取，apple = 1 表示可以取。信号量 orange 表示盘子中是否有橘子，初值为 0 表示盘子为空，不许取，orange = 1 表示可以取。

解决该问题的代码如下：

```

semaphore plate=1, apple=0, orange=0;
dad() {                                         //父亲进程
    while(1){
        prepare an apple;
        P(plate);                                //互斥向盘中取、放水果
        put the apple on the plate;               //向盘中放苹果
        V(apple);                                //允许取苹果
    }
}
mom() {                                         //母亲进程
    while(1){
        prepare an orange;
        P(plate);                               //互斥向盘中取、放水果
        ...
    }
}

```



图 2.9 进程之间的关系

```

        put the orange on the plate;           //向盘中放橘子
        V(orange);                          //允许取橘子
    }
}
son(){                           //儿子进程
    while(1){
        P(orange);                     //互斥向盘中取橘子
        take an orange from the plate;
        V(plate);                      //允许向盘中取、放水果
        eat the orange;
    }
}
daughter(){                      //女儿进程
    while(1){
        P(apple);                    //互斥向盘中取苹果
        take an apple from the plate;
        V(plate);                   //运行向盘中取、放水果
        eat the apple;
    }
}
}

```

进程间的关系如图 2.9 所示。dad() 和 daughter()、mom() 和 son() 必须连续执行，正因为如此，也只能在女儿拿走苹果后或儿子拿走橘子后才能释放盘子，即 V(plate) 操作。

2. 读者-写者问题

问题描述：有读者和写者两组并发进程，共享一个文件，当两个或以上的读进程同时访问共享数据时不会产生副作用，但若某个写进程和其他进程（读进程或写进程）同时访问共享数据时则可能导致数据不一致的错误。因此要求：① 允许多个读者可以同时对文件执行读操作；② 只允许一个写者往文件中写信息；③ 任一写者在完成写操作之前不允许其他读者或写者工作；④ 写者执行写操作前，应让已有的读者和写者全部退出。

问题分析：

- 1) **关系分析。**由题目分析读者和写者是互斥的，写者和写者也是互斥的，而读者和读者不存在互斥问题。
- 2) **整理思路。**两个进程，即读者和写者。写者是比较简单的，它和任何进程互斥，用互斥信号量的 P 操作、V 操作即可解决。读者的问题比较复杂，它必须在实现与写者互斥的同时，实现与其他读者的同步，因此简单的一对 P 操作、V 操作是无法解决问题的。这里用到了一个计数器，用它来判断当前是否有读者读文件。当有读者时，写者是无法写文件的，此时读者会一直占用文件，当没有读者时，写者才可以写文件。同时，这里不同读者对计数器的访问也应该是互斥的。
- 3) **信号量设置。**首先设置信号量 count 为计数器，用于记录当前读者的数量，初值为 0；设置 mutex 为互斥信号量，用于保护更新 count 变量时的互斥；设置互斥信号量 rw，用于保证读者和写者的互斥访问。

代码如下：

```

int count=0;                      //用于记录当前的读者数量
semaphore mutex=1;                //用于保护更新 count 变量时的互斥
semaphore rw=1;                   //用于保证读者和写者互斥地访问文件
writer(){                         //写者进程
    while(1){

```

```

    P(rw);           //互斥访问共享文件
    writing;         //写入
    V(rw);          //释放共享文件
}
}

reader() {           //读者进程
    while(1){
        P(mutex);      //互斥访问 count 变量
        if(count==0)   //当第一个读进程读共享文件时
            P(rw);       //阻止写进程写
        count++;        //读者计数器加 1
        V(mutex);       //释放互斥变量 count
        reading;        //读取
        P(mutex);       //互斥访问 count 变量
        count--;        //读者计数器减 1
        if(count==0)   //当最后一个读进程读完共享文件
            V(rw);       //允许写进程写
        V(mutex);       //释放互斥变量 count
    }
}
}

```

在上面的算法中，读进程是优先的，即当存在读进程时，写操作将被延迟，且只要有一个读进程活跃，随后而来的读进程都将被允许访问文件。这样的方式会导致写进程可能长时间等待，且存在写进程“饿死”的情况。

若希望写进程优先，即当有读进程正在读共享文件时，有写进程请求访问，这时应禁止后续读进程的请求，等到已在共享文件的读进程执行完毕，立即让写进程执行，只有在无写进程执行的情况下才允许读进程再次运行。为此，增加一个信号量并在上面程序的 writer() 和 reader() 函数中各增加一对 PV 操作，就可以得到写进程优先的解决程序。

```

int count=0;           //用于记录当前的读者数量
semaphore mutex=1;     //用于保护更新 count 变量的互斥
semaphore rw=1;         //用于保证读者和写者互斥地访问文件
semaphore w=1;          //用于实现“写优先”
writer() {              //写者进程
    while(1){
        P(w);           //在无写进程请求时进入
        P(rw);          //互斥访问共享文件
        writing;         //写入
        V(rw);          //释放共享文件
        V(w);           //恢复对共享文件的访问
    }
}
reader() {             //读者进程
    while(1){
        P(w);           //在无写进程请求时进入
        P(mutex);        //互斥访问 count 变量
        if (count==0)   //当第一个读进程读共享文件时
            P(rw);       //阻止写进程写
        count++;        //读者计数器加 1
        V(mutex);       //释放互斥变量 count
        V(w);           //恢复对共享文件的访问
        reading;        //读取
    }
}

```

```

P(mutex);
count--;
if (count==0)
    V(rw);
V(mutex);
}
}

```

这里的写进程优先是相对而言的，有些书上把这个算法称为读写公平法，即读写进程具有一样的优先级。当一个写进程访问文件时，若先有一些读进程要求访问文件，后有另一个写进程要求访问文件，则当前访问文件的进程结束对文件的写操作时，会是一个读进程而不是一个写进程占用文件（在信号量 w 的阻塞队列上，因为读进程先来，因此排在阻塞队列队首，而 V 操作唤醒进程时唤醒的是队首进程），所以说这里的写优先是相对的，想要了解如何做到真正写者优先，可参考其他相关资料。

读者-写者问题有一个关键的特征，即有一个互斥访问的计数器 count，因此遇到一个不太好解决的同步互斥问题时，要想一想用互斥访问的计数器 count 能否解决问题。

3. 哲学家进餐问题

问题描述：一张圆桌边上坐着 5 名哲学家，每两名哲学家之间的桌上摆一根筷子，两根筷子中间是一碗米饭，如图 2.10 所示。哲学家们倾注毕生精力用于思考和进餐，哲学家在思考时，并不影响他人。只有当哲学家饥饿时，才试图拿起左、右两根筷子（一根一根地拿起）。若筷子已在他人手上，则需要等待。饥饿的哲学家只有同时拿到了两根筷子才可以开始进餐，进餐完毕后，放下筷子继续思考。

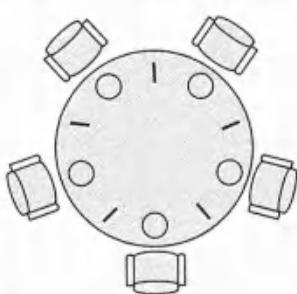


图 2.10 5 名哲学家进餐

问题分析：

- 关系分析。5 名哲学家与左右邻居对其中间筷子的访问是互斥关系。
- 整理思路。显然，这里有 5 个进程。本题的关键是如何让一名哲学家拿到左右两根筷子而不造成死锁或饥饿现象。解决方法有两个：一是让他们同时拿两根筷子；二是对每名哲学家的动作制定规则，避免饥饿或死锁现象的发生。
- 信号量设置。定义互斥信号量数组 chopstick[5] = {1, 1, 1, 1, 1}，用于对 5 个筷子的互斥访问。哲学家按顺序编号为 0~4，哲学家 i 左边筷子的编号为 i，哲学家右边筷子的编号为 (i+1)%5。

```

semaphore chopstick[5]={1,1,1,1,1}; //定义信号量数组 chopstick[5]，并初始化
Pi(){
    do{
        P(chopstick[i]);           //取左边筷子
        P(chopstick[(i+1)%5]);    //取右边筷子
        eat;                      //进餐
        V(chopstick[i]);          //放回左边筷子
        V(chopstick[(i+1)%5]);    //放回右边筷子
        think;                    //思考
    } while(1);
}

```

该算法存在以下问题：当 5 名哲学家都想要进餐并分别拿起左边的筷子时（都恰好执行完 wait(chopstick[i]);）筷子已被拿光，等到他们再想拿右边的筷子时（执行 wait(chopstick[(i+1)%5]);）

就全被阻塞，因此出现了死锁。

为防止死锁发生，可对哲学家进程施加一些限制条件，比如至多允许 4 名哲学家同时进餐；仅当一名哲学家左右两边的筷子都可用时，才允许他抓起筷子；对哲学家顺序编号，要求奇数号哲学家先拿左边的筷子，然后拿右边的筷子，而偶数号哲学家刚好相反。

制定的正确规则如下：假设采用第二种方法，当一名哲学家左右两边的筷子都可用时，才允许他抓起筷子。

```

semaphore chopstick[5]={1,1,1,1,1}; //初始化信号量
semaphore mutex=1; //设置取筷子的信号量
Pi(){
    do{
        P(mutex); //在取筷子前获得互斥量
        P(chopstick[i]); //取左边筷子
        P(chopstick[(i+1)%5]); //取右边筷子
        V(mutex); //释放取筷子的信号量
        eat; //进餐
        V(chopstick[i]); //放回左边筷子
        V(chopstick[(i+1)%5]); //放回右边筷子
        think; //思考
    } while(1);
}

```

此外，还可采用 AND 型信号量机制来解决哲学家进餐问题，有兴趣的读者可以查阅相关资料，自行思考。

熟悉 ACM 或有过相关训练的读者都应知道贪心算法，哲学家进餐问题的思想其实与贪心算法的思想截然相反，贪心算法强调争取眼前认为最好的，而不考虑后续会有什么后果。若哲学家进餐问题用贪心算法来解决，即只要眼前有筷子能拿起就拿起的话，就会出现死锁。然而，若不仅考虑眼前的一步，而且考虑下一步，即不因为有筷子能拿起就拿起，而考虑能不能一次拿起两根筷子才做决定的话，就会避免死锁问题，这就是哲学家进餐问题的思维精髓。

大部分练习题和真题用消费者-生产者模型或读者-写者问题就能解决，但对于哲学家进餐问题和吸烟者问题仍然要熟悉。考研复习的关键在于反复多次和全面，“偷工减料”是要吃亏的。

4. 吸烟者问题

问题描述：假设一个系统有三个抽烟者进程和一个供应者进程。每个抽烟者不停地卷烟并抽掉它，但要卷起并抽掉一支烟，抽烟者需要有三种材料：烟草、纸和胶水。三个抽烟者中，第一个拥有烟草，第二个拥有纸，第三个拥有胶水。供应者进程无限地提供三种材料，供应者每次将两种材料放到桌子上，拥有剩下那种材料的抽烟者卷一根烟并抽掉它，并给供应者一个信号告诉已完成，此时供应者就会将另外两种材料放到桌上，如此重复（让三个抽烟者轮流地抽烟）。

问题分析：

- 1) **关系分析。** 供应者与三个抽烟者分别是同步关系。由于供应者无法同时满足两个或以上的抽烟者，三个抽烟者对抽烟这个动作互斥（或由三个抽烟者轮流抽烟得知）。
- 2) **整理思路。** 显然这里有 4 个进程。供应者作为生产者向三个抽烟者提供材料。
- 3) **信号量设置。** 信号量 offer1, offer2, offer3 分别表示烟草和纸组合的资源、烟草和胶水组合的资源、纸和胶水组合的资源。信号量 finish 用于互斥进行抽烟动作。

代码如下：

```

int num=0; //存储随机数
semaphore offer1=0; //定义信号量对应烟草和纸组合的资源

```

```

semaphore offer2=0;           //定义信号量对应烟草和胶水组合的资源
semaphore offer3=0;           //定义信号量对应纸和胶水组合的资源
semaphore finish=0;           //定义信号量表示抽烟是否完成
process P1(){                 //供应商
    while(1){
        num++;
        num=num%3;
        if(num==0)
            V(offerr1);          //提供烟草和纸
        else if(num==1)
            V(offerr2);          //提供烟草和胶水
        else
            V(offerr3);          //提供纸和胶水
        任意两种材料放在桌子上;
        P(finish);
    }
}
process P2(){                 //拥有烟草者
    while(1){
        P(offerr3);
        拿纸和胶水，卷成烟，抽掉;
        V(finish);
    }
}
process P3(){                 //拥有纸者
    while(1){
        P(offerr2);
        拿烟草和胶水，卷成烟，抽掉;
        V(finish);
    }
}
process P4(){                 //拥有胶水者
    while(1){
        P(offerr1);
        拿烟草和纸，卷成烟，抽掉;
        V(finish);
    }
}

```

2.3.6 本节小结

本节开头提出的问题的参考答案如下。

1) 为什么要引入进程同步的概念?

在多道程序共同执行的条件下，进程与进程是并发执行的，不同进程之间存在不同的相互制约关系。为了协调进程之间的相互制约关系，引入了进程同步的概念。

2) 不同的进程之间会存在什么关系?

进程之间存在同步与互斥的制约关系。

同步是指为完成某种任务而建立的两个或多个进程，这些进程因为需要在某些位置上协调它们的工作次序而等待、传递信息所产生的制约关系。

互斥是指当一个进程进入临界区使用临界资源时，另一个进程必须等待，当占用临界资源的进程退出临界区后，另一进程才允许去访问此临界资源。

3) 当单纯用本节介绍的方法解决这些问题时会遇到什么新的问题吗?

当两个或两个以上的进程在执行过程中,因占有一些资源而又需要对方的资源时,会因为争夺资源而造成一种互相等待的现象,若无外力作用,它们都将无法推进下去。这种现象称为死锁,具体介绍和解决方案请参考下一节。

2.3.7 本节习题精选

一、单项选择题

1. 下列对临界区的论述中,正确的是()。
 - A. 临界区是指进程中用于实现进程互斥的那段代码
 - B. 临界区是指进程中用于实现进程同步的那段代码
 - C. 临界区是指进程中用于实现进程通信的那段代码
 - D. 临界区是指进程中用于访问临界资源的那段代码
2. 不需要信号量就能实现的功能是()。

A. 进程同步	B. 进程互斥
C. 执行的前驱关系	D. 进程的并发执行
3. 若一个信号量的初值为3, 经过多次PV操作后当前值为-1, 这表示等待进入临界区的进程数是()。

A. 1	B. 2	C. 3	D. 4
------	------	------	------
4. 【2010统考真题】设与某资源关联的信号量初值为3, 当前值为1。若M表示该资源的可用个数, N表示等待该资源的进程数, 则M, N分别是()。

A. 0, 1	B. 1, 0	C. 1, 2	D. 2, 0
---------	---------	---------	---------
5. 一个正在访问临界资源的进程由于申请等待I/O操作而被中断时, 它()。
 - A. 允许其他进程进入与该进程相关的临界区
 - B. 不允许其他进程进入任何临界区
 - C. 允许其他进程抢占处理器, 但不得进入该进程的临界区
 - D. 不允许任何进程抢占处理器
6. 两个旅行社甲和乙为旅客到某航空公司订飞机票, 形成互斥资源的是()。

A. 旅行社	B. 航空公司
C. 飞机票	D. 旅行社与航空公司
7. 临界区是指并发进程访问共享变量段的()。

A. 管理信息	B. 信息存储	C. 数据	D. 代码程序
---------	---------	-------	---------
8. 以下不是同步机制应遵循的准则的是()。

A. 让权等待	B. 空闲让进	C. 忙则等待	D. 无限等待
---------	---------	---------	---------
9. 以下()不属于临界资源。

A. 打印机	B. 非共享数据	C. 共享变量	D. 共享缓冲区
--------	----------	---------	----------
10. 以下()属于临界资源。

A. 磁盘存储介质	B. 公用队列
C. 私用数据	D. 可重入的程序代码
11. 在操作系统中, 要对并发进程进行同步的原因是()。

A. 进程必须在有限的时间内完成	B. 进程具有动态性
C. 并发进程是异步的	D. 进程具有结构性

12. 进程 A 和进程 B 通过共享缓冲区协作完成数据处理，进程 A 负责产生数据并放入缓冲区，进程 B 从缓冲区读数据并输出。进程 A 和进程 B 之间的制约关系是（ ）。
- A. 互斥关系 B. 同步关系 C. 互斥和同步关系 D. 无制约关系
13. 在操作系统中，P, V 操作是一种（ ）。
- A. 机器指令 B. 系统调用命令
C. 作业控制命令 D. 低级进程通信原语
14. P 操作可能导致（ ）。
- A. 进程就绪 B. 进程结束 C. 进程阻塞 D. 新进程创建
15. 原语是（ ）。
- A. 运行在用户态的过程 B. 操作系统的内核
C. 可中断的指令序列 D. 不可分割的指令序列
16. （ ）定义了共享数据结构和各种进程在该数据结构上的全部操作。
- A. 管程 B. 类程 C. 线程 D. 程序
17. 用 V 操作唤醒一个等待进程时，被唤醒进程变为（ ）态。
- A. 运行 B. 等待 C. 就绪 D. 完成
18. 在用信号量机制实现互斥时，互斥信号量的初值为（ ）。
- A. 0 B. 1 C. 2 D. 3
19. 用 P, V 操作实现进程同步，信号量的初值为（ ）。
- A. -1 B. 0 C. 1 D. 由用户确定
20. 可以被多个进程在任意时刻共享的代码必须是（ ）。
- A. 顺序代码 B. 机器语言代码
C. 不允许任何修改的代码 D. 无转移指令代码
21. 一个进程映像由程序、数据及 PCB 组成，其中（ ）必须用可重入编码编写。
- A. PCB B. 程序 C. 数据 D. 共享程序段
22. 用来实现进程同步与互斥的 PV 操作实际上是由（ ）过程组成的。
- A. 一个可被中断的 B. 一个不可被中断的
C. 两个可被中断的 D. 两个不可被中断的
23. 有三个进程共享同一程序段，而每次只允许两个进程进入该程序段，若用 PV 操作同步机制，则信号量 S 的取值范围是（ ）。
- A. 2, 1, 0, -1 B. 3, 2, 1, 0 C. 2, 1, 0, -1, -2 D. 1, 0, -1, -2
24. 对于两个并发进程，设互斥信号量为 mutex（初值为 1），若 mutex = 0，则（ ）。
- A. 表示没有进程进入临界区
B. 表示有一个进程进入临界区
C. 表示有一个进程进入临界区，另一个进程等待进入
D. 表示有两个进程进入临界区
25. 对于两个并发进程，设互斥信号量为 mutex（初值为 1），若 mutex = -1，则（ ）。
- A. 表示没有进程进入临界区
B. 表示有一个进程进入临界区
C. 表示有一个进程进入临界区，另一个进程等待进入
D. 表示有两个进程进入临界区
26. 一个进程因在互斥信号量 mutex 上执行 V(mutex) 操作而导致唤醒另一个进程时，执行 V

- 操作后 mutex 的值为()。
- 大于 0
 - 小于 0
 - 大于等于 0
 - 小于等于 0
27. 若一个系统中共有 5 个并发进程涉及某个相同的变量 A，则变量 A 的相关临界区是由()个临界区构成的。
- 1
 - 3
 - 5
 - 6
28. 下述()选项不是管程的组成部分。
- 局限于管程的共享数据结构
 - 对管程内数据结构进行操作的一组过程
 - 管程外过程调用管程内数据结构的说明
 - 对局限于管程的数据结构设置初始值的语句
29. 以下关于管程的叙述中，错误的是()。
- 管程是进程同步工具，解决信号量机制大量同步操作分散的问题
 - 管程每次只允许一个进程进入管程
 - 管程中 signal 操作的作用和信号量机制中的 V 操作相同
 - 管程是被进程调用的，管程是语法范围，无法创建和撤销
30. 【2016 统考真题】下列关于管程的叙述中，错误的是()。
- 管程只能用于实现进程的互斥
 - 管程是由编程语言支持的进程同步机制
 - 任何时候只能有一个进程在管程中执行
 - 管程中定义的变量只能被管程内的过程访问
31. 对信号量 S 执行 P 操作后，使该进程进入资源等待队列的条件是()。
- $S.value < 0$
 - $S.value \leq 0$
 - $S.value > 0$
 - $S.value \geq 0$
32. 若系统有 n 个进程，则就绪队列中进程的个数最多有(①)个；阻塞队列中进程的个数最多有(②)个。
- $n+1$
 - n
 - $n-1$
 - 1
- $n+1$
 - n
 - $n-1$
 - 1
33. 下列关于 PV 操作的说法中，正确的是()。
- PV 操作是一种系统调用命令
 - PV 操作是一种低级进程通信原语
 - PV 操作是由一个不可被中断的过程组成
 - PV 操作是由两个不可被中断的过程组成
- I、III
 - II、IV
 - I、II、IV
 - I、IV
34. 下列关于临界区和临界资源的说法中，正确的是()。
- 银行家算法可以用来解决临界区(Critical Section)问题
 - 临界区是指进程中用于实现进程互斥的那段代码
 - 公用队列属于临界资源
 - 私用数据属于临界资源
- I、II
 - I、IV
 - 仅 III
 - 以上答案都错误
35. 有一个计数信号量 S:
- 假如若干进程对 S 进行 28 次 P 操作和 18 次 V 操作后，信号量 S 的值为 0。

2) 假如若干进程对信号量 S 进行了 15 次 P 操作和 2 次 V 操作。请问此时有多少个进程等待在信号量 S 的队列中? ()

- A. 2 B. 3 C. 5 D. 7

36. 有两个并发进程 P_1 和 P_2 , 其程序代码如下:

```
P1() {
    x=1;           //A1
    y=2;
    z=x+y;
    print z;     //A2
}
P2() {
    x=-3;         //B1
    c=x*x;
    print c;     //B2
}
```

可能打印出的 z 值有 (), 可能打印出的 c 值有 () (其中 x 为 P_1, P_2 的共享变量)。

- | | |
|---------------------------|--------------------------|
| A. $z = 1, -3; c = -1, 9$ | B. $z = -1, 3; c = 1, 9$ |
| C. $z = -1, 3, 1; c = 9$ | D. $z = 3; c = 1, 9$ |

37. 【2010 统考真题】进程 P_0 和进程 P_1 的共享变量定义及其初值为:

```
boolean flag[2];
```

```
int turn=0;
```

```
flag[0]=false; flag[1]=false;
```

若进程 P_0 和进程 P_1 访问临界资源的类 C 代码实现如下:

```
void P0() //进程 P0
{
    while(true)
    {
        flag[0]=true; turn=1;
        while(flag[1] && (turn==1));
        临界区;
        flag[0]=false;
    }
}

void P1() //进程 P1
{
    while(true)
    {
        flag[1]=true; turn=0;
        while(flag[0] && (turn==0));
        临界区;
        flag[1]=false;
    }
}
```

则并发执行进程 P_0 和进程 P_1 时产生的情况是 ()。

- | |
|------------------------------|
| A. 不能保证进程互斥进入临界区, 会出现“饥饿”现象 |
| B. 不能保证进程互斥进入临界区, 不会出现“饥饿”现象 |
| C. 能保证进程互斥进入临界区, 会出现“饥饿”现象 |
| D. 能保证进程互斥进入临界区, 不会出现“饥饿”现象 |

38. 【2016 统考真题】进程 P_1 和 P_2 均包含并发执行的线程, 部分伪代码描述如下所示。

// 进程P1	// 进程P2
int x=0;	int x=0;
Thread1()	Thread3()
{ int a;	{ int a;
a=1; x+=1;	a=x; x+=3;
}	}
Thread2()	Thread4()
{ int a;	{ int b;
a=2; x+=2;	b=x; x+=4;
}	}

下列选项中，需要互斥执行的操作是（ ）。

- | | |
|--------------------|--------------------|
| A. $a=1$ 与 $a=2$ | B. $a=x$ 与 $b=x$ |
| C. $x+=1$ 与 $x+=2$ | D. $x+=1$ 与 $x+=3$ |

39. 【2011 统考真题】有两个并发执行的进程 P_1 和进程 P_2 ，共享初值为 1 的变量 x 。 P_1 对 x 加 1， P_2 对 x 减 1。加 1 和减 1 操作的指令序列分别如下：

//加 1 操作 load R1,x //取 x 到寄存器 R1 inc R1 store x,R1 //将 R1 的内容存入 x	//减 1 操作 load R2,x //取 x 到寄存器 R2 dec R2 store x,R2 //将 R2 的内容存入 x
--	--

两个操作完成后， x 的值（ ）。

- | | |
|-----------------|---------------------|
| A. 可能为 -1 或 3 | B. 只能为 1 |
| C. 可能为 0, 1 或 2 | D. 可能为 -1, 0, 1 或 2 |

40. 【2016 统考真题】使用 TSL (Test and Set Lock) 指令实现进程互斥的伪代码如下所示。

```
do {
    ...
    while(TSL(&lock));
    critical section;
    lock=FALSE;
    ...
} while(TRUE);
```

下列与该实现机制相关的叙述中，正确的是（ ）。

- | |
|----------------------------------|
| A. 退出临界区的进程负责唤醒阻塞态进程 |
| B. 等待进入临界区的进程不会主动放弃 CPU |
| C. 上述伪代码满足“让权等待”的同步准则 |
| D. while(TSL(&lock))语句应在关中断状态下执行 |

41. 并发进程之间的关系是（ ）。

- | | |
|----------|--------------------|
| A. 无关的 | B. 相关的 |
| C. 可能相关的 | D. 可能是无关的，也可能是有交往的 |

42. 若有 4 个进程共享同一程序段，每次允许 3 个进程进入该程序段，若用 P, V 操作作为同步机制，则信号量的取值范围是（ ）。

- | | | | |
|-------------------|--------------------|-------------------|--------------------|
| A. 4, 3, 2, 1, -1 | B. 2, 1, 0, -1, -2 | C. 3, 2, 1, 0, -1 | D. 2, 1, 0, -2, -3 |
|-------------------|--------------------|-------------------|--------------------|

43. 在 9 个生产者、6 个消费者共享容量为 8 的缓冲器的生产者-消费者问题中，互斥使用缓冲器的信号量初始值为（ ）。

- | | | | |
|------|------|------|------|
| A. 1 | B. 6 | C. 8 | D. 9 |
|------|------|------|------|

44. 信箱通信是一种（ ）通信方式。

- | | | | |
|---------|---------|---------|--------|
| A. 直接通信 | B. 间接通信 | C. 低级通信 | D. 信号量 |
|---------|---------|---------|--------|

45. 有两个优先级相同的并发程序 P_1 和 P_2 ，它们的执行过程如下所示。假设当前信号量 $s1 = 0$, $s2 = 0$ 。当前的 $z = 2$ ，进程运行结束后， x , y 和 z 的值分别是（ ）。

进程 P_1	进程 P_2
----------	----------

...	...
$y:=1;$	$x:=1$
$y:=y+2;$	$x:=x+1;$
$z:=y+1;$	$P(s1);$
$V(s1);$	$x:=x+y;$

- | | |
|---------|---------|
| P(s2); | z:=x+z; |
| y:=z+y; | V(s2); |
| ... | ... |
- A. 5, 9, 9 B. 5, 9, 4 C. 5, 12, 9 D. 5, 12, 4

46. 【2018 统考真题】属于同一进程的两个线程 thread1 和 thread2 并发执行，共享初值为 0 的全局变量 x。thread1 和 thread2 实现对全局变量 x 加 1 的机器级代码描述如下。

thread1	thread2
mov R1, x // (x)→R1	mov R2, x // (x)→R2
inc R1 // (R1)+1→R1	inc R2 // (R2)+1→R2
mov x, R1 // (R1)→x	mov x, R2 // (R2)→x

在所有可能的指令执行序列中，使 x 的值为 2 的序列个数是（）。

- A. 1 B. 2 C. 3 D. 4

47. 【2018 统考真题】若 x 是管程内的条件变量，则当进程执行 x.wait()时所做的工作是（）。

- A. 实现对变量 x 的互斥访问
- B. 唤醒一个在 x 上阻塞的进程
- C. 根据 x 的值判断该进程是否进入阻塞态
- D. 阻塞该进程，并将之插入 x 的阻塞队列中

48. 【2018 统考真题】在下列同步机制中，可以实现让权等待的是（）。

- A. Peterson 方法 B. swap 指令 C. 信号量方法 D. TestAndSet 指令

49. 【2020 统考真题】下列准则中，实现临界区互斥机制必须遵循的是（）。

- I. 两个进程不能同时进入临界区
- II. 允许进程访问空闲的临界资源
- III. 进程等待进入临界区的时间是有限的
- IV. 不能进入临界区的执行态进程立即放弃 CPU

- A. 仅 I、IV B. 仅 II、III C. 仅 I、II、III D. 仅 I、III、IV

二、综合应用题

1. 何谓管程？管程由几部分组成？说明引入管程的必要性。
2. 进程之间存在哪几种制约关系？各是什么原因引起的？以下活动各属于哪种制约关系？
 - 1) 若干学生去图书馆借书。
 - 2) 两队进行篮球比赛。
 - 3) 流水线生产的各道工序。
 - 4) 商品生产和消费。
3. 【2009 统考真题】三个进程 P₁, P₂, P₃ 互斥使用一个包含 N (N > 0) 个单元的缓冲区。P₁ 每次用 produce() 生成一个正整数并用 put() 送入缓冲区某一空单元；P₂ 每次用 getodd() 从该缓冲区中取出一个奇数并用 countodd() 统计奇数个数；P₃ 每次用 geteven() 从该缓冲区中取出一个偶数并用 counteven() 统计偶数个数。请用信号量机制实现这三个进程的同步与互斥活动，并说明所定义的信号量的含义（要求用伪代码描述）。
4. 下面是两个并发执行的进程，它们能正确运行吗？若不能请举例说明并改正。

```
int x;
process_P1{
    int y,z;
}
process_P2{
    int t,u;
}
```

```

x=1;           x=0;
y=0;           t=0;
if(x>=1)      if(x<=1)
    y=y+1;      t=t+2;
z=y;           u=t;
}
}

```

5. 有两个并发进程 P₁, P₂, 其程序代码如下:

```

P1() {
    x=1;
    y=2;
    if(x>0)
        z=x+y;
    else
        z=x*y;
    print z;
}

P2() {
    x=-1;
    a=x+3;
    x=a+x;
    b=a+x;
    c=b*b;
    print c;
}

```

1) 可能打印出的 z 值有多少? (假设每条赋值语句是一个原子操作)

2) 可能打印出的 c 值有多少? (其中 x 为 P₁ 和 P₂ 的共享变量)

6. 在一个仓库中可以存放 A 和 B 两种产品, 要求:

① 每次只能存入一种产品。

② A 产品数量 - B 产品数量 < M.

③ B 产品数量 - A 产品数量 < N.

其中, M, N 是正整数, 试用 P 操作、V 操作描述产品 A 与产品 B 的入库过程。

7. 面包师有很多面包, 由 n 名销售人员推销。每名顾客进店后取一个号, 并且等待叫号, 当一名销售人员空闲时, 就叫下一个号。试设计一个使销售人员和顾客同步的算法。

8. 【2013 统考真题】某博物馆最多可容纳 500 人同时参观, 有一个出入口, 该出入口一次仅允许一人通过。参观者的活动描述如下:

```

cobegin
    参观者进程 i:
    {
        ...
        进门。
        ...
        参观;
        ...
        出门;
        ...
    }
coend

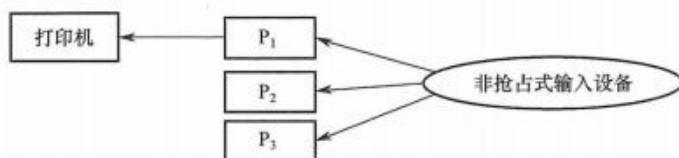
```

请添加必要的信号量和 P, V [或 wait(), signal()] 操作, 以实现上述过程中的互斥与同步。要求写出完整的过程, 说明信号量的含义并赋初值。

9. 某工厂有两个生产车间和一个装配车间, 两个生产车间分别生产 A, B 两种零件, 装配车间的任务是把 A, B 两种零件组装成产品。两个生产车间每生产一个零件后, 都要分别把它们送到专配车间的货架 F₁, F₂ 上。F₁ 存放零件 A, F₂ 存放零件 B, F₁ 和 F₂ 的容量均可存放 10 个零件。装配工人每次从货架上取一个零件 A 和一个零件 B 后组装成产品。请

用 P, V 操作进行正确管理。

10. 某寺庙有小和尚、老和尚若干，有一水缸，由小和尚提水入缸供老和尚饮用。水缸可容 10 桶水，水取自同一井中。水井径窄，每次只能容一个桶取水。水桶总数为 3 个。每次入缸取水仅为 1 桶水，且不可同时进行。试给出有关从缸取水、入水的算法描述。
11. 如下图所示，三个合作进程 P_1, P_2, P_3 ，它们都需要通过同一设备输入各自的数据 a, b, c ，该输入设备必须互斥地使用，而且其第一个数据必须由 P_1 进程读取，第二个数据必须由 P_2 进程读取，第三个数据必须由 P_3 进程读取。然后，三个进程分别对输入数据进行下列计算：



$$P_1: x = a + b;$$

$$P_2: y = a * b;$$

$$P_3: z = y + c - a;$$

最后， P_1 进程通过所连接的打印机将计算结果 x, y, z 的值打印出来。请用信号量实现它们的同步。

12. 【2011 统考真题】某银行提供 1 个服务窗口和 10 个供顾客等待的座位。顾客到达银行时，若有空座位，则到取号机上领取一个号，等待叫号。取号机每次仅允许一位顾客使用。当营业员空闲时，通过叫号选取一位顾客，并为其服务。顾客和营业员的活动过程描述如下：

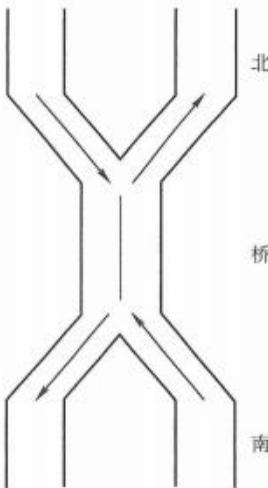
```

cobegin
{
    process 顾客 i
    {
        从取号机获取一个号码;
        等待叫号;
        获取服务;
    }
    process 营业员
    {
        While (TRUE)
        {
            叫号;
            为客户服务;
        }
    }
} coend
  
```

请添加必要的信号量和 P, V [或 wait(), signal()] 操作，实现上述过程中的互斥与同步。要求写出完整的过程，说明信号量的含义并赋初值。

13. 有桥如下图所示。车流方向如箭头所示。回答如下问题：

- 1) 假设桥上每次只能有一辆车行驶，试用信号灯的 P, V 操作实现交通管理。
- 2) 假设桥上不允许两车交会，但允许同方向多辆车一次通过（即桥上可有多辆同方向行驶的车）。试用信号灯的 P, V 操作实现桥上的交通管理。



14. 假设有两个线程（编号为 0 和 1）需要去访问同一个共享资源，为避免竞争状态的问题，我们必须实现一种互斥机制，使得在任何时候只能有一个线程访问这个资源。假设有如下一段代码：

```
bool flag[2];           //flag 数组，初始化为 FALSE
Enter_Critical_Section(int my_thread_id,int other_thread_id){
    while(flag[other_thread_id]==TRUE);           //空循环语句
    flag[my_thread_id]=TRUE;
}
Exit_Critical_Section(int my_thread_id,int other_thread_id){
    flag[my_thread_id]=FALSE;
}
```

当一个线程想要访问临界资源时，就调用上述的这两个函数。例如，线程 0 的代码可能是这样的：

```
Enter_Critical_Section(0,1);
使用这个资源;
Exit_Critical_Section(0,1);
做其他的事情;
试问:
```

1) 以上的这种机制能够实现资源互斥访问吗？为什么？

2) 若把 Enter_Critical_Section() 函数中的两条语句互换一下位置，结果会如何？

15. 设自行车生产线上有一个箱子，其中有 N 个位置 ($N \geq 3$)，每个位置可存放一个车架或一个车轮；又设有 3 名工人，其活动分别为：

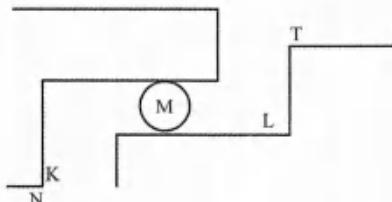
工人 1 活动:	工人 2 活动:	工人 3 活动:
do{ 加工一个车架; 车架放入箱中; }while(1)	do{ 加工一个车轮; 车轮放入箱中; }while(1)	do{ 箱中取一个车架; 箱中取二个车轮; 组装为一台车; }while(1)

试分别用信号量与 PV 操作实现三名工人的合作，要求解中不含死锁。

16. 设 P, Q, R 共享一个缓冲区，P, Q 构成一对生产者-消费者，R 既为生产者又为消费者。使用 P, V 操作实现其同步。
17. 理发店里有一位理发师、一把理发椅和 n 把供等候理发的顾客坐的椅子。若没有顾客，理发师便在理发椅上睡觉，一位顾客到来时，顾客必须叫醒理发师，若理发师正在理发

时又有顾客来到，若有空椅子可坐，则坐下来等待，否则就离开。试用 P, V 操作实现，并说明信号量的定义和初值。

18. 假设一个录像厅有 1, 2, 3 三种不同的录像片可由观众选择放映，录像厅的放映规则如下：
 - 1) 任一时刻最多只能放映一种录像片，正在放映的录像片是自动循环放映的，最后一名观众主动离开时结束当前录像片的放映。
 - 2) 选择当前正在放映的录像片的观众可立即进入，允许同时有多位选择同一种录像片的观众同时观看，同时观看的观众数量不受限制。
 - 3) 等待观看其他录像片的观众按到达顺序排队，当一种新的录像片开始放映时，所有等待观看该录像片的观众可依次序进入录像厅同时观看。用一个进程代表一个观众，要求：用信号量方法 PV 操作实现，并给出信号量定义和初始值。
19. 南开大学和天津大学之间有一条弯曲的路，每次只允许一辆自行车通过，但中间有小的安全岛 M（同时允许两辆车通过），可供已进入两端的两辆小车错车，如下图所示。设计算法并使用 P, V 操作实现。



20. 【2014 统考真题】系统中有多个生产者进程和多个消费者进程，共享一个能存放 1000 件产品的环形缓冲区（初始为空）。缓冲区未满时，生产者进程可以放入其生产的一件产品，否则等待；缓冲区未空时，消费者进程可从缓冲区取走一件产品，否则等待。要求一个消费者进程从缓冲区连续取出 10 件产品后，其他消费者进程才可以取产品。请使用信号量 P, V (wait(), signal()) 操作实现进程间的互斥与同步，要求写出完整的过程，并说明所用信号量的含义和初值。
21. 设公共汽车上驾驶员和售票员的活动分别如下图所示。驾驶员的活动：启动车辆，正常行车，到站停车；售票员的活动：关车门，售票，开车门。在汽车不断地到站、停车、行驶的过程中，这两个活动有什么同步关系？用信号量和 P, V 操作实现它们的同步。



22. 【2015 统考真题】有 A, B 两人通过信箱进行辩论，每个人都从自己的信箱中取得对方的问题。将答案和向对方提出的新问题组成一个邮件放入对方的邮箱中。假设 A 的信箱最多放 M 个邮件，B 的信箱最多放 N 个邮件。初始时 A 的信箱中有 x 个邮件 ($0 < x < M$)，B 的信箱中有 y 个邮件 ($0 < y < N$)。辩论者每取出一个邮件，邮件数减 1。A 和 B 两人的操作过程描述如下：

CoBegin

<pre>A{ while(TRUE){ 从 A 的信箱中取出一个邮件; 回答问题并提出一个新问题; 将新邮件放入 B 的信箱; } }</pre>	<pre>B{ while(TRUE){ 从 B 的信箱中取出一个邮件; 回答问题并提出一个新问题; 将新邮件放入 A 的信箱; } }</pre>
--	--

CoEnd

当信箱不为空时，辩论者才能从信箱中取邮件，否则等待。当信箱不满时，辩论者才能将新邮件放入信箱，否则等待。请添加必要的信号量和 P, V [或 wait(), signal()] 操作，以实现上述过程的同步。要求写出完整的过程，并说明信号量的含义和初值。

23. 【2017 统考真题】某进程中 3 个并发执行的线程 thread1, thread2 和 thread3，其伪代码如下所示。

// 复数的结构类型定义	thread1	thread3
<pre>typedef struct { float a; float b; } cnum;</pre>	<pre>cnum w; w = add(x, y); ... }</pre>	<pre>cnum w; w.a = 1; w.b = 1; z = add(z, w); y = add(y, w); ... }</pre>
<pre>cnum x, y, z; // 全局变量</pre>		
<pre>// 计算两个复数之和</pre>	<pre>thread2</pre>	<pre>thread2</pre>
<pre>cnum add(cnum p, cnum q) { cnum s; s.a = p.a+q.a; s.b = p.b+q.b; return s; }</pre>	<pre>cnum w; w = add(y, z); ... }</pre>	

请添加必要的信号量和 P, V [或 wait(), signal()] 操作，要求确保线程互斥访问临界资源，并且最大限度地并发执行。

24. 【2019 统考真题】有 n ($n \geq 3$) 名哲学家围坐在一张圆桌边，每名哲学家交替地就餐和思考。在圆桌中心有 m ($m \geq 1$) 个碗，每两名哲学家之间有一根筷子。每名哲学家必须取到一个碗和两侧的筷子后，才能就餐，进餐完毕，将碗和筷子放回原位，并继续思考。为使尽可能多的哲学家同时就餐，且防止出现死锁现象，请使用信号量的 P, V 操作 [wait(), signal() 操作] 描述上述过程中的互斥与同步，并说明所用信号量及初值的含义。
25. 【2020 统考真题】现有 5 个操作 A、B、C、D 和 E，操作 C 必须在 A 和 B 完成后执行，操作 E 必须在 C 和 D 完成后执行，请使用信号量的 wait()、signal() 操作 (P, V 操作) 描述上述操作之间的同步关系，并说明所用信号量及其初值。

2.3.8 答案与解析

一、单项选择题

1. D

多个进程可以共享系统中的资源，一次仅允许一个进程使用的资源称为临界资源。访问临界资源的那段代码称为临界区。

2. D

在多道程序技术中，信号量机制是一种有效实现进程同步和互斥的工具。进程执行的前趋关系实质上是指进程的同步关系。除此之外，只有进程的并发执行不需要信号量来控制，因此正确答案为 D。

3. A

信号量是一个特殊的整型变量，只有初始化和 PV 操作才能改变其值。通常，信号量分为互斥量和资源量，互斥量的初值一般为 1，表示临界区只允许一个进程进入，从而实现互斥。当互斥量等于 0 时，表示临界区已有一个进程进入，临界区外尚无进程等待；当互斥量小于 0 时，表示临界区中有一个进程，互斥量的绝对值表示在临界区外等待进入的进程数。同理，资源信号量的初值可以是任意整数，表示可用的资源数，当资源量小于 0 时，表示所有资源已全部用完，而且还有进程正在等待使用该资源，等待的进程数就是资源量的绝对值。

4. B

信号量表示相关资源的当前可用数量。当信号量 $K > 0$ 时，表示还有 K 个相关资源可用，所以该资源的可用个数是 1。而当信号量 $K < 0$ 时，表示有 $|K|$ 个进程在等待该资源。由于资源有剩余，可见没有其他进程等待使用该资源，因此进程数为 0。

5. C

进程进入临界区必须满足互斥条件，当进程进入临界区但尚未离开时就被迫进入阻塞是可以的，系统中经常出现这样的情形。在此状态下，只要其他进程在运行过程中不寻求进入该进程的临界区，就应允许其运行，即分配 CPU。该进程所锁定的临界区是不允许其他进程访问的，其他进程若要访问，必定会在临界区的“锁”上阻塞，期待该进程下次运行时可以离开并将临界区交给它。所以正确答案为 C。

6. C

一张飞机票不能售给不同的旅客，因此飞机票是互斥资源，其他因素只是为完成飞机票订票的中间过程，与互斥资源无关。

7. D

所谓临界区，并不是指临界资源，如共享的数据、代码或硬件设备等，而是指访问临界资源的那段代码程序，如 P/V 操作、加减锁等。操作系统访问临界资源时，关心的是临界区的操作过程，具体对临界资源做何操作是应用程序的事情，操作系统并不关心。

8. D

同步机制的 4 个准则是空闲让进、忙则等待、让权等待和有限等待。

9. B

临界资源是互斥共享资源，非共享数据不属于临界资源。打印机、共享变量和共享缓冲区都只允许一次供一个进程使用。

10. B

临界资源与共享资源的区别在于，在一段时间内能否允许被多个进程访问（并发使用），显

然磁盘属于共享设备。公用队列可供多个进程使用，但一次只可供一个进程使用，试想若多个进程同时使用公用队列，势必造成队列中的数据混乱而无法使用。私用数据仅供一个进程使用，不存在临界区问题，可重入的程序代码一次可供多个进程使用。

11. C

进程同步是指进程之间一种直接的协同工作关系，这些进程的并发是异步的，它们相互合作，共同完成一项任务。

12. C

并发进程因为共享资源而产生相互之间的制约关系，可以分为两类：① 互斥关系，指进程之间因相互竞争使用独占型资源（互斥资源）所产生的制约关系；② 同步关系，指进程之间为协同工作需要交换信息、相互等待而产生的制约关系。本题中两个进程之间的制约关系是同步关系，进程 B 必须在进程 A 将数据放入缓冲区后才能从缓冲区中读出数据。此外，共享的缓冲区一定是互斥访问的，所以它们也具有互斥关系。

13. D

P、V 操作是一种低级的进程通信原语，它是不能被中断的。

14. C

P 操作即 wait 操作，表示等待某种资源直到可用。若这种资源暂时不可用，则进程进入阻塞态。注意，执行 P 操作时的进程处于运行态。

15. D

原语（Primitive/Atomic Action），顾名思义，就是原子性的、不可分割的操作。其严格定义为：由若干机器指令构成的完成某种特定功能的一段程序，其执行必须是连续的，在执行过程中不允许被中断。

16. A

管程定义了一个数据结构和能为并发进程所执行（在该数据结构上）的一组操作，这组操作能同步进程并改变管程中的数据。

17. C

只有就绪进程能获得处理器资源，被唤醒的进程并不能直接转换为运行态。

18. B

互斥信号量的初值为 1，P 操作成功则将其减 1，禁止其他进程进入；V 操作成功则将其加 1，允许等待队列中的一个进程进入。

19. D

与互斥信号量初值一般为 1 时不同，用 P、V 操作实现进程同步，信号量的初值应根据具体情况来确定。若期望的消息尚未产生，则对应的初值应为 0；若期望的消息已存在，则信号量的初值应设为一个非 0 的正整数。

20. C

若代码可被多个进程在任意时刻共享，则要求任一个进程在调用此段代码时都以同样的方式运行；而且进程在运行过程中被中断后再继续执行，其执行结果不受影响。这必然要求代码不能被任何进程修改，否则无法满足共享的要求。这样的代码就是可重入代码，也称纯代码，即允许多个进程同时访问的代码。

21. D

共享程序段可能同时被多个进程使用，所以必须可重入编码，否则无法实现共享的功能。

22. D

P 操作和 V 操作都属于原语操作，不可被中断。

23. A

因为每次允许两个进程进入该程序段，信号量最大值取 2。至多有三个进程申请，则信号量最小为 -1，所以信号量可以取 2, 1, 0, -1。

24. B

临界区不允许两个进程同时进入，D 选项明显错误。mutex 的初值为 1，表示允许一个进程进入临界区，当有一个进程进入临界区且没有进程等待进入时，mutex 减 1，变为 0。

25. C

当有一个进程进入临界区且有另一个进程等待进入临界区时， $\text{mutex} = -1$ 。 mutex 小于 0 时，其绝对值等于等待进入临界区的进程数。

26. D

由题意可知，系统原来存在等待进入临界区的进程， mutex 小于等于 -1，因此在执行 V(mutex) 操作后， mutex 的值小于等于 0。

27. C

这里的临界区是指访问临界资源 A 的那段代码（临界区的定义）。那么 5 个并发进程共有 5 个操作共享变量 A 的代码段。

28. C

管程由局限于管程的共享变量说明、对管程内的数据结构进行操作的一组过程及对局限于管程的数据设置初始值的语句组成。

29. C

管程的 signal 操作与信号量机制中的 V 操作不同，信号量机制中的 V 操作一定会改变信号量的值 $S = S + 1$ 。而管程中的 signal 操作是针对某个条件变量的，若不存在因该条件而阻塞的进程，则 signal 不会产生任何影响。

30. A

管程是由一组数据及定义在这组数据之上的对这组数据的操作组成的软件模块，这组操作能初始化并改变管程中的数据和同步进程。管程不仅能实现进程间的互斥，而且能实现进程间的同步，因此 A 错误、B 正确；管程具有如下特性：① 局部于管程的数据只能被局部于管程内的过程所访问；② 一个进程只有通过调用管程内的过程才能进入管程访问共享数据；③ 每次仅允许一个进程在管程内执行某个内部过程，因此 C 和 D 正确。

31. A

参见记录型信号量的解析。此处极易出 S.value 的物理概念题，现总结如下：

S.value > 0，表示某类可用资源的数量。每次 P 操作，意味着请求分配一个单位的资源。

S.value <= 0，表示某类资源已经没有，或者说还有因请求该资源而被阻塞的进程。

S.value <= 0 时的绝对值，表示等待进程数目。

一定要看清题目中的陈述是执行 P 操作前还是执行 P 操作后。

32. ①C ②B

① 系统中有 n 个进程，其中至少有一个进程正在执行（处理器至少有一个），因此就绪队列中的进程个数最多有 $n - 1$ 个。B 选项容易被错选，以为出现了处理器为空、就绪队列全满的情况，实际调度无此状态。

注意：系统中有 n 个进程，其中至少有一个进程正在执行（处理器至少有一个），其实这句话对于一般情况是错误的，但我们只需考虑就绪队列中进程最多这种特殊情况即可。

② 此题 C 选项容易被错选，阻塞队列有 $n - 1$ 个进程这种情况是可能发生的，但不是最多的情况。可能不少读者会忽视死锁的情况，死锁就是 n 个进程都被阻塞，所以最多可以有 n 个进程在阻塞队列。

33. B

PV 操作是一种低级的进程通信原语，不是系统调用，因此 II 正确；P 操作和 V 操作都属于原子操作，所以 PV 操作由两个不可被中断的过程组成，因此 IV 正确。

34. C

临界资源是指每次仅允许一个进程访问的资源。每个进程中访问临界资源的那段代码称为临界区。I 错误，银行家算法是避免死锁的算法。II 错误，每个进程中访问临界资源的那段代码称为临界区。III 正确，公用队列可供多个进程使用，但一次只可供一个程序使用。IV 错误，私用数据仅供一个进程使用，不存在临界区问题。综上分析，正确答案为 C。

35. B

对 S 进行了 28 次 P 操作和 18 次 V 操作，即 $S - 28 + 18 = 0$ ，得信号量的初值为 10；然后，对信号量 S 进行了 15 次 P 操作和 2 次 V 操作，即 $S - 15 + 2 = 10 - 15 + 2 = -3$ ， S 信号量的负值的绝对值表示等待队列中的进程数。所以有 3 个进程等待在信号量 S 的队列中。

36. B

本题的关键是，输出语句 A2, B2 中读取的 x 的值不同，由于 A1, B1 执行有先后问题，使得在执行 A2, B2 前， x 的可能取值有两个，即 1, -3；这样，输出 z 的值可能是 $1 + 2 = 3$ 或 $(-3) + 2 = -1$ ；输出 c 的值可能是 $1 \times 1 = 1$ 或 $(-3) \times (-3) = 9$ 。

37. D

这是 Peterson 算法的实际实现，保证进入临界区的进程合理安全。

该算法为了防止两个进程为进入临界区而无限期等待，设置了变量 turn，表示不允许进入临界区的编号，每个进程在先设置自己的标志后再设置 turn 标志，不允许另一个进程进入。这时，再同时检测另一个进程状态标志和不允许进入表示，就可保证当两个进程同时要求进入临界区时只允许一个进程进入临界区。保存的是较晚的一次赋值，因此较晚的进程等待，较早的进程进入。先到先入，后到等待，从而完成临界区访问的要求。

其实这里可想象为两个人进门，每个人进门前都会和对方客套一句“你走先”。若进门时没别人，就当和空气说句废话，然后大步登门入室；若两人同时进门，就互相先请，但各自只客套一次，所以先客套的人请完对方，就等着对方请自己，然后光明正大地进门。

38. C

P_1 中对 a 进行赋值，并不影响最终的结果，因此 $a = 1$ 与 $a = 2$ 不需要互斥执行； $a = x$ 与 $b = x$ 执行先后不影响 a 与 b 的结果，无须互斥执行； $x += 1$ 与 $x += 2$ 执行先后会影响 x 的结果，需要互斥执行； P_1 中的 x 和 P_2 中的 x 是不同范围中的 x ，互不影响，不需要互斥执行。

39. C

将 P_1 中的 3 条语句依次编号为 1, 2, 3，将 P_2 中的 3 条语句依次编号为 4, 5, 6，则依次执行 1, 2, 3, 4, 5, 6 得结果 1，依次执行 1, 2, 4, 5, 6, 3 得结果 2，依次执行 4, 5, 1, 2, 3, 6 得结果 0。结果 -1 不可能得出。

40. B

当进程退出临界区时置 lock 为 FALSE，会负责唤醒处于就绪态的进程，选项 A 错误。等待进入临界区的进程会一直停留在执行 `while(TSL(&lock))` 的循环中，不会主动放弃 CPU，选项 B 正确。让权等待，即进程不能进入临界区时，应立即释放处理器，防止进程忙等待。通过 B 选项

的分析发现，上述伪代码并不满足“让权等待”的同步准则，选项 C 错误。while(TSL(&lock))在关中断状态下执行时，若 TSL(&lock)一直为 true，不再开中断，则系统可能会因此终止，选项 D 错误。

41. D

并发进程之间的关系没有必然的要求，只有执行时间上的偶然重合，可能无关也可能有交往。

42. C

由于每次允许三个进程进入该程序段，因此可能出现的情况是：没有进程进入，有一个进程进入，有两个进程进入，有三个进程进入，有三个进程进入并有一个在等待进入，因此这 5 种情况对应的信号量值为 3, 2, 1, 0, -1。

43. A

所谓互斥使用某临界资源，是指在同一时间段只允许一个进程使用此资源，所以互斥信号量的初值都为 1。

44. B

信箱通信是一种间接通信方式。

45. C

由于进程并发，因此进程的执行具有不确定性，在 P₁, P₂ 执行到第一个 P, V 操作前，应该是相互无关的。现在考虑第一个对 s1 的 P, V 操作，由于进程 P₂ 是 P(s1) 操作，所以它必须等待 P₁ 执行完 V(s1) 操作后才可继续运行，此时的 x, y, z 值分别是 2, 3, 4，当进程 P₁ 执行完 V(s1) 后便在 P(s2) 上阻塞，此时 P₂ 可以运行直到 V(s2)，此时的 x, y, z 值分别是 5, 3, 9，进程 P₁ 继续运行到结束，最终的 x, y, z 值分别为 5, 12, 9。

46. B

仔细阅读两个线程代码可知，thread1 和 thread2 均是对 x 进行加 1 操作，x 的初始值为 0，若要使得最终 x = 2，只能先执行完 thread1 再执行 thread2，或先执行完 thread2 再执行 thread1，因此仅有 2 种可能，选 B。

47. D

“条件变量”是管程内部说明和使用的一种特殊变量，其作用类似于信号量机制中的“信号量”，都用于实现进程同步。需要注意的是，在同一时刻，管程中只能有一个进程在执行。若进程 A 执行了 x.wait() 操作，则该进程会阻塞，并挂到条件变量 x 对应的阻塞队列上。这样，管程的使用权被释放，就可以有另一个进程进入管程。若进程 B 执行了 x.signal() 操作，则会唤醒 x 对应的阻塞队列的队首进程。在 Pascal 语言的管程中，规定只有一个进程要离开管程时才能调用 signal() 操作。

48. C

硬件方法实现进程同步时不能实现让权等待，因此 B、D 错误；Peterson 算法满足有限等待但不满足让权等待，因此 A 错误；记录型信号量由于引入阻塞机制，消除了不让权等待的情况，因此 C 正确。

49. C

实现临界区互斥需满足多个准则。“忙则等待”准则，即两个进程不能同时访问临界区，I 正确。“空闲让进”准则，若临界区空闲，则允许其他进程访问，II 正确。“有限等待”准则，即进程应该在有限时间内访问临界区，III 正确。I、II 和 III 是互斥机制必须遵循的原则。IV 是“让权等待”准则，不一定非得实现，如皮特森算法。

二、综合应用题

1. 解答：

当共享资源用共享数据结构表示时，资源管理程序可用对该数据结构进行操作的一组过程来表示，如资源的请求和释放过程 `request` 和 `release`。把这样一组相关的数据结构和过程一并归为管程。Hansan 为管程所下的定义是：“一个管程定义了一个数据结构和能为并发进程所执行（在该数据结构上）的一组操作，这组操作能同步进程和改变管程中的数据。”由定义可知，管程由三部分组成：

- 1) 局部于管程的共享变量说明。
- 2) 该数据结构进行操作的一组过程。
- 3) 对局部于管程的数据设置初始值的语句；此外，还需为该管程赋予一个名字。

管程的引入是为了解决临界区分散所带来的管理和控制问题。在没有管程之前，对临界区的访问分散在各个进程之中，不易发现和纠正分散在用户程序中的不正确使用 P, V 操作等问题。管程将这些分散在各进程中的临界区集中起来，并加以控制和管理，管程一次只允许一个进程进入管程内，从而既便于系统管理共享资源，又能保证互斥。

2. 解答：

进程之间存在两种制约关系，即同步和互斥。

同步是由于并发进程之间需要协调完成同一个任务时引起的一种关系，是一个进程等待另一个进程向它直接发送消息或数据时的一种制约关系。

互斥是由并发进程之间竞争系统的临界资源引起的，是一个进程等待另一个进程已经占有的必须互斥使用的资源时的一种制约关系。

- 1) 是互斥关系，同一本书只能被一名学生借阅，或任何时刻只能有一名学生借阅一本书。
- 2) 是互斥关系，篮球是互斥资源，只可被一个队伍获得。
- 3) 是同步关系，一个工序完成后开始下一个工序。
- 4) 是同步关系，生产商品后才能消费。

3. 解答：

互斥资源：缓冲区只能互斥访问，因此设置互斥信号量 `mutex`。

同步问题：`P1`, `P2` 因为奇数的放置与取用而同步，设同步信号量 `odd`；`P1`, `P3` 因为偶数的放置与取用而同步，设置同步信号量 `even`；`P1`, `P2`, `P3` 因为共享缓冲区，设同步信号量 `empty`，初值为 `N`。程序如下：

```

semaphore mutex=1;           //缓冲区操作互斥信号量
semaphore odd=0,even=0;       //奇数、偶数进程的同步信号量
semaphore empty=N;          //空缓冲区单元个数信号量
cobegin{
    Process P1()
    while(True)
    {
        x=produce();           //生成一个数
        P(empty);              //判断缓冲区是否有空单元
        P(mutex);               //缓冲区是否被占用
        Put();                  //释放缓冲区
        V(mutex);               //释放缓冲区
        if(x%2==0)
            V(even);             //若是偶数，向 P3 发出信号
        else
    }
}

```

```

        V(odd);           //若是奇数, 向 P2 发出信号
    }
Process P2()
while(True)
{
    P(odd);           //收到 P1 发来的信号, 已产生一个奇数
    P(mutex);
    getodd();
    V(mutex);
    V(empty);
    countodd();
}
Process P3()
while(True)
{
    P(even);          //收到 P1 发来的信号, 已产生一个偶数
    P(mutex);
    geteven();
    V(mutex);
    V(empty);
    counteven();
}
}coend

```

4. 解答:

P_1 和 P_2 两个并发进程的执行结果是不确定的, 它们都对同一变量 X 进程操作, X 是一个临界资源, 而没有进行保护。例如:

- 1) 若先执行完 P_1 再执行 P_2 , 结果是 $x=0, y=1, z=1, t=2, u=2$ 。
- 2) 若先执行 P_1 到 “ $x=1$ ”, 然后一个中断去执行完 P_2 , 再一个中断回来执行完 P_1 , 结果是 $x=0, y=0, z=0, t=2, u=2$ 。

显然, 两次执行结果不同, 所以这两个并发进程不能正确运行。可将这个程序改为:

```

int x;
semaphore S=1;           //访问 x 的互斥信号量
process_P1{               process_P2{
    int y,z;
    P(S);
    x=1;
    y=0;
    if(x>=1)
        y=y + 1;
    V(S);
    z=y;
}
}

```

5. 解答:

- 1) z 的值有: $-2, 1, 2, 3, 5, 7$ 。
- 2) c 的值有: $9, 25, 81$ 。

6. 解答:

使用信号量 $mutex$ 控制两个进程互斥访问临界资源 (仓库), 使用同步信号量 S_a 和 S_b (分别代表产品 A 与 B 的还可容纳的数量差, 以及产品 B 与 A 的还可容纳的数量差) 满足条件 2 和条

件3。代码如下：

```

Semaphore Sa=M-1,Sb=N-1;
Semaphore mutex=1;           //访问仓库的互斥信号量
process_A(){
    while(1){
        P(Sa);
        P(mutex);
        A产品入库;
        V(mutex);
        V(Sb);
    }
}
process_B(){
    while(1){
        P(Sb);
        P(mutex);
        B产品入库;
        V(mutex);
        V(Sa);
    }
}

```

7. 解答：

顾客进店后按序取号，并等待叫号；销售人员空闲后也按序叫号，并销售面包。因此同步算法只要对顾客取号和销售人员叫号进行合理同步即可。我们使用两个变量 i 和 j 分别记录当前的取号值和叫号值，并各自使用一个互斥信号量用于对 i 和 j 进行访问和修改。

```

int i=0,j=0;
semaphore mutex_i=1,mutex_j=1;
Consumer(){                         //顾客
    进入面包店;
    P(mutex_i);                     //互斥访问 i
    取号 i;
    i++;
    V(mutex_i);                   //释放对 i 的访问
    等待叫号 i 并购买面包;
}
Seller(){                           //销售人员
    while(1){
        P(mutex_j);               //互斥访问 j
        if(j<=i){                //号 j 已有顾客取走并等待
            叫号 j;
            j++;
            V(mutex_j);           //释放对 j 的访问
            销售面包;
        }
        else{                     //暂时没有顾客在等待
            V(mutex_j);           //释放对 j 的访问
            休息片刻;
        }
    }
}

```

8. 解答：

出入口一次仅允许一个人通过，设置互斥信号量 mutex，初值为 1。博物馆最多可同时容纳 500 人，因此设置信号量 empty，初值为 500。

```

Semaphore empty=500;                 //博物馆可以容纳的最多人数
Semaphore mutex=1;                  //用于出入口资源的控制
cobegin
    参观者进程 i:
    {
        ...
    }

```

```

    P(empty);
    //可容纳人数减 1
    P(mutex);
    进门;
    V(mutex);
    参观;
    P(mutex);
    //互斥使用门
    出门;
    V(mutex);
    V(empty);
    //可容纳人数增 1
    ...
}
coend

```

9. 解答:

本题是生产者-消费者问题的变体，生产者“车间 A”和消费者“装配车间”共享缓冲区“货架 F1”；生产者“车间 B”和消费者“装配车间”共享缓冲区“货架 F2”。因此，可为它们设置 6 个信号量：empty1 对应货架 F1 上的空闲空间，初值为 10；full1 对应货架 F1 上面的 A 产品，初值为 0；empty2 对应货架 F2 上的空闲空间，初值为 10；full2 对应货架 F2 上面的 B 产品，初值为 0；mutex1 用于互斥地访问货架 F1，初值为 1；mutex2 用于互斥地访问货架 F2，初值为 1。

A 车间的工作过程可描述为：

```

while(1){
    生产一个产品 A;
    P(empty1);
    //判断货架 F1 是否有空
    P(mutex1);
    //互斥访问货架 F1
    将产品 A 存放到货架 F1 上;
    V(mutex1);
    //释放货架 F1
    V(full1);
    //货架 F1 上的零件 A 的个数加 1
}

```

B 车间的工作过程可描述为：

```

while(1){
    生产一个产品 B;
    P(empty2);
    //判断货架 F2 是否有空
    P(mutex2);
    //互斥访问货架 F2
    将产品 B 存放到货架 F2 上;
    V(mutex2);
    //释放货架 F2
    V(full2);
    //货架 F2 上的零件 B 的个数加 1
}

```

装配车间的工作过程可描述为：

```

while(1){
    P(full1);
    //判断货架 F1 上是否有产品 A
    P(mutex1);
    //互斥访问货架 F1
    从货架 F1 上取一个 A 产品;
    V(mutex1);
    //释放货架 F1
    V(empty1);
    //货架 F1 上的空闲空间数加 1
    P(full2);
    //判断货架 F2 上是否有产品 B
    P(mutex2);
    //互斥访问货架 F2
    从货架 F2 上取一个 B 产品;
    V(mutex2);
    //释放货架 F2
    V(empty2);
    //货架 F2 上的空闲空间数加 1
    将取得的 A 产品和 B 产品组装成产品;
}

```

10. 解答：

从井中取水并放入水缸是一个连续的动作，可视为一个进程；从缸中取水可视为另一个进程。设水井和水缸为临界资源，引入 well 和 vat；三个水桶无论是从井中取水还是将水倒入水缸都是一次一个，应该给它们一个信号量 pail，抢不到水桶的进程只好等待。水缸满时，不可以再放水，设置 empty 信号量来控制入水量；水缸空时，不可以取水，设置 full 信号量来控制。本题需要设置 5 个信号量来进行控制：

```

semaphore well=1;           //用于互斥地访问水井
semaphore vat=1;           //用于互斥地访问水缸
semaphore empty=10;         //用于表示水缸中剩余空间能容纳的水的桶数
semaphore full=0;           //表示水缸中的水的桶数
semaphore pail=3;           //表示有多少个水桶可以用，初值为 3
//老和尚
while(1){
    P(full);
    P(pail);
    P(vat);
    从水缸中打一桶水;
    V(vat);
    V(empty);
    喝水;
    V(pail);
}
//小和尚
while(1){
    P(empty);
    P(pail);
    P(well);
    从井中打一桶水;
    V(well);
    P(vat);
    将水倒入水缸中;
    V(vat);
    V(full);
    V(pail);
}

```

11. 解答：

为了控制三个进程依次使用输入设备进行输入，需分别设置三个信号量 S1, S2, S3，其中 S1 的初值为 1, S2 和 S3 的初值为 0。使用上述信号量后，三个进程不会同时使用输入设备，因此不必再为输入设备设置互斥信号量。另外，还需要设置信号量 Sb, Sy, Sz 来表示数据 b 是否已经输入，以及 y, z 是否已计算完成，它们的初值均为 0。三个进程的动作可描述为：

```

P1(){
    P(S1);
    从输入设备输入数据 a;
    V(S2);
    P(Sb);
    x=a+b;
    P(Sy);
    P(Sz);
    使用打印机打印出 x, y, z 的结果;
}

```

```

    }
P2() {
    P(S2);
    从输入设备输入数据 b;
    V(S3);
    V(Sb);
    y=a*b;
    V(Sy);
    V(Sy);
}
P3() {
    P(S3);
    从输入设备输入数据 c;
    P(Sy);
    Z=y+c-a;
    V(Sz);
}

```

12. 解答：

互斥资源：取号机（一次只有一位顾客领号），因此设置互斥信号量 mutex。

同步问题：顾客需要获得空座位等待叫号。营业员空闲时，将选取一位顾客并为其服务。空座位的有、无影响等待顾客的数量，顾客的有无决定了营业员是否能开始服务，因此分别设置信号量 empty 和 full 来实现这一同步关系。另外，顾客获得空座位后，需要等待叫号和被服务。这样，顾客与营业员就服务何时开始又构成了一个同步关系，定义信号量 service 来完成这一同步过程。

```

semaphore empty=10;           //空座位的数量，初值为 10
semaphore mutex=1;            //互斥使用取号机
semaphore full=0;             //已占座位的数量，初值 0
semaphore service=0;          //等待叫号
cobegin
{
    Process 顾客 i
    {
        P(empty);           //等空位
        P(mutex);            //申请使用取号机
        从取号机上取号;
        V(mutex);             //取号完毕
        V(full);              //通知营业员有新顾客
        P(service);           //等待营业员叫号
        接受服务;
    }
    Process 营业员
    {
        while(True){
            P(full);           //没有顾客则休息
            V(empty);            //离开座位
            V(service);           //叫号
            为顾客服务;
        }
    }
}coend

```

13. 解答：

- 1) 桥上每次只能有一辆车行驶，所以只要设置一个信号量 bridge 就可判断桥是否使用，若在使用中，等待；若无人使用，则执行 P 操作进入；出桥后，执行 V 操作。

```

semaphore bridge=1;           //用于互斥地访问桥
NtoS() {
    P(bridge);               //从北向南
    通过桥;
    V(bridge);
}
StoN() {                      //从南向北
    P(bridge);
    通过桥;
    V(bridge);
}

```

- 2) 桥上可以同方向多车行驶，需要设置 bridge，还需要对同方向车辆计数。为了防止同方向计数中同时申请 bridge 造成同方向不可同时行车的问题，要对计数过程加以保护，因此设置信号量 mutexSN 和 mutexNS。

```

int countSN=0;                //用于表示从南到北的汽车数量
int countNS=0;                //用于表示从北到南的汽车数量
semaphore mutexSN=1;          //用于保护 countSN
semaphore mutexNS=1;          //用于保护 countNS
semaphore bridge=1;           //用于互斥地访问桥
StoN() {
    P(mutexSN);               //从南向北
    if(countSN==0)
        P(bridge);
    countSN++;
    V(mutexSN);
    过桥;
    P(mutexSN);
    countSN--;
    if(countSN==0)
        V(bridge);
    V(mutexSN);
}
NtoS() {                      //从北向南
    P(mutexNS);
    if(countNS==0)
        P(bridge);
    countNS++;
    V(mutexNS);
    过桥;
    P(mutexNS);
    countNS--;
    if(countNS==0)
        V(bridge);
    V(mutexNS);
}

```

14. 答解：

- 1) 这种机制不能实现资源的互斥访问。考虑如下情形：

- ① 初始化时, flag 数组的两个元素值均为 FALSE。
- ② 线程 0 先执行, 执行 while 循环语句时, 由于 flag[1] 为 FALSE, 所以顺利结束, 不会被卡住。假设这时来了一个时钟中断, 打断了它的运行。
- ③ 线程 1 去执行, 执行 while 循环语句时, 由于 flag[0] 为 FALSE, 所以顺利结束, 不会被卡住, 然后进入临界区。
- ④ 后来当线程 0 再执行时, 也进入临界区, 这样就同时有两个线程在临界区。

总结: 不能成功的根本原因是无法保证 Enter_Critical_Section() 函数执行的原子性。我们从上面的软件实现方法中可以看出, 对于两个进程间的互斥, 最主要的问题是标志的检查和修改不能作为一个整体来执行, 因此容易导致无法保证互斥访问的问题。

2) 可能会出现死锁。考虑如下情形:

- ① 初始化时, flag 数组的两个元素值均为 FALSE。
- ② 线程 0 先执行, flag[0] 为 TRUE, 假设这时来了一个时钟中断, 打断了它的运行。
- ③ 线程 1 去执行, flag[1] 为 TRUE, 在执行 while 循环语句时, 由于 flag[0] = TRUE, 所以在这个地方被卡住, 直到时间片用完。
- ④ 线程 0 再执行时, 由于 flag[1] 为 TRUE, 它也在 while 循环语句的地方被卡住, 因此这两个线程都无法执行下去, 从而死锁。

15. 分析:

用信号量与 PV 操作实现三名工人的合作。

首先不考虑死锁问题, 工人 1 与工人 3、工人 2 与工人 3 构成生产者与消费者关系, 这两对生产/消费关系通过共同的缓冲区相联系。从资源的角度来看, 箱子中的空位置相当于工人 1 和工人 2 的资源, 而车架和车轮相当于工人 3 的资源。

分析上述解法易见, 当工人 1 推进速度较快时, 箱中空位置可能完全被车架占满或只留有一个存放车轮的位置, 此时工人 3 同时取 2 个车轮将无法得到, 而工人 2 又无法将新加工的车轮放入箱中; 当工人 2 推进速度较快时, 箱中空位置可能完全被车轮占满, 而此时工人 3 取车架将无法得到, 而工人 1 又无法将新加工的车架放入箱中。上述两种情况都意味着死锁。为防止死锁的发生, 箱中车架的数量不可超过 $N - 2$, 车轮的数量不可超过 $N - 1$, 这些限制可以用两个信号量来表达。

解答:

```

semaphore empty=N;           //空位置
semaphore wheel=0;           //车轮
semaphore frame=0;           //车架
semaphore s1=N-2;             //车架最大数
semaphore s2=N-1;             //车轮最大数

工人 1 活动:
do{
    加工一个车架;
    P(s1);                  //检查车架数是否达到最大值
    P(empty);                //检查是否有空位
    车架放入箱中;
    V(frame);                //车架数加 1
}while(1);

工人 2 活动:
do{
    加工一个车轮;
    P(s2);                  //检查车轮数是否达到最大值
    P(empty);                //检查是否有空位
    车轮放入箱中;
    V(frame);                //车架数减 1
}while(1);

```

```

P(s2);          //检查车轮数是否达到最大值
P(empty);       //检查是否有空位
车轮放入箱中;
V(wheel);       //车轮数加 1
}while(1);
工人 3 活动:
do{
    P(frame);      //检查是否有车架
    箱中取一车架;
    V(empty);       //空位数加 1
    V(s1);          //可装入车架数加 1
    P(wheel);        //检查是否有一个车轮
    P(wheel);        //检查是否有另一个车轮
    箱中取二车轮;
    V(empty);       //取走一个车轮, 空位数加 1
    V(empty);       //取走另一个车轮, 空位数加 1
    V(s2);          //可装入车轮数加 1
    V(s2);          //可装入车轮数再加 1
    组装为一台车;
}while(1);

```

16. 解答:

P, Q 构成消费者-生产者关系, 因此设三个信号量 full, empty, mutex。full 和 empty 用来控制缓冲池状态, mutex 用来互斥进入。R 既为消费者又为生产者, 因此必须在执行前判断状态, 若 empty==1, 则执行生产者功能; 若 full==1, 执行消费者功能。

Procedure P	Procedure Q	Procedure R
{	{	{
while(TRUE) {	while(TRUE) {	if(empty==1){
p(empty);	p(full);	p(empty);
P(mutex);	P(mutex);	P(mutex);
Product one;	consume one;	product one;
v(mutex);	v(mutex);	v(mutex);
v(full);	v(empty);	v(full);
}	}	}
}		

17. 解答:

- 1) 控制变量 waiting 用来记录等候理发的顾客数, 初值为 0, 进来一个顾客时, waiting 加 1, 一个顾客理发时, waiting 减 1。
- 2) 信号量 customers 用来记录等候理发的顾客数, 并用作阻塞理发师进程, 初值为 0。
- 3) 信号量 barbers 用来记录正在等候顾客的理发师数, 并用作阻塞顾客进程, 初值为 0。

4) 信号量 mutex 用于互斥, 初值为 1。

```

int waiting=0;           //等候理发的顾客数
int chairs=n;            //为顾客准备的椅子数
semaphore customers=0,barbers=0,mutex=1;
barber(){
    while(1){          //理完一人, 还有顾客吗?
        P(customers);   //若无顾客, 理发师睡眠
        P(mutex);         //进程互斥
        waiting=waiting-1; //等候顾客数少一个
        V(barbers);       //理发师去为一个顾客理发
        V(mutex);         //开放临界区
        Cut_hair();        //正在理发
    }
}
customer(){           //顾客进程
    P(mutex);             //进程互斥
    if(waiting<chairs){  //若有空的椅子, 就找到椅子坐下等待
        waiting=waiting+1; //等候顾客数加 1
        V(customers);     //唤醒理发师
        V(mutex);          //开放临界区
        P(barbers);        //无理发师, 顾客坐着
        get_haircut();      //一个顾客坐下等待理发
    }
    else
        V(mutex);          //人满, 离开
}

```

18. 解答:

电影院一次只能放映一部影片, 希望观看的观众可能有不同的爱好, 但每次只能满足部分观众的需求, 即希望观看另外两部影片的用户只能等待。分别为三部影片设置三个信号量 s0, s1, s2, 初值分别为 1, 1, 1。电影院一次只能放一部影片, 因此需要互斥使用。由于观看影片的观众有多个, 因此必须分别设置三个计数器 (初值都是 0), 用来统计观众个数。当然, 计数器是个共享变量, 需要互斥使用。

```

int s=1,s0=1,s1=1,s2=1;
int count0=0,count1=0,count2=0;
process videoshow1{           //看第一部影片的观众
    P(s0);
    count0=count0+1;
    if(count0==1)
        P(s);
    V(s0);
    看影片;
    P(s0);
    count0=count0-1;
    if(count0==0)                 //没人看了, 就结束放映
        V(s);
    V(s0);
}
process videoshow2{           //看第二部影片的观众
}

```

```

P(s1);
count1=count1+1;
if(count1==1)
    P(s);
V(s1);
看影片;
P(s1);
count1=count1-1;
if(count1==0)           //没人看了，就结束放映
    V(s);
V(s1);
}
process videoshow3{          //看第三部影片的观众
P(s2);
count2=count2+1;
if(count2==1)
    P(s);
V(s2);
看影片;
P(s2);
count1=count1-1;
if(count2==0)           //没人看了，就结束放映
    V(s);
V(s2);
}

```

19. 解答：

由于安全岛 M 仅允许两辆车停留，本应作为临界资源而设置信号量，但根据题意，任意时刻进入安全岛的车不会超过两辆（两个方向最多各有一辆），因此不需要为 M 设置信号量，而要在路口 T 和路口 N 都设置信号量，以控制来自两个方向的车对路口资源的争夺。这两个信号量的初值都是 1。此外，由于从 N 到 T 的一段路只允许一辆车通过，所以还需要设置另外的信号量用于控制，由于 M 的存在，可为两端的小路分别设置一个互斥信号量。

```

int T2N=1;                  //从 T 到 N 的互斥信号量
int N2T=1;                  //从 N 到 T 的互斥信号量
int L=1;                    //经过 L 路段的互斥信号量
int K=1;                    //经过 K 路段的互斥信号量

Procedure Bike T2N{
    P(T2N);
    P(L);
    go T to L;
    go into M;
    V(L);
    P(K);
    go K to N;
    V(K);
    V(T2N);
}

Procedure Bike N2T{
    P(N2T);

```

```

    p(k);
    go N to k;
    go into M;
    V(k);
    P(L);
    go L to T;
    V(L);
    V(N2T);
}

```

20. 解答：

这是典型的生产者和消费者问题，只对典型问题加了一个条件，只需在标准模型上新加一个信号量，即可完成指定要求。

设置 4 个变量 mutex1, mutex2, empty 和 full, mutex1 用于控制一个消费者进程在一个周期(10 次)内对缓冲区的访问，初值为 1; mutex2 用于控制进程单次互斥地访问缓冲区，初值为 1; empty 代表缓冲区的空位数，初值为 1000; full 代表缓冲区的产品数，初值为 0，具体进程描述如下：

```

semaphore mutex1=1;
semaphore mutex2=1;
semaphore empty=n;
semaphore full=0;
producer(){
    while(1){
        生产一个产品;
        P(empty);           //判断缓冲区是否有空位
        P(mutex2);          //互斥访问缓冲区
        把产品放入缓冲区;
        V(mutex2);          //互斥访问缓冲区
        V(full);            //产品的数量加 1
    }
}

consumer(){
    while(1){
        P(mutex1)           //连续取 10 次
        for(int i=0; i<10; ++i){
            P(full);         //判断缓冲区是否有产品
            P(mutex2);        //互斥访问缓冲区
            从缓冲区取出一件产品;
            V(mutex2);         //互斥访问缓冲区
            V(empty);          //腾出一个空位
            消费这件产品;
        }
        V(mutex1)
    }
}

```

21. 解答：

在汽车行驶过程中，驾驶员活动与售票员活动之间的同步关系为：售票员关车门后，向驾驶员发开车信号，驾驶员接到开车信号后启动车辆，在汽车正常行驶过程中售票员售票，到站时驾驶员停车，售票员在车停后开门让乘客上下车。因此，驾驶员启动车辆的动作必须与售票员关车

门的动作同步；售票员开车门的动作也必须与驾驶员停车同步。应设置两个信号量 S1, S2：S1 表示是否允许驾驶员启动汽车（初值为 0）；S2 表示是否允许售票员开门（初值为 0）。

```

semaphore S1=S2=0;
Procedure driver
{
    while(1)
    {
        P(S1);
        Start;
        Driving;
        Stop;
        V(S2);
    }
}
Procedure Conductor
{
    while(1)
    {
        关车门;
        V(s1);
        售票;
        P(s2);
        开车门;
        上下乘客;
    }
}

```

22. 解答：

```

semaphore Full_A = x;           //Full_A 表示 A 的信箱中的邮件数量
semaphore Empty_A = M-x;        //Empty_A 表示 A 的信箱中还可存放的邮件数量
semaphore Full_B = y;           //Full_B 表示 B 的信箱中的邮件数量
semaphore Empty_B = N-y;         //Empty_B 表示 B 的信箱中还可存放的邮件数量
semaphore mutex_A = 1;          //mutex_A 用于 A 的信箱互斥
semaphore mutex_B = 1;          //mutex_B 用于 B 的信箱互斥

```

Cobegin

<pre> A{ while(TRUE){ P(Full_A); P(mutex_A); 从 A 的信箱中取出一个邮件; V(mutex_A); V(Empty_A); 回答问题并提出一个新问题; P(Empty_B); P(mutex_B); 将新邮件放入 B 的信箱; V(mutex_B); V(Full_B); } } </pre>	<pre> B{ while(TRUE){ P(Full_B); P(mutex_B); 从 B 的信箱中取出一个邮件; V(mutex_B); V(Empty_B); 回答问题并提出一个新问题; P(Empty_A); P(mutex_A); 将新邮件放入 A 的信箱; V(mutex_A); V(Full_A); } } </pre>
--	--

23. 解答：

先找出线程对在各个变量上的互斥、并发关系。若为一读一写或两个都为写，则为互斥关系。每个互斥关系都需要一个信号量进行调节。

```

semaphore mutex_y1=1; //mutex_y1 用于 thread1 与 thread3 对变量 y 的互斥访问
semaphore mutex_y2=1; //mutex_y2 用于 thread2 与 thread3 对变量 y 的互斥访问
semaphore mutex_z=1;  //mutex_z 用于变量 z 的互斥访问

```

互斥代码如下：

thread1	thread2	thread3
<pre> cnum w; wait(mutex_y1); w = add(x, y); signal(mutex_y1); ... } </pre>	<pre> cnum w; wait(mutex_y2); w = add(y, z); signal(mutex_z); ... } </pre>	<pre> cnum w; w.a = 1; w.b = 1; wait(mutex_z); z = add(z, w); signal(mutex_z); wait(mutex_y1); wait(mutex_y2); y = add(y, w); signal(mutex_y1); signal(mutex_y2); ... } </pre>

24. 解答：

回顾传统的哲学家问题，假设餐桌上有 n 名哲学家、 n 根筷子，那么可以用这种方法避免死锁（本书考点讲解中提供了这一思路）：限制至多允许 $n-1$ 名哲学家同时“抢”筷子，那么至少会有 1 名哲学家可以获得两根筷子并顺利进餐，于是不可能发生死锁的情况。

本题可以用碗这个限制资源来避免死锁：当碗的数量 m 小于哲学家的数量 n 时，可以让碗的资源量等于 m ，确保不会出现所有哲学家都拿一侧筷子而无限等待另一侧筷子进而造成死锁的情况；当碗的数量 m 大于等于哲学家的数量 n 时，为了让碗起到同样的限制效果，我们让碗的资源量等于 $n-1$ ，这样就能保证最多只有 $n-1$ 名哲学家同时进餐，所以得到碗的资源量为 $\min\{n-1, m\}$ 。在进行 PV 操作时，碗的资源量起限制哲学家取筷子的作用，所以需要先对碗的资源量进行 P 操作。具体过程如下：

```

//信号量
semaphore bowl;           //用于协调哲学家对碗的使用
semaphore chopsticks[n];   //用于协调哲学家对筷子的使用
for(int i=0;i<n;i++)
    chopsticks[i]=1;        //设置两名哲学家之间筷子的数量
bowl=min(n-1,m);          //bowl≤n-1, 确保不死锁

CoBegin
while(TRUE){                //哲学家 i 的程序
    思考;
    P(bowl);                //取碗
    P(chopsticks[i]);        //取左边筷子
    P(chopsticks[(i+1)%n]); //取右边筷子
    就餐;
    V(chopsticks[i]);
    V(chopsticks[(i+1)%n]);
    V(bowl);
}
CoEnd

```

25. 解答：

本题要求实现操作的先后顺序，没有互斥关系，是一个简单的同步问题。

本题虽然有 5 个操作，但是只有 4 个同步关系，因此分别设置信号量 SAC、SBC、SCE 和 SDE 对应 4 个同步关系。

```
Semaphore SAC = 0;      //控制 A 和 C 的执行顺序
Semaphore SBC = 0;      //控制 B 和 C 的执行顺序
Semaphore SCE = 0;      //控制 C 和 E 的执行顺序
Semaphore SDE = 0;      //控制 D 和 E 的执行顺序
```

5 个操作可描述为如下。

```
CoBegin
A() {
    完成动作 A;
    V(SAC);           //实现 A、C 之间的同步关系
}
B() {
    完成动作 B;
    V(SBC);           //实现 B、C 之间的同步关系
}
C() {
    //C 必须在 A、B 都完成后才能完成
    P(SAC);
    P(SBC);
    完成动作 C;
    V(SCE);           //实现 C、E 之间的同步关系
}
D() {
    完成动作 D;
    V(SDE);           //实现 D、E 之间的同步关系
}
E() {
    //E 必须在完成 C、D 之后执行
    P(SCE);
    P(SDE);
    完成动作 E;
}
CoEnd
```

2.4 死锁

在学习本节时，请读者思考以下问题：

- 1) 为什么会产生死锁？产生死锁有什么条件？
- 2) 有什么办法可以解决死锁问题？

学完本节，读者应了解死锁的由来、产生条件及基本解决方法，区分死锁的避免和死锁的预防。

2.4.1 死锁的概念

1. 死锁的定义

在多道程序系统中，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，多个进程的并发执行也带来了新的问题—死锁。所谓死锁，是指多个进程因竞争资源而造成的一种僵局（互相等待），若无外力作用，这些进程都将无法向前推进。

下面通过一些实例来说明死锁现象。

先看生活中的一个实例。在一条河上有一座桥，桥面很窄，只能容纳一辆汽车通行。若有两辆汽车分别从桥的左右两端驶上该桥，则会出现下述冲突情况：此时，左边的汽车占有桥面左边的一段，要想过桥还需等待右边的汽车让出桥面右边的一段；右边的汽车占有桥面右边的一段，要想过桥还需等待左边的汽车让出桥面左边的一段。此时，若左右两边的汽车都只能向前行驶，则两辆汽车都无法过桥。

在计算机系统中也存在类似的情况。例如，某计算机系统中只有一台打印机和一台输入设备，进程 P_1 正占用输入设备，同时又提出使用打印机的请求，但此时打印机正被进程 P_2 所占用，而 P_2 在未释放打印机之前，又提出请求使用正被 P_1 占用的输入设备。这样，两个进程相互无休止地等待下去，均无法继续执行，此时两个进程陷入死锁状态。

2. 死锁产生的原因

(1) 系统资源的竞争

通常系统中拥有的不可剥夺资源，其数量不足以满足多个进程运行的需要，使得进程在运行过程中，会因争夺资源而陷入僵局，如磁带机、打印机等。只有对不可剥夺资源的竞争才可能产生死锁，对可剥夺资源的竞争是不会引起死锁的。

(2) 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，也同样会导致死锁。例如，并发进程 P_1, P_2 分别保持了资源 R_1, R_2 ，而进程 P_1 申请资源 R_2 、进程 P_2 申请资源 R_1 时，两者都会因为所需资源被占用而阻塞。

信号量使用不当也会造成死锁。进程间彼此相互等待对方发来的消息，也会使得这些进程间无法继续向前推进。例如，进程 A 等待进程 B 发的消息，进程 B 又在等待进程 A 发的消息，可以看出进程 A 和 B 不是因为竞争同一资源，而是在等待对方的资源导致死锁。

(3) 死锁产生的必要条件

产生死锁必须同时满足以下 4 个条件，只要其中任意一个条件不成立，死锁就不会发生。

互斥条件：进程要求对所分配的资源（如打印机）进行排他性控制，即在一段时间内某资源仅为一个进程所占有。此时若有其他进程请求该资源，则请求进程只能等待。

不剥夺条件：进程所获得的资源在未使用完之前，不能被其他进程强行夺走，即只能由获得该资源的进程自己来释放（只能是主动释放）。

请求并保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。

循环等待条件：存在一种进程资源的循环等待链，链中每个进程已获得的资源同时被链中下一个进程所请求。即存在一个处于等待态的进程集合 $\{P_1, P_2, \dots, P_n\}$ ，其中 P_i 等待的资源被 P_{i+1} ($i = 0, 1, \dots, n-1$) 占有， P_n 等待的资源被 P_0 占有，如图 2.11 所示。

直观上看，循环等待条件似乎和死锁的定义一样，其实不然。按死锁定义构成等待环所要求的条件更严，它要求 P_i 等待的资源必须由 P_{i+1} 来满足，而循环等待条件则无此限制。例如，系统中有两台输出设备， P_0 占有一台， P_K 占有另一台，且 K 不属于集合 $\{0, 1, \dots, n\}$ 。 P_n 等待一台输出设备，它可从 P_0 获得，也可能从 P_K 获得。因此，虽然 P_n, P_0 和其他一些进程形成了循环等待圈，但 P_K 不在圈内，若 P_K 释放了输出设备，则可打破循环等待，如图 2.12 所示。因此循环等待只是死锁的必要条件。

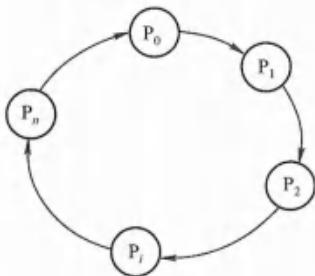


图 2.11 循环等待

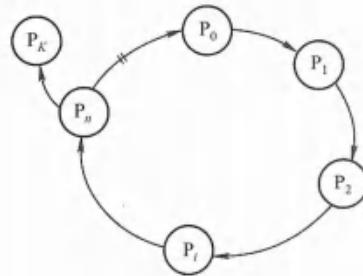


图 2.12 满足条件但无死锁

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 1。但若系统中每类资源都只有一个资源，则资源分配图含圈就变成了系统出现死锁的充分必要条件。

要注意区分不剥夺条件与请求并保持条件。下面用一个简单的例子进行说明：若你手上拿着一个苹果（即便你不打算吃），别人不能把你手上的苹果拿走，则这就是不剥夺条件；若你左手拿着一个苹果，允许你右手再去拿一个苹果，则这就是请求并保持条件。

2.4.2 死锁的处理策略

为使系统不发生死锁，必须设法破坏产生死锁的 4 个必要条件之一，或允许死锁产生，但当死锁发生时能检测出死锁，并有能力实现恢复。

1. 死锁预防

设置某些限制条件，破坏产生死锁的 4 个必要条件中的一个或几个，以防止发生死锁。

2. 避免死锁

在资源的动态分配过程中，用某种方法防止系统进入不安全状态，从而避免死锁。

3. 死锁的检测及解除

无须采取任何限制性措施，允许进程在运行过程中发生死锁。通过系统的检测机构及时地检测出死锁的发生，然后采取某种措施解除死锁。

预防死锁和避免死锁都属于事先预防策略，预防死锁的限制条件比较严格，实现起来较为简单，但往往导致系统的效率低，资源利用率低；避免死锁的限制条件相对宽松，资源分配后需要通过算法来判断是否进入不安全状态，实现起来较为复杂。

死锁的几种处理策略的比较见表 2.4。

表 2.4 死锁处理策略的比较

	资源分配策略	各种可能模式	主要优点	主要缺点
死锁预防	保守，宁可资源闲置	一次请求所有资源，资源剥夺，资源按序分配	适用于突发式处理的进程，不必进行剥夺	效率低，进程初始化时间延长；剥夺次数过多；不便灵活申请新资源
死锁避免	是“预防”和“检测”的折中（在运行时判断是否可能死锁）	寻找可能的安全允许顺序	不必进行剥夺	必须知道将来的资源需求；进程不能被长时间阻塞
死锁检测	宽松，只要允许就分配资源	定期检查死锁是否已经发生	不延长进程初始化时间，允许对死锁进行现场处理	通过剥夺解除死锁，造成损失

2.4.3 死锁预防

防止死锁的发生只需破坏死锁产生的 4 个必要条件之一即可。

1. 破坏互斥条件

若允许系统资源都能共享使用，则系统不会进入死锁状态。但有些资源根本不能同时访问，如打印机等临界资源只能互斥使用。所以，破坏互斥条件而预防死锁的方法不太可行，而且在有的场合应该保护这种互斥性。

2. 破坏不剥夺条件

当一个已保持了某些不可剥夺资源的进程请求新的资源而得不到满足时，它必须释放已经保持的所有资源，待以后需要时再重新申请。这意味着，一个进程已占有的资源会被暂时释放，或者说是被剥夺，或从而破坏了不剥夺条件。

该策略实现起来比较复杂，释放已获得的资源可能造成前一阶段工作的失效，反复地申请和释放资源会增加系统开销，降低系统吞吐量。这种方法常用于状态易于保存和恢复的资源，如 CPU 的寄存器及内存资源，一般不能用于打印机之类的资源。

3. 破坏请求并保持条件

采用预先静态分配方法，即进程在运行前一次申请完它所需要的全部资源，在它的资源未满足前，不把它投入运行。一旦投入运行，这些资源就一直归它所有，不再提出其他资源请求，这样就可以保证系统不会发生死锁。

这种方式实现简单，但缺点也显而易见，系统资源被严重浪费，其中有些资源可能仅在运行初期或运行快结束时才使用，甚至根本不使用。而且还会导致“饥饿”现象，由于个别资源长期被其他进程占用时，将致使等待该资源的进程迟迟不能开始运行。

4. 破坏循环等待条件

为了破坏循环等待条件，可采用顺序资源分配法。首先给系统中的资源编号，规定每个进程必须按编号递增的顺序请求资源，同类资源一次申请完。也就是说，只要进程提出申请分配资源 R_i ，则该进程在以后的资源申请中就只能申请编号大于 R_i 的资源。

这种方法存在的问题是，编号必须相对稳定，这就限制了新类型设备的增加；尽管在为资源编号时已考虑到大多数作业实际使用这些资源的顺序，但也经常会发生作业使用资源的顺序与系统规定顺序不同的情况，造成资源的浪费；此外，这种按规定次序申请资源的方法，也必然会给用户的编程带来麻烦。

2.4.4 死锁避免

避免死锁同样属于事先预防策略，但并不是事先采取某种限制措施破坏死锁的必要条件，而是在资源动态分配过程中，防止系统进入不安全状态，以避免发生死锁。这种方法所施加的限制条件较弱，可以获得较好的系统性能。

1. 系统安全状态

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

所谓安全状态，是指系统能按某种进程推进顺序 (P_1, P_2, \dots, P_n) 为每个进程 P_i 分配其所需的资源，直至满足每个进程对资源的最大需求，使每个进程都可顺序完成。此时称 P_1, P_2, \dots, P_n 为

安全序列。若系统无法找到一个安全序列，则称系统处于不安全状态。

假设系统中有三个进程 P_1 , P_2 和 P_3 , 共有 12 台磁带机。进程 P_1 共需要 10 台磁带机, P_2 和 P_3 分别需要 4 台和 9 台。假设在 T_0 时刻, 进程 P_1 , P_2 和 P_3 已分别获得 5 台、2 台和 2 台, 尚有 3 台未分配, 见表 2.5。

在 T_0 时刻是安全的, 因为存在一个安全序列 P_2, P_1, P_3 , 只要系统按此进程序列分配资源, 那么每个进程都能顺利完成。也就是说, 当前可用磁带机为 3 台, 先把 3 台磁带机分配给 P_2 以满足其最大需求, P_2 结束并归还资源后, 系统有 5 台磁带机可用; 接下来给 P_1 分配 5 台磁带机以满足其最大需求, P_1 结束并归还资源后, 剩余 10 台磁带机可用; 最后分配 7 台磁带机给 P_3 , 这样 P_3 也能顺利完成。

若在 T_0 时刻后, 系统分配 1 台磁带机给 P_3 , 系统剩余可用资源数为 2, 此时系统进入不安全状态, 因为此时已无法再找到一个安全序列。当系统进入不安全状态后, 便可能导致死锁。例如, 把剩下的 2 台磁带机分配给 P_2 , 这样, P_2 完成后只能释放 4 台磁带机, 既不能满足 P_1 又不能满足 P_3 , 致使它们都无法推进到完成, 彼此都在等待对方释放资源, 陷入僵局, 即导致死锁。

并非所有的不安全状态都是死锁状态, 但当系统进入不安全状态后, 便可能进入死锁状态; 反之, 只要系统处于安全状态, 系统便可避免进入死锁状态。

2. 银行家算法

银行家算法是最著名的死锁避免算法, 其思想是: 把操作系统视为银行家, 操作系统管理的资源相当于银行家管理的资金, 进程向操作系统请求分配资源相当于用户向银行家贷款。操作系统按照银行家制定的规则为进程分配资源。进程运行之前先声明对各种资源的最大需求量, 当进程在执行中继续申请资源时, 先测试该进程已占用的资源数与本次申请的资源数之和是否超过该进程声明的最大需求量。若超过则拒绝分配资源, 若未超过则再测试系统现存的资源能否满足该进程尚需的最大资源量, 若能满足则按当前的申请量分配资源, 否则也要推迟分配。

(1) 数据结构描述

可利用资源向量 Available: 含有 m 个元素的数组, 其中每个元素代表一类可用的资源数目。Available[j] = K 表示系统中现有 R_j 类资源 K 个。

最大需求矩阵 Max: $n \times m$ 矩阵, 定义系统中 n 个进程中的每个进程对 m 类资源的最大需求。简单来说, 一行代表一个进程, 一列代表一类资源。Max[i, j] = K 表示进程 i 需要 R_j 类资源的最大数目为 K 。

分配矩阵 Allocation: $n \times m$ 矩阵, 定义系统中每类资源当前已分配给每个进程的资源数。Allocation[i, j] = K 表示进程 i 当前已分得 R_j 类资源的数目为 K 。初学者容易混淆 Available 向量和 Allocation 矩阵, 在此特别提醒。

需求矩阵 Need: $n \times m$ 矩阵, 表示每个进程接下来最多还需要多少资源。Need[i, j] = K 表示进程 i 还需要 R_j 类资源的数目为 K 。

上述三个矩阵间存在下述关系:

$$\text{Need} = \text{Max} - \text{Allocation}$$

一般情况下, 在银行家算法的题目中, Max 矩阵和 Allocation 矩阵是已知条件, 而求出 Need 矩阵是解题的第一步。

表 2.5 资源分配

进程	最大需求	已分配	可用
P_1	10	5	3
P_2	4	2	
P_3	9	2	

(2) 银行家算法描述

设 Request_i 是进程 P_i 的请求向量, $\text{Request}_i[j] = K$ 表示进程 P_i 需要 j 类资源 K 个。当 P_i 发出资源请求后, 系统按下述步骤进行检查:

- ① 若 $\text{Request}_i[j] \leq \text{Need}[i, j]$, 则转向步骤②; 否则认为出错, 因为它所需要的资源数已超过它所宣布的最大值。
- ② 若 $\text{Request}_i[j] \leq \text{Available}[j]$, 则转向步骤③; 否则, 表示尚无足够资源, P_i 须等待。
- ③ 系统试探着把资源分配给进程 P_i , 并修改下面数据结构中的数值:

$$\begin{aligned}\text{Available} &= \text{Available} - \text{Request}_i; \\ \text{Allocation}[i, j] &= \text{Allocation}[i, j] + \text{Request}_i[j]; \\ \text{Need}[i, j] &= \text{Need}[i, j] - \text{Request}_i[j];\end{aligned}$$

- ④ 系统执行安全性算法, 检查此次资源分配后, 系统是否处于安全状态。若安全, 才正式将资源分配给进程 P_i , 以完成本次分配; 否则, 将本次的试探分配作废, 恢复原来的资源分配状态, 让进程 P_i 等待。

(3) 安全性算法

设置工作向量 Work , 有 m 个元素, 表示系统中的剩余可用资源数目。在执行安全性算法开始时, $\text{Work} = \text{Available}$ 。

- ① 初始时安全序列为空。
- ② 从 Need 矩阵中找出符合下面条件的行: 该行对应的进程不在安全序列中, 而且该行小于等于 Work 向量, 找到后, 把对应的进程加入安全序列; 若找不到, 则执行步骤④。
- ③ 进程 P_i 进入安全序列后, 可顺利执行, 直至完成, 并释放分配给它的资源, 因此应执行 $\text{Work} = \text{Work} + \text{Allocation}[i]$, 其中 $\text{Allocation}[i]$ 表示进程 P_i 代表的在 Allocation 矩阵中对应的行, 返回步骤②。
- ④ 若此时安全序列中已所有进程, 则系统处于安全状态, 否则系统处于不安全状态。

看完上面对银行家算法的过程描述后, 可能会有眼花缭乱的感觉, 现在通过举例来加深理解。

3. 安全性算法举例

假定系统中有 5 个进程 $\{P_0, P_1, P_2, P_3, P_4\}$ 和三类资源 $\{A, B, C\}$, 各种资源的数量分别为 10, 5, 7, 在 T_0 时刻的资源分配情况见表 2.6。

T_0 时刻的安全性。利用安全性算法对 T_0 时刻的资源分配进行分析。

表 2.6 T_0 时刻的资源分配表

进程 资源情况	Max	Allocation	Available
	A B C	A B C	A B C
P_0	7 5 3	0 1 0	
P_1	3 2 2	2 0 0 (3 0 2)	
P_2	9 0 2	3 0 2	3 3 2 (2 3 0)
P_3	2 2 2	2 1 1	
P_4	4 3 3	0 0 2	

- ① 从题目中我们可以提取 Max 矩阵和 Allocation 矩阵, 这两个矩阵相减可得到 Need 矩阵:

$$\begin{array}{c} \left[\begin{array}{ccc} 7 & 5 & 3 \\ 3 & 2 & 2 \\ 9 & 0 & 2 \\ 2 & 2 & 2 \\ 4 & 3 & 3 \end{array} \right] - \left[\begin{array}{ccc} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{array} \right] = \left[\begin{array}{ccc} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{array} \right] \\ \text{Max} \quad \text{Allocation} \quad \text{Need} \end{array}$$

- ② 然后，将 Work 向量与 Need 矩阵的各行进行比较，找出比 Work 矩阵小的行。例如，在初始时，

$$(3,3,2) > (1,2,2)$$

$$(3,3,2) > (0,1,1)$$

对应的两个进程分别为 P_1 和 P_3 ，这里我们选择 P_1 （也可以选择 P_3 ）暂时加入安全序列。

- ③ 释放 P_1 所占的资源，即把 P_1 进程对应的 Allocation 矩阵中的一行与 Work 向量相加：

$$(3 \ 3 \ 2) + (2 \ 0 \ 0) = (5 \ 3 \ 2) = \text{Work}$$

此时需求矩阵更新为（去掉了 P_1 对应的一行）：

$$\begin{array}{c} P_0 \left[\begin{array}{ccc} 7 & 4 & 3 \end{array} \right] \\ P_2 \left[\begin{array}{ccc} 6 & 0 & 0 \end{array} \right] \\ P_3 \left[\begin{array}{ccc} 0 & 1 & 1 \end{array} \right] \\ P_4 \left[\begin{array}{ccc} 4 & 3 & 1 \end{array} \right] \end{array}$$

再用更新的 Work 向量和 Need 矩阵重复步骤②。利用安全性算法分析 T_0 时刻的资源分配情况如表 2.7 所示，最后得到一个安全序列 $\{P_1, P_3, P_4, P_2, P_0\}$ 。

表 2.7 T_0 时刻的安全序列的分析

资源情况 进程	Work			Need			Allocation			Work+Allocation		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	3	3	2	1	2	2	2	0	0	5	3	2
P_3	5	3	2	0	1	1	2	1	1	7	4	3
P_4	7	4	3	4	3	1	0	0	2	7	4	5
P_2	7	4	5	6	0	0	3	0	2	10	4	7
P_0	10	4	7	7	4	3	0	1	0	10	5	7

4. 银行家算法举例

安全性算法是银行家算法的核心，在银行家算法的题目中，一般会有某个进程的一个资源请求向量，读者只要执行上面所介绍的银行家算法的前三步，马上就会得到更新的 Allocation 矩阵和 Need 矩阵，再按照上例的安全性算法判断，就能知道系统能否满足进程提出的资源请求。

假设当前系统中资源的分配和剩余情况如表 2.6 所示。

(1) P_1 请求资源： P_1 发出请求向量 $\text{Request}_1(1, 0, 2)$ ，系统按银行家算法进行检查：

$$\text{Request}_1(1, 0, 2) \leq \text{Need}_1(1, 2, 2)$$

$$\text{Request}_1(1, 0, 2) > \text{Available}_1(3, 3, 2)$$

系统先假定可为 P_1 分配资源，并修改

$$\text{Available} = \text{Available} - \text{Request}_1 = (2, 3, 0)$$

$$\text{Allocation}_1 = \text{Allocation}_1 + \text{Request}_1 = (3, 0, 2)$$

$$\text{Need}_1 = \text{Need}_1 - \text{Request}_1 = (0, 2, 0)$$

由此形成的资源变化情况如表 2.6 中的圆括号所示。

令 $\text{Work} = \text{Available} = (2, 3, 0)$, 再利用安全性算法检查此时系统是否安全, 如表 2.8 所示。

表 2.8 P_1 申请资源时的安全性检查

资源情况 进程	Work			Need			Allocation			Work+Allocation		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	2	3	0	0	2	0	3	0	2	5	3	2
P_3	5	3	2	0	1	1	2	1	1	7	4	3
P_4	7	4	3	4	3	1	0	0	2	7	4	5
P_0	7	4	5	7	4	3	0	1	0	7	5	5
P_2	7	5	5	6	0	0	3	0	2	10	5	7

由所进行的安全性检查得知, 可找到一个安全序列 $\{P_1, P_3, P_4, P_2, P_0\}$ 。因此, 系统是安全的, 可以立即将 P_1 所申请的资源分配给它。分配后系统中的资源情况如表 2.9 所示。

表 2.9 为 P_1 分配资源后的有关资源数据

资源情况 进程	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

(2) P_4 请求资源: P_4 发出请求向量 $\text{Request}_4(3, 3, 0)$, 系统按银行家算法进行检查:

$$\text{Request}_4(3, 3, 0) \leq \text{Need}_4(4, 3, 1);$$

$$\text{Request}_4(3, 3, 0) \leq \text{Available}(2, 3, 0), \text{ 让 } P_4 \text{ 等待。}$$

(3) P_0 请求资源: P_0 发出请求向量 $\text{Request}_0(0, 2, 0)$, 系统按银行家算法进行检查:

$$\text{Request}_0(0, 2, 0) \leq \text{Need}_0(7, 4, 3);$$

$$\text{Request}_0(0, 2, 0) \leq \text{Available}(2, 3, 0)$$

系统暂时先假定可为 P_0 分配资源, 并修改有关数据:

$$\text{Available} = \text{Available} - \text{Request}_0 = (2, 1, 0)$$

$$\text{Allocation}_0 = \text{Allocation}_0 + \text{Request}_0 = (0, 3, 0)$$

$$\text{Need}_0 = \text{Need}_0 - \text{Request}_0 = (7, 2, 3), \text{ 结果如表 2.10 所示。}$$

表 2.10 为 P_0 分配资源后的有关资源数据

资源情况 进程	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	3	0	7	2	3	2	1	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

进行安全性检查：可用资源 Available(2, 1, 0)已不能满足任何进程的需要，因此系统进入不安全状态，因此拒绝 P_0 的请求，让 P_0 等待，并将 Available, Allocation₀, Need₀ 恢复为之前的值。

2.4.5 死锁检测和解除

前面介绍的死锁预防和避免算法，都是在为进程分配资源时施加限制条件或进行检测，若系统为进程分配资源时不采取任何措施，则应该提供死锁检测和解除的手段。

1. 资源分配图

系统死锁可利用资源分配图来描述。如图 2.13 所示，用圆圈代表一个进程，用框代表一类资源。由于一种类型的资源可能有多个，因此用框中的一个圆代表一类资源中的一个资源。从进程到资源的有向边称为请求边，表示该进程申请一个单位的该类资源；从资源到进程的边称为分配边，表示该类资源已有一个资源分配给了该进程。

在图 2.13 所示的资源分配图中，进程 P_1 已经分得了两个 R_1 资源，并又请求一个 R_2 资源；进程 P_2 分得了一个 R_1 资源和一个 R_2 资源，并又请求一个 R_1 资源。

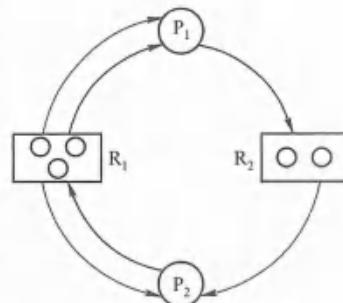


图 2.13 资源分配示例

2. 死锁定理

简化资源分配图可检测系统状态 S 是否为死锁状态。简化方法如下：

- 1) 在资源分配图中，找出既不阻塞又不孤点的进程 P_i （即找出一条有向边与它相连，且该有向边对应资源的申请数量小于等于系统中已有的空闲资源数量，如在图 2.13 中， R_1 没有空闲资源， R_2 有一个空闲资源。若所有连接该进程的边均满足上述条件，则这个进程能继续运行直至完成，然后释放它所占有的所有资源）。消去它所有的请求边和分配边，使之成为孤立的结点。在图 2.14(a)中， P_1 是满足这一条件的进程结点，将 P_1 的所有边消去，便得到图 2.14(b)所示的情况。

这里要注意一个问题，判断某种资源是否有空间，应用它的资源数量减去它在资源分配图中的出度，例如在图 2.13 中， R_1 的资源数为 3，而出度也为 3，所以 R_1 没有空闲资源， R_2 的资源数为 2，出度为 1，所以 R_2 有一个空闲资源。

- 2) 进程 P_i 所释放的资源，可以唤醒某些因等待这些资源而阻塞的进程，原来的阻塞进程可能变为非阻塞进程。在图 2.13 中，进程 P_2 就满足这样的条件。根据 1) 中的方法进行一系列简化后，若能消去图中所有的边，则称该图是可完全简化的，如图 2.14(c)所示。

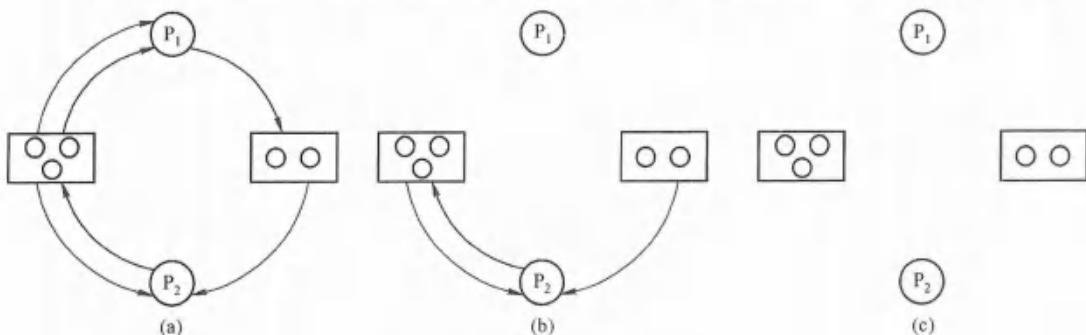


图 2.14 资源分配图的化简

S 为死锁的条件是当且仅当 S 状态的资源分配图是不可完全简化的，该条件为死锁定理。

3. 死锁解除

一旦检测出死锁，就应立即采取相应的措施来解除死锁。死锁解除的主要方法有：

- 1) 资源剥夺法。挂起某些死锁进程，并抢占它的资源，将这些资源分配给其他的死锁进程。但应防止被挂起的进程长时间得不到资源而处于资源匮乏的状态。
- 2) 撤销进程法。强制撤销部分甚至全部死锁进程并剥夺这些进程的资源。撤销的原则可以按进程优先级和撤销进程代价的高低进行。
- 3) 进程回退法。让一（或多）个进程回退到足以回避死锁的地步，进程回退时自愿释放资源而非被剥夺。要求系统保持进程的历史信息，设置还原点。

2.4.6 本节小结

本节开头提出的问题的参考答案如下。

- 1) 为什么会产生死锁？产生死锁有什么条件？

由于系统中存在一些不可剥夺资源，当两个或两个以上的进程占有自身的资源并请求对方的资源时，会导致每个进程都无法向前推进，这就是死锁。死锁产生的必要条件有 4 个，分别是互斥条件、不剥夺条件、请求并保持条件和循环等待条件。

互斥条件是指进程要求分配的资源是排他性的，即最多只能同时供一个进程使用。

不剥夺条件是指进程在使用完资源之前，资源不能被强制夺走。

请求并保持条件是指进程占有自身本来拥有的资源并要求其他资源。

循环等待条件是指存在一种进程资源的循环等待链。

- 2) 有什么办法可以解决死锁问题？

死锁的处理策略可以分为预防死锁、避免死锁及死锁的检测与解除。

死锁的预防是指通过设立一些限制条件，破坏死锁的一些必要条件，让死锁无法发生。

死锁的避免是指在动态分配资源的过程中，用一些算法防止系统进入不安全状态，从而避免死锁。

死锁的检测和解除是指在死锁产生前不采取任何措施，只检测当前系统有没有发生死锁，若有，则采取一些措施解除死锁。

2.4.7 本节习题精选

一、单项选择题

1. 下列情况中，可能导致死锁的是（ ）。

A. 进程释放资源	B. 一个进程进入死循环
C. 多个进程竞争资源出现了循环等待	D. 多个进程竞争使用共享型的设备
2. 在操作系统中，死锁出现是指（ ）。

A. 计算机系统发生重大故障	B. 资源个数远远小于进程数
C. 若干进程因竞争资源而无限等待其他进程释放已占有的资源	D. 进程同时申请的资源数超过资源总数
3. 一次分配所有资源的方法可以预防死锁的发生，它破坏死锁 4 个必要条件中的（ ）。

A. 互斥	B. 占有并请求	C. 非剥夺	D. 循环等待
-------	----------	--------	---------

4. 系统产生死锁的可能原因是()。
 A. 独占资源分配不当 B. 系统资源不足
 C. 进程运行太快 D. CPU 内核太多
5. 死锁的避免是根据()采取措施实现的。
 A. 配置足够的系统资源 B. 使进程的推进顺序合理
 C. 破坏死锁的四个必要条件之一 D. 防止系统进入不安全状态
6. 死锁预防是保证系统不进入死锁状态的静态策略，其解决办法是破坏产生死锁的四个必要条件之一。下列方法中破坏了“循环等待”条件的是()。
 A. 银行家算法 B. 一次性分配策略
 C. 剥夺资源法 D. 资源有序分配策略
7. 某系统中有三个并发进程都需要四个同类资源，则该系统必然不会发生死锁的最少资源是()。
 A. 9 B. 10 C. 11 D. 12
8. 某系统中共有 11 台磁带机， X 个进程共享此磁带机设备，每个进程最多请求使用 3 台，则系统必然不会死锁的最大 X 值是()。
 A. 4 B. 5 C. 6 D. 7
9. 【2009 统考真题】某计算机系统中有 8 台打印机，由 K 个进程竞争使用，每个进程最多需要 3 台打印机。该系统可能会发生死锁的 K 的最小值是()。
 A. 2 B. 3 C. 4 D. 5
10. 解除死锁通常不采用的方法是()。
 A. 终止一个死锁进程 B. 终止所有死锁进程
 C. 从死锁进程处抢夺资源 D. 从非死锁进程处抢夺资源
11. 采用资源剥夺法可以解除死锁，还可以采用()方法解除死锁。
 A. 执行并行操作 B. 撤销进程
 C. 拒绝分配新资源 D. 修改信号量
12. 在下列死锁的解决方法中，属于死锁预防策略的是()。
 A. 银行家算法 B. 资源有序分配算法
 C. 死锁检测算法 D. 资源分配图化简法
13. 引入多道程序技术的前提条件之一是系统具有()。
 A. 多个 CPU B. 多个终端 C. 中断功能 D. 分时功能
14. 三个进程共享四个同类资源，这些资源的分配与释放只能一次一个。已知每个进程最多需要两个该类资源，则该系统()。
 A. 有些进程可能永远得不到该类资源 B. 必然有死锁
 C. 进程请求该类资源必然能得到 D. 必然是死锁
15. 以下有关资源分配图的描述中，正确的是()。
 A. 有向边包括进程指向资源类的分配边和资源类指向进程申请边两类
 B. 矩形框表示进程，其中圆点表示申请同一类资源的各个进程
 C. 圆圈结点表示资源类
 D. 资源分配图是一个有向图，用于表示某时刻系统资源与进程之间的状态
16. 死锁的四个必要条件中，无法破坏的是()。
 A. 环路等待资源 B. 互斥使用资源

- C. 占有且等待资源 D. 非抢夺式分配
17. 死锁与安全状态的关系是()。
 A. 死锁状态有可能是安全状态
 C. 不安全状态就是死锁状态
18. 死锁检测时检查的是()。
 A. 资源有向图 B. 前驱图 C. 搜索树 D. 安全图
19. 某个系统采用下列资源分配策略。若一个进程提出资源请求得不到满足，而此时没有由于等待资源而被阻塞的进程，则自己就被阻塞。而当此时已有等待资源而被阻塞的进程，则检查所有由于等待资源而被阻塞的进程。若它们有申请进程所需要的资源，则将这些资源取出并分配给申请进程。这种分配策略会导致()。
 A. 死锁 B. 颠簸 C. 回退 D. 饥饿
20. 系统的资源分配图在下列情况下，无法判断是否处于死锁状态的有()。
 I. 出现了环路 II. 没有环路
 III. 每种资源只有一个，并出现环路 IV. 每个进程结点至少有一条请求边
 A. I、II、III、IV B. I、III、IV
 C. I、IV D. 以上答案都不正确
21. 下列关于死锁的说法中，正确的有()。
 I. 死锁状态一定是不安全状态
 II. 产生死锁的根本原因是系统资源分配不足和进程推进顺序非法
 III. 资源的有序分配策略可以破坏死锁的循环等待条件
 IV. 采用资源剥夺法可以解除死锁，还可以采用撤销进程方法解除死锁
 A. I、III B. II C. IV D. 四个说法都对
22. 下面是一个并发进程的程序代码，正确的是()。
- ```

Semaphore x1=x2=y=1;
Int c1=c2=0;
P1()
{
 while(1){
 P(x1);
 if(++c1==1) P(y);
 V(x1);
 computer(A);
 P(x1);
 if(--c1==0) V(y);
 V(x1);
 }
}
P2()
{
 while(1){
 P(x2);
 if(++c2==1) P(y);
 V(x2);
 computer(B);
 P(x2);
 if(--c2==0) V(y);
 V(x2);
 }
}

```
- A. 进程不会死锁，也不会“饥饿”                      B. 进程不会死锁，但是会“饥饿”  
 C. 进程会死锁，但是不会“饥饿”                      D. 进程会死锁，也会“饥饿”
23. 有两个并发进程，对于如下这段程序的运行，正确的说法是( )。
- ```

int x,y,z,t,u;
P1()
{
  while(1){
    x=1;
    ...
}
P2()
{
  while(1){
    x=0;
    ...
}
  
```

```

y=0;
if x>=1 then y=y+1;
z=y;
}
}
t=0
if x<=1 then t=t+2;
u=t;
}
}

```

- A. 程序能正确运行，结果唯一 B. 程序不能正确运行，可能有两种结果
 C. 程序不能正确运行，结果不确定 D. 程序不能正确运行，可能会死锁
24. 一个进程在获得资源后，只能在使用完资源后由自己释放，这属于死锁必要条件的()。
 A. 互斥条件 B. 请求和释放条件
 C. 不剥夺条件 D. 防止系统进入不安全状态
25. 死锁定理是用于处理死锁的()方法。
 A. 预防死锁 B. 避免死锁 C. 检测死锁 D. 解除死锁
26. 【2013 统考真题】下列关于银行家算法的叙述中，正确的是()。
 A. 银行家算法可以预防死锁
 B. 当系统处于安全状态时，系统中一定无死锁进程
 C. 当系统处于不安全状态时，系统中一定会出现死锁进程
 D. 银行家算法破坏了死锁必要条件中的“请求和保持”条件
27. 假设具有 5 个进程的进程集合 $P = \{P_0, P_1, P_2, P_3, P_4\}$ ，系统中有三类资源 A, B, C，假设在某时刻有如下状态，见下表。

	Allocation			Max			Available
	A	B	C	A	B	C	
P_0	0	0	3	0	0	4	
P_1	1	0	0	1	7	5	
P_2	1	3	5	2	3	5	
P_3	0	0	2	0	6	4	
P_4	0	0	1	0	6	5	

A B C
x y z

- 请问当 x, y, z 取下列哪些值时，系统是处于安全状态的？
- I. 1, 4, 0 II. 0, 6, 2 III. 1, 1, 1 IV. 0, 4, 7
 A. II、III B. I、II C. 仅 I D. I、III
28. 【2011 统考真题】某时刻进程的资源使用情况见下表，此时的安全序列是()。
 A. P_1, P_2, P_3, P_4 B. P_1, P_3, P_2, P_4 C. P_1, P_4, P_3, P_2 D. 不存在

进程	已分配资源			尚需分配			可用资源		
	R ₁	R ₂	R ₃	R ₁	R ₂	R ₃	R ₁	R ₂	R ₃
P_1	2	0	0	0	0	1	0	2	1
P_2	1	2	0	1	3	2			
P_3	0	1	1	1	3	1			
P_4	0	0	1	2	0	0			

29. 【2012 统考真题】假设 5 个进程 P_0, P_1, P_2, P_3, P_4 共享三类资源 R₁, R₂, R₃，这些资源总数分别为 18, 6, 22。 T_0 时刻的资源分配情况如下表所示，此时存在的一个安全序列是()。

进程	已分配资源			资源最大需求		
	R ₁	R ₂	R ₃	R ₁	R ₂	R ₃
P ₀	3	2	3	5	5	10
P ₁	4	0	3	5	3	6
P ₂	4	0	5	4	0	11
P ₃	2	0	4	4	2	5
P ₄	3	1	4	4	2	4

- A. P₀, P₂, P₄, P₁, P₃ B. P₁, P₀, P₃, P₄, P₂ C. P₂, P₁, P₀, P₃, P₄ D. P₃, P₄, P₂, P₁, P₀
30. 【2014 统考真题】某系统有 n 台互斥使用的同类设备，三个并发进程分别需要 3, 4, 5 台设备，可确保系统不发生死锁的设备数 n 最小为（ ）。
- A. 9 B. 10 C. 11 D. 12
31. 【2016 统考真题】系统中有 3 个不同的临界资源 R₁, R₂ 和 R₃，被 4 个进程 P₁, P₂, P₃, P₄ 共享。各进程对资源的需求为：P₁ 申请 R₁ 和 R₂, P₂ 申请 R₂ 和 R₃, P₃ 申请 R₁ 和 R₃, P₄ 申请 R₂。若系统出现死锁，则处于死锁状态的进程数至少是（ ）。
- A. 1 B. 2 C. 3 D. 4
32. 【2015 统考真题】若系统 S₁ 采用死锁避免方法，S₂ 采用死锁检测方法。下列叙述中，正确的是（ ）。
- I. S₁ 会限制用户申请资源的顺序，而 S₂ 不会
II. S₁ 需要进程运行所需的资源总量信息，而 S₂ 不需要
III. S₁ 不会给可能导致死锁的进程分配资源，而 S₂ 会
- A. 仅 I、II B. 仅 II、III C. 仅 I、III D. I、II、III
33. 【2018 统考真题】假设系统中有 4 个同类资源，进程 P₁, P₂ 和 P₃ 需要的资源数分别为 4, 3 和 1, P₁, P₂ 和 P₃ 已申请到的资源数分别为 2, 1 和 0，则执行安全性检测算法的结果是（ ）。
- A. 不存在安全序列，系统处于不安全状态
B. 存在多个安全序列，系统处于安全状态
C. 存在唯一安全序列 P₃, P₁, P₂，系统处于安全状态
D. 存在唯一安全序列 P₃, P₂, P₁，系统处于安全状态
34. 【2019 统考真题】下列关于死锁的叙述中，正确的是（ ）。
- I. 可以通过剥夺进程资源解除死锁
II. 死锁的预防方法能确保系统不发生死锁
III. 银行家算法可以判断系统是否处于死锁状态
IV. 当系统出现死锁时，必然有两个或两个以上的进程处于阻塞态
- A. 仅 II、III B. 仅 I、II、IV C. 仅 I、II、III D. 仅 I、III、IV
35. 【2020 统考真题】某系统中有 A、B 两类资源各 6 个， t 时刻资源分配及需求情况如下表所示。

进程	A 已分配数量	B 已分配数量	A 需求总量	B 需求总量
P ₁	2	3	4	4
P ₂	2	1	3	1
P ₃	1	2	3	4

t 时刻安全性检测结果是（ ）。

- A. 存在安全序列 P_1, P_2, P_3
 C. 存在安全序列 P_2, P_3, P_1
 B. 存在安全序列 P_2, P_1, P_3
 D. 不存在安全序列

二、综合应用题

1. 设系统中有下述解决死锁的方法：

- 1) 银行家算法。
- 2) 检测死锁，终止处于死锁状态的进程，释放该进程占有的资源。
- 3) 资源预分配。

简述哪种办法允许最大的并发性，即哪种办法允许更多的进程无等待地向前推进。请按“并发性”从大到小对上述三种办法排序。

2. 某银行计算机系统要实现一个电子转账系统，基本业务流程是：首先对转出方和转入方的账户进行加锁，然后进行转账业务，最后对转出方和转入方的账户进行解锁。若不采取任何措施，系统会不会发生死锁？为什么？请设计一个能够避免死锁的办法。
3. 设有进程 P_1 和进程 P_2 并发执行，都需要使用资源 R_1 和 R_2 ，使用资源的情况见下表。

进程 P_1	进程 P_2
申请资源 R_1	申请资源 R_2
申请资源 R_2	申请资源 R_1
释放资源 R_1	释放资源 R_2

试判断是否会发生死锁，并解释和说明产生死锁的原因与必要条件。

4. 系统有同类资源 m 个，供 n 个进程共享，若每个进程对资源的最大需求量为 k ，试问：
 当 m, n, k 的值分别为下列情况时（见下表），是否会发生死锁？

序号	m	n	k	是否会死锁	说明
1	6	3	3		
2	9	3	3		
3	13	6	3		

5. 有三个进程 P_1, P_2 和 P_3 并发工作。进程 P_1 需要资源 S_3 和资源 S_1 ；进程 P_2 需要资源 S_2 和资源 S_1 ；进程 P_3 需要资源 S_3 和资源 S_2 。问：

- 1) 若对资源分配不加限制，会发生什么情况？为什么？
 - 2) 为保证进程正确运行，应采用怎样的分配策略？列出所有可能的方法。
6. 某系统有 R_1, R_2 和 R_3 共三种资源，在 T_0 时刻 P_1, P_2, P_3 和 P_4 这四个进程对资源的占用和需求情况见下表，此时系统的可用资源向量为 $(2, 1, 2)$ 。试问：
- 1) 用向量或矩阵表示系统中各种资源的总数和此刻各进程对各资源的需求数目。
 - 2) 若此时进程 P_1 和进程 P_2 均发出资源请求向量 $Request(1, 0, 1)$ ，为了保证系统的安全性，应如何分配资源给这两个进程？说明所采用策略的原因。

进程	资源情况			最大资源需求量		已分配资源数量		
	R_1	R_2	R_3	R_1	R_2	R_3		
P_1	3	2	2	1	0	0		
P_2	6	1	3	4	1	1		
P_3	3	1	4	2	1	1		
P_4	4	2	2	0	0	2		

3) 若 2) 中两个请求立即得到满足后, 系统此刻是否处于死锁状态?

7. 考虑某个系统在下表时刻的状态。

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	0	0	1	2	0	0	1	2	1	5	2	0
P ₁	1	0	0	0	1	7	5	0				
P ₂	1	3	5	4	2	3	5	6				
P ₃	0	0	1	4	0	6	5	6				

使用银行家算法回答下面的问题:

1) Need 矩阵是怎样的?

2) 系统是否处于安全状态? 如安全, 请给出一个安全序列。

3) 若从进程 P₁发来一个请求(0, 4, 2, 0), 这个请求能否立刻被满足? 如安全, 请给出一个安全序列。

8. 假设具有 5 个进程的进程集合 P = {P₀, P₁, P₂, P₃, P₄}, 系统中有三类资源 A, B, C, 假设在某时刻有如下状态:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P ₀	0	0	3	0	0	4	1	4	0
P ₁	1	0	0	1	7	5			
P ₂	1	3	5	2	3	5			
P ₃	0	0	2	0	6	4			
P ₄	0	0	1	0	6	5			

请问当前系统是否处于安全状态? 若系统中的可利用资源 Available 为(0, 6, 2), 系统是否安全? 若系统处在安全状态, 请给出安全序列; 若系统处在非安全状态, 简要说明原因。

9. 假定某计算机系统有 R₁ 和 R₂ 两类可使用资源 (其中 R₁ 有两个单位, R₂ 有一个单位), 它们被进程 P₁ 和 P₂ 所共享, 且已知两个进程均以下列顺序使用两类资源: 申请 R₁ → 申请 R₂ → 申请 R₁ → 释放 R₁ → 释放 R₂ → 释放 R₁。试求出系统运行过程中可能到达的死锁点, 并画出死锁点的资源分配图 (或称进程资源图)。

2.4.8 答案与解析

一、单项选择题

1. C

引起死锁的 4 个必要条件是: 互斥、占有并等待、非剥夺和循环等待。本题中, 出现了循环等待的现象, 意味着可能会导致死锁。进程释放资源不会导致死锁, 进程自己进入死循环只能产生“饥饿”, 不涉及其他进程。共享型设备允许多个进程申请使用, 因此不会造成死锁。再次提醒, 死锁一定要有两个或两个以上的进程才会导致, 而饥饿可能由一个进程导致。

2. C

死锁是指多个进程因竞争系统资源或相互通信而处于永久阻塞态, 若无外力作用, 这些进程都将无法推进。

3. B

发生死锁的 4 个必要条件: 互斥、占有并请求、非剥夺和循环等待。一次分配所有资源的方

法是当进程需要资源时，一次性提出所有的请求，若请求的所有资源均满足则分配，只要有一项不满足，就不分配任何资源，该进程阻塞，直到所有的资源空闲后，满足进程的所有需求时再分配。这种分配方式不会部分地占有资源，因此打破了死锁的4个必要条件之一，实现了对死锁的预防。但是，这种分配方式需要凑齐所有资源，因此当一个进程所需的资源较多时，资源的利用率会较低，甚至会造成进程“饥饿”。

4. A

系统死锁的可能原因主要是时间上和空间上的。时间上由于进程运行中推进顺序不当，即调度时机不合适，不该切换进程时进行了切换，可能会造成死锁；空间上的原因是对独占资源分配不当，互斥资源部分分配又不可剥夺，极易造成死锁。那么，为什么系统资源不足不是造成死锁的原因呢？系统资源不足只会对进程造成“饥饿”。例如，某系统只有三台打印机，若进程运行中要申请四台，显然不能满足，该进程会永远等待下去。若该进程在创建时便声明需要四台打印机，则操作系统立即就会拒绝，这实际上是资源分配不当的一种表现。不能以系统资源不足来描述剩余资源不足的情形。

5. D

死锁避免是指在资源动态分配过程中用某些算法加以限制，防止系统进入不安全状态从而避免死锁的发生。选项B是避免死锁后的结果，而不是措施的原理。

6. D

资源有序分配策略可以限制循环等待条件的发生。选项A判断是否为不安全状态；选项B破坏了占有请求条件；选项C破坏了非剥夺条件。

7. B

资源数为9时，存在三个进程都占有三个资源，为死锁；资源数为10时，必然存在一个进程能拿到4个资源，然后可以顺利执行完其他进程。

8. B

考虑一下极端情况：每个进程已经分配了两台磁带机，那么其中任何一个进程只要再分配一台磁带机即可满足它的最大需求，该进程总能运行下去直到结束，然后将磁带机归还给系统再次分配给其他进程使用。因此，系统中只要满足 $2X+1=11$ 这个条件即可认为系统不会死锁，解得 $X=5$ ，也就是说，系统中最多可以并发5个这样的进程是不会死锁的。

9. C

这种题要用到组合数学中鸽巢原理的思想。考虑最极端的情况，因为每个进程最多需要3台打印机，若每个进程已经占有了2台打印机，则只要还有多的打印机，总能满足一个进程达到3台的条件，然后顺利执行，所以将8台打印机分给K个进程，每个进程有2台打印机，这个情况就是极端情况，K为4。

10. D

解除死锁的方法有，①剥夺资源法：挂起某些死锁进程，并抢占它的资源，将这些资源分配给其他的死锁进程；②撤销进程法：强制撤销部分甚至全部死锁进程并剥夺这些进程的资源。

11. B

资源剥夺法允许一个进程强行剥夺其他进程所占有的系统资源。而撤销进程强行释放一个进程已占有的系统资源，与资源剥夺法同理，都通过破坏死锁的“请求和保持”条件来解除死锁。拒绝分配新资源只能维持死锁的现状，无法解除死锁。

12. B

其中，银行家算法为死锁避免算法，死锁检测算法和资源分配图化简法为死锁检测，根据排

除法可以得出资源有序分配算法为死锁预防策略。

13. C

多道程序技术要求进程间能实现并发，需要实现进程调度以保证 CPU 的工作效率，而并发性的实现需要中断功能的支持。

14. C

不会发生死锁。因为每个进程都分得一个资源时，还有一个资源可以让任意一个进程满足，这样这个进程可以顺利运行完成进而释放它的资源。

15. D

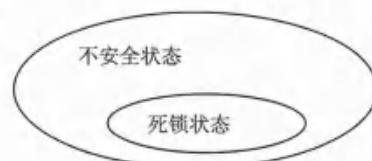
进程指向资源的有向边称为申请边，资源指向进程的有向边称为分配边，A 选项张冠李戴；矩形框表示资源，其中的圆点表示资源的数目，选项 B 错；圆圈结点表示进程，选项 C 错；选项 D 的说法是正确的。

16. B

所谓破坏互斥使用资源，是指允许多个进程同时访问资源，但有些资源根本不能同时访问，如打印机只能互斥使用。因此，破坏互斥条件而预防死锁的方法不太可行，而且在有的场合应该保护这种互斥性。其他三个条件都可以实现。

17. D

如右图所示，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；死锁状态必定是不安全状态。



18. A

死锁检测一般采用两种方法：资源有向图法和资源矩阵法。前驱图只是说明进程之间的同步关系，搜索树用于数据结构的分析，安全图并不存在。注意死锁避免和死锁检测的区别：死锁避免是指避免死锁发生，即死锁没有发生；死锁检测是指死锁已出现，要把它检测出来。

19. D

某个进程主动释放资源不会导致死锁，因为破坏了请求并保持条件，选项 A 错。

颠簸也就是抖动，这是请求分页系统中页面调度不当而导致的现象，是下一章讨论的问题，这里权且断定选项 B 是错的。

回退是指从此时此刻的状态退回到一分钟之前的状态，假如一分钟之前拥有资源 X，它有可能释放了资源 X，那就不称回到一分钟之前的状态，也就不是回退，选项 C 错。

由于进程过于“慷慨”，不断把自己已得到的资源送给别人，导致自己长期无法完成，所以是饥饿，选项 D 对。

20. C

出现了环路，只是满足了循环等待的必要条件，而满足必要条件不一定会导致死锁，I 对；没有环路，破坏了循环等待条件，一定不会发生死锁，II 错；每种资源只有一个，又出现了环路，这是死锁的充分条件，可以确定是否有死锁，III 错；即使每个进程至少有一条请求边，若资源足够，则不会发生死锁，但若资源不充足，则有发生死锁的可能，IV 对。

综上所述，选择 C 选项。

21. D

I 正确：见 17 题答案解析图。

II 正确：这是产生死锁的两大原因。

III 正确：在对资源进行有序分配时，进程间不可能出现环形链，即不会出现循环等待。

IV 正确：资源剥夺法允许一个进程强行剥夺其他进程占有的系统资源。而撤销进程强行释放一个进程已占有的系统资源，与资源剥夺法同理，都通过破坏死锁的“请求和保持”条件来解除死锁，所以选择 D 选项。

22. B

遇到这种问题时千万不要慌张，下面我们来慢慢分析，给读者一个清晰的解题过程：

① 假设 P_1 进程稍快， P_2 进程稍慢，同时运行；② P_1 进程首先进入 if 条件语句，因此获得了 y 的互斥访问权， P_2 被阻塞；③ 在第一个 P_1 进程未释放 y 之前，又有另一个 P_1 进入，c1 的值变成 2，当第一个 P_1 离开时， P_2 仍然被阻塞，这种情形不断发生；④ 在这种情况下会发生什么事？ P_1 顺利执行， P_2 很郁闷，长期被阻塞。

综上所述，不会发生死锁，但会出现饥饿现象。选择 B 选项。

23. C

本题中两个进程不能正确地工作，运行结果的可能性详见下面的说明。

- | | |
|------------------------------------|------------------------------------|
| 1. $x = 1;$ | 5. $x = 0;$ |
| 2. $y = 0;$ | 6. $t = 0$ |
| 3. If $x \geq 1$ then $y = y + 1;$ | 7. if $x \leq 1$ then $t = t + 2;$ |
| 4. $z = y;$ | 8. $u = t;$ |

不确定的原因是由于使用了公共变量 x，考查程序中与变量 x 有关的语句共四处，执行的顺序是 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$ 时，结果是 $y=1, z=1, t=2, u=2, x=0$ ；并发执行过程是 $1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 8$ 时，结果是 $y=0, z=0, t=2, u=2, x=0$ ；执行的顺序是 $5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ 时，结果是 $y=1, z=1, t=2, u=2, x=1$ ；执行的顺序是 $5 \rightarrow 6 \rightarrow 1 \rightarrow 2 \rightarrow 7 \rightarrow 8 \rightarrow 3 \rightarrow 4$ 时，结果是 $y=1, z=1, t=2, u=2, x=1$ 。可见结果有多种可能性。

很明显，无论执行顺序如何，x 的结果只能是 0 或 1，因此语句 7 的条件一定成立，即 $t=u=2$ 的结果是一定的；而 $y=z$ 必定成立，只可能有 0, 1 两种情况，又不可能出现 $x=1, y=z=0$ 的情况，所以总共只有 3 种结果（答案中的 3 种）。

24. C

一个进程在获得资源后，只能在使用完资源后由自己释放，即它的资源不能被系统剥夺，答案为 C 选项。

25. C

死锁定理是用于检测死锁的方法。

26. B

银行家算法是避免死锁的方法，选项 A、D 错。

根据 17 题的答案解析图，选项 B 对，选项 C 错。

27. C

$$\text{Need} = \text{Max} - \text{Allocation} = \begin{bmatrix} 0 & 0 & 4 \\ 1 & 7 & 5 \\ 2 & 3 & 5 \\ 0 & 6 & 4 \\ 0 & 6 & 5 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 3 \\ 1 & 0 & 0 \\ 1 & 3 & 5 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 7 & 5 \\ 1 & 0 & 0 \\ 0 & 6 & 2 \\ 0 & 6 & 4 \end{bmatrix}$$

I：根据 need 矩阵可知，当 Available 为(1, 4, 0)时，可满足 P_2 的需求； P_2 结束后释放资源，Available 为(2, 7, 5)可以满足 P_0, P_1, P_3, P_4 中任一进程的需求，所以系统不会出现死锁，处于安全

状态。

II: 当 Available 为(0, 6, 2)时, 可以满足进程 P_0 , P_3 的需求; 这两个进程结束后释放资源, Available 为(0, 6, 7), 仅可以满足进程 4 的需求; P_4 结束并释放后, Available 为(0, 6, 8), 此时不能满足余下任一进程的需求, 系统出现死锁, 因此当前处在非安全状态。

III: 当 Available 为(1, 1, 1)时, 可以满足进程 P_0 , P_2 的需求; 这两个进程结束后释放资源, Available 为(2, 4, 9), 此时不能满足余下任一进程的需求, 系统出现死锁, 处于非安全状态。

IV: 当 Available 为(0, 4, 7)时, 可以满足 P_0 的需求, 进程结束后释放资源, Available 为(0, 4, 10), 此时不能满足余下任一进程的需求, 系统出现死锁, 处于非安全状态。

综上分析: 只有 I 处于安全状态。

28. D

本题应采用排除法, 逐个代入分析。剩余资源分配给 P_1 , 待 P_1 执行完后, 可用资源数为(2, 2, 1), 此时仅能满足 P_4 的需求, 排除选项 A、B; 接着分配给 P_4 , 待 P_4 执行完后, 可用资源数为(2, 2, 2), 此时已无法满足任何进程的需求, 排除选项 C。

此外, 本题还可以使用银行家算法求解 (对选择题来说显得过于复杂)。

29. D

首先求得各进程的需求矩阵 Need 与可利用资源向量 Available:

进程	Need			Available			
	R ₁	R ₂	R ₃		R ₁	R ₂	R ₃
P_0	2	3	7				
P_1	1	3	3				
P_2	0	0	6				
P_3	2	2	1				
P_4	1	1	0	2	3	3	

比较 Need 和 Available 发现, 初始时进程 P_1 与 P_3 可满足需求, 排除 A、C。尝试给 P_1 分配资源时, P_1 完成后 Available 将变为(6, 3, 6), 无法满足 P_0 的需求, 排除 B。尝试给 P_3 分配资源时, P_3 完成后 Available 将变为(4, 3, 7), 该向量能满足其他所有进程的需求。因此, 以 P_3 开头的所有序列都是安全序列。

30. B

三个并发进程分别需要 3, 4, 5 台设备, 当系统只有 $(3 - 1) + (4 - 1) + (5 - 1) = 9$ 台设备时, 第一个进程分配 2 台, 第二个进程分配 3 台, 第三个进程分配 4 台。这种情况下, 三个进程均无法继续执行下去, 发生死锁。当系统中再增加 1 台设备, 即共 10 台设备时, 最后 1 台设备分配给任意一个进程都可以顺利执行完成, 因此保证系统不发生死锁的最小设备数为 10。

31. C

对于本题, 先满足一个进程的资源需求, 再看其他进程是否出现死锁状态。因为 P_4 只申请一个资源, 当将 R_2 分配给 P_4 后, P_4 执行完后将 R_2 释放, 这时使得系统满足死锁的条件是 R_1 分配给 P_1 , R_2 分配给 P_2 , R_3 分配给 P_3 (或者 R_2 分配给 P_1 , R_3 分配给 P_2 , R_1 分配给 P_3)。穷举其他情况如 P_1 申请的资源 R_1 和 R_2 , 先都分配给 P_1 , 运行完并释放占有的资源后, 可分别将 R_1 , R_2 和 R_3 分配给 P_3 , P_4 和 P_2 , 也满足系统死锁的条件。各种情况需要使得处于死锁状态的进程数至少为 3。

32. B

死锁的处理采用三种策略: 死锁预防、死锁避免、死锁检测和解除。

死锁预防采用破坏产生死锁的4个必要条件中的一个或几个来防止发生死锁。其中之一的“破坏循环等待条件”，一般采用顺序资源分配法，首先给系统的资源编号，规定每个进程必须按编号递增的顺序请求资源，即限制了用户申请资源的顺序，因此I的前半句属于死锁预防的范畴。

银行家算法是最著名的死锁避免算法，其中的最大需求矩阵Max定义了每个进程对m类资源的最大需求量，系统在执行安全性算法中都会检查此次资源试分配后，系统是否处于安全状态，若不安全则将本次的试探分配作废。在死锁的检测和解除中，系统为进程分配资源时不采取任何措施，但提供死锁的检测和解除手段，因此II、III正确。

33. A

由题意可知，仅剩最后一个同类资源，若将其分给P₁或P₂，则均无法正常执行；若分给P₃，则P₃正常执行完成后，释放的这一个资源仍无法使P₁、P₂正常执行，因此不存在安全序列。

34. B

剥夺进程资源，将其分配给其他死锁进程，可以解除死锁，I正确。死锁预防是死锁处理策略（死锁预防、死锁避免、死锁检测）中最为严苛的一种策略，破坏死锁产生的4个必要条件之一，可以确保系统不发生死锁，II正确。银行家算法是一种死锁避免算法，用于计算动态资源分配的完全性以避免系统进入死锁状态，不能用于判断系统是否处于死锁，III错误。通过简化资源分配图可以检测系统是否为死锁状态，当系统出现死锁时，资源分配图不可完全简化，只有两个或两个以上的进程才会出现“环”而不能被简化，IV正确。

35. B

首先求出需求矩阵：

$$\text{Need} = \text{Max} - \text{Allocation} = \begin{bmatrix} A & B \\ 4 & 4 \\ 3 & 1 \\ 3 & 4 \end{bmatrix} - \begin{bmatrix} A & B \\ 2 & 3 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} A & B \\ 2 & 1 \\ 1 & 0 \\ 2 & 2 \end{bmatrix}$$

由Allocation得知当前Available为(1, 0)。由需求矩阵可知，初始只能满足P₂的需求，A错误。P₂释放资源后Available变为(3, 1)，此时仅能满足P₁的需求，C错误。P₁释放资源后Available变为(5, 4)，可以满足P₃的需求，得到的安全序列为P₂, P₁, P₃，B正确，D错误。

二、综合应用题

1. 分析：

死锁在系统中不可能完全消灭，但我们要尽可能地减少死锁的发生。对死锁的处理有4种方法：忽略、检测与恢复、避免和预防，每种方法对死锁的处理从宽到严，同时系统并发性由大到小。这里银行家算法属于避免死锁，资源预分配属于预防死锁。

解答：

死锁检测方法可以获得最大的并发性。并发性排序：死锁检测方法、银行家算法、资源预分配法。

2. 解答：

系统会死锁。因为对两个账户进行加锁操作是可以分割进行的，若此时有两个用户同时进行转账，P₁先对账户A进行加锁，再申请账户B；P₂先对账户B进行加锁，再申请账户A，此时产生死锁。解决的办法是：可以采用资源顺序分配法对A、B账户进行编号，用户转账时只能按照编号由小到大进行加锁；也可采用资源预分配法，要求用户在使用资源前将所有资源一次性申请到。

3. 解答：

这段程序在不同的运行推进速度下，可能会产生死锁。例如，进程P₁先申请资源R₁，得到

资源 R_1 , 然后进程 P_2 申请资源 R_2 , 得到资源 R_2 , 进程 P_1 又申请资源 R_2 , 因资源 R_2 已分配, 使得进程 P_1 阻塞。进程 P_1 和进程 P_2 都因申请不到资源而形成死锁。若改变进程的运行顺序, 则这两个进程就不会出现死锁现象。

产生死锁的原因可归结为两点:

- 1) 竞争资源。
- 2) 进程推进顺序非法。

产生死锁的必要条件:

- 1) 互斥条件。
- 2) 请求并保持条件。
- 3) 不剥夺条件。
- 4) 环路等待条件。

4. 解答:

不发生死锁要求, 必须保证至少有一个进程能得到所需的全部资源并执行完毕, $m \geq n(k-1)+1$ 时, 一定不会发生死锁。

序号	m	n	k	是否会死锁	说 明
1	6	3	3	可能会	$6 < 3(3-1)+1$
2	9	3	3	不会	$9 > 3(3-1)+1$
3	13	6	3	不会	$13 = 6(3-1)+1$

5. 解答:

1) 可能会发生死锁。满足发生死锁的 4 大条件, 例如, P_1 占有 S_1 申请 S_3 , P_2 占有 S_2 申请 S_1 , P_3 占有 S_3 申请 S_2 。

2) 可有以下几种答案:

- A. 采用静态分配: 由于执行前已获得所需的全部资源, 因此不会出现占有资源又等待别的资源的现象 (或不会出现循环等待资源的现象)。
- B. 采用按序分配: 不会出现循环等待资源的现象。
- C. 采用银行家算法: 因为在分配时, 保证了系统处于安全状态。

6. 解答:

1) 系统中资源总量为某时刻系统中可用资源量与各进程已分配资源量之和, 即 $(2, 1, 2) + (1, 0, 0) + (4, 1, 1) + (2, 1, 1) + (0, 0, 2) = (9, 3, 6)$, 各进程对资源的需求量为各进程对资源的最大需求量与进程已分配资源量之差, 即

$$\begin{bmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 1 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 0 & 2 \\ 1 & 0 & 3 \\ 4 & 2 & 0 \end{bmatrix}$$

2) 若此时 P_1 发出资源请求 $\text{Request}_1(1, 0, 1)$, 则按银行家算法进行检查:

$$\text{Request}_1(1, 0, 1) \leq \text{Need}_1(2, 2, 2)$$

$$\text{Request}_1(1, 0, 1) \leq \text{Available}(2, 1, 2)$$

试分配并修改相应数据结构, 由此形成的进程 P_1 请求资源后的资源分配情况见下表。

进程 资源情况	Allocation			Need			Available		
	P ₁	2	0	1	1	2	1		
P ₂	4	1	1	2	0	2			
P ₃	2	1	1	1	0	3			
P ₄	0	0	2	4	2	0			

再利用安全性算法检查系统是否安全，可用资源 Available(1, 1, 1) 已不能满足任何进程，系统进入不安全状态，此时系统不能将资源分配给进程 P₁。

若此时进程 P₂ 发出资源请求 Request₂(1, 0, 1)，则按银行家算法进行检查：

$$\text{Request}_2(1, 0, 1) \leq \text{Need}_2(2, 0, 2)$$

$$\text{Request}_2(1, 0, 1) \leq \text{Available}(2, 1, 2)$$

试分配并修改相应数据结构，由此形成的进程 P₂ 请求资源后的资源分配情况下表：

进程 资源情况	Allocation			Need			Available		
	P ₁	1	0	0	2	2	2		
P ₂	5	1	2	1	0	1			
P ₃	2	1	1	1	0	3			
P ₄	0	0	2	4	2	0			

再利用安全性算法检查系统是否安全，可得到如下表中所示的安全性检测情况。注意表中各个进程对应的 Work+Allocation 向量表示在该进程释放资源之后更新的 Work 向量。

进程 资源情况	Work			Need			Allocation			9			
	P ₂	1	1	1	1	0	1	5	1	2	6	2	3
P ₃	6	2	3	1	0	3	2	1	1	8	3	4	
P ₄	8	3	4	4	2	0	0	0	2	8	3	6	
P ₁	8	3	6	2	2	2	1	0	0	9	3	6	

从上表中可以看出，此时存在一个安全序列 {P₂, P₃, P₄, P₁}，因此该状态是安全的，可以立即将进程 P₂ 所申请的资源分配给它。

若 2) 中的两个请求立即得到满足，则此刻系统并未立即进入死锁状态，因为这时所有的进程未提出新的资源申请，全部进程均未因资源请求没有得到满足而进入阻塞态。只有当进程提出资源申请且全部进程都进入阻塞态时，系统才处于死锁状态。

7. 解答：

1)

$$\text{Need} = \text{Max} - \text{Allocation} = \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 7 & 5 & 0 \\ 2 & 3 & 5 & 6 \\ 0 & 6 & 5 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 0 & 0 & 0 \\ 1 & 3 & 5 & 4 \\ 0 & 0 & 1 & 4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 7 & 5 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 6 & 4 & 2 \end{bmatrix}$$

2) Work 向量初始化值 = Available(1, 5, 2, 0)。

系统安全性分析：

资源情况 进程	Work				Need				Allocation				Work + Allocation			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	1	5	2	0	0	0	0	0	0	0	1	2	1	5	3	2
P ₂	1	5	3	2	1	0	0	2	1	3	5	4	2	8	8	6
P ₁	2	8	8	6	0	7	5	0	1	0	0	0	3	8	8	6
P ₃	3	8	8	6	0	6	4	2	0	0	1	4	3	8	9	10

因为存在一个安全序列<P₀, P₂, P₁, P₃>, 所以系统处于安全状态。

3)

Request₁(0, 4, 2, 0) < Need₁(0, 7, 5, 0)

Request₁(0, 4, 2, 0) < Available(1, 5, 2, 0)

假设先试着满足进程 P₁ 的这个请求, 则 Available 变为(1, 1, 0, 0)。

系统状态变化见下表:

资源情况 进程	Max				Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	0	0	1	2	0	0	1	2	0	0	0	0	1	1	0	0
P ₁	1	7	5	0	1	4	2	0	0	3	3	0				
P ₂	2	3	5	6	1	3	5	4	1	0	0	2				
P ₃	0	6	5	6	0	0	1	4	0	6	4	2				

再对系统进行安全性分析, 见下表:

资源情况 进程	Work				Need				Allocation				Work + Allocation			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	1	1	0	0	0	0	0	0	0	0	1	2	1	1	1	2
P ₂	1	1	1	2	1	0	0	2	1	3	5	4	2	4	6	6
P ₁	2	4	6	6	0	3	3	0	1	4	2	0	3	8	8	6
P ₃	3	8	8	6	0	6	4	2	0	0	1	4	3	8	9	10

因为存在一个安全序列<P₀, P₂, P₁, P₃>, 所以系统仍处于安全状态。所以进程 P₁ 的这个请求应该马上被满足。

8. 解答:

- 1) 根据 Need 矩阵可知, 初始 Work 等于 Available 为(1, 4, 0), 可以满足进程 P₂ 的需求; 进程 P₂ 结束后释放资源, Work 为(2, 7, 5), 可以满足 P₀, P₁, P₃ 和 P₄ 中任一进程的需求, 所以系统不会出现死锁, 当前处于安全状态。

$$\text{Need} = \text{Max} - \text{Allocation} = \begin{bmatrix} 0 & 0 & 4 \\ 1 & 7 & 5 \\ 2 & 3 & 5 \\ 0 & 6 & 4 \\ 0 & 6 & 5 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 3 \\ 1 & 0 & 0 \\ 1 & 3 & 5 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 7 & 5 \\ 1 & 0 & 0 \\ 0 & 6 & 2 \\ 0 & 6 & 4 \end{bmatrix}$$

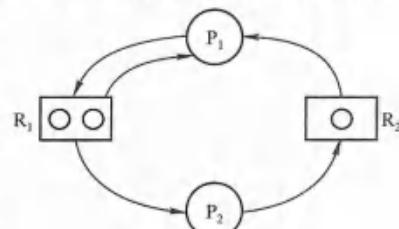
- 2) 若初始 Work = Available 为(0, 6, 2), 可满足进程 P₀, P₃ 的需求; 这两个进程结束后释放资源, Work 为(0, 6, 7), 仅可满足进程 P₄ 的需求; P₄ 结束后释放资源, Work 为(0, 6, 8), 此时不能满足余下任一进程的需求, 系统出现死锁, 因此当前系统处在非安全状态。

注意：在银行家算法中，实际计算分析系统安全状态时，并不需要逐个进程进行。如本题中，在1)的情况下，当计算至进程P₂结束并释放资源时，系统当前空闲资源可满足余下任一进程的最大需求量，这时已经不需要考虑进程的执行顺序。系统分配任意一个进程所需的最大需求资源，在其执行结束释放资源后，系统当前空闲资源会增加，所以余下的进程仍然可以满足最大需求量。因此，在这里可以直接判断系统处于安全状态。在2)的情况下，系统当前可满足进程P₀, P₃的需求，所以可以直接让系统推进到P₀, P₃执行完并释放资源后的情形，这时系统出现死锁；由于此时是系统空闲资源所能达到的最大值，所以按照其他方式推进，系统必然还是出现死锁。因此，在计算过程中，将每步中可满足需求的进程作为一个集合，同时执行并释放资源，可以简化银行家算法的计算。

9. 答解：

在本题中，当两个进程都执行完第一步后，即进程P₁和进程P₂都申请到了一个R₁类资源时，系统进入不安全状态。随着两个进程向前推进，无论哪个进程执行完第二步，系统都将进入死锁状态。可能达到的死锁点是：进程P₁占有一个单位的R₁类资源及一个单位的R₂类资源，进程P₂占有一个单位的R₁类资源，此时系统内已无空闲资源，而两个进程都在保持已占有资源不释放的情况下继续申请资源，从而造成死锁；或进程P₂占有一个单位的R₁类资源和一个单位的R₂类资源，进程P₁占有一个单位的R₁类资源，此时系统内已无空闲资源，而两个进程都在保持已占有资源不释放的情况下继续申请资源，从而造成死锁。

假定进程P₁成功执行了第二步，则死锁点的资源分配如右图所示。



2.5 本章疑难点

1. 进程与程序的区别与联系

- 1) 进程是程序及其数据在计算机上的一次运行活动，是一个动态的概念。进程的运行实体是程序，离开程序的进程没有存在的意义。从静态角度看，进程是由程序、数据和进程控制块（PCB）三部分组成的。而程序是一组有序的指令集合，是一种静态的概念。
- 2) 进程是程序的一次执行过程，它是动态地创建和消亡的，具有一定的生命周期，是暂时存在的；而程序则是一组代码的集合，是永久存在的，可长期保存。
- 3) 一个进程可以执行一个或几个程序，一个程序也可构成多个进程。进程可创建进程，而程序不可能形成新的程序。
- 4) 进程与程序的组成不同。进程的组成包括程序、数据和PCB。

2. 死锁与饥饿

具有等待队列的信号量的实现可能导致这样的情况：两个或多个进程无限地等待一个事件，而该事件只能由这些等待进程之一来产生。这里的事件是V操作的执行（即释放资源）。出现这样的状态时，这些进程称为死锁（Deadlocked）。

为加以说明，考虑一个由两个进程P₀和P₁组成的系统，每个进程都访问两个信号量S和Q，这两个信号量的初值均为1。

```

P0 () {
    while(1) {
        P(S);
        P(Q);
        ...
        V(S);
        V(Q);
    }
}
}

P1 () {
    while(1) {
        P(Q);
        P(S);
        ...
        V(Q);
        V(S);
    }
}
}

```

假设进程 P_0 执行 $P(S)$, 接着进程 P_1 执行 $P(Q)$ 。当进程 P_0 执行 $P(Q)$ 时, 它必须等待, 直到进程 P_1 执行 $V(Q)$ 。类似地, 当进程 P_1 执行 $P(S)$ 时, 它必须等待, 直到进程 P_0 执行 $V(S)$ 。由于这两个 V 操作都不能执行, 因此进程 P_0 和进程 P_1 就死锁了。

一组进程处于死锁状态是指组内的每个进程都在等待一个事件, 而该事件只可能由组内的另一个进程产生。这里所关心的主要是事件是资源的获取和释放。

与死锁相关的另一个问题是无限期阻塞 (Indefinite Blocking) 或饥饿 (Starvation), 即进程在信号量内无穷等待的情况。

产生饥饿的主要原因是: 在一个动态系统中, 对于每类系统资源, 操作系统需要确定一个分配策略, 当多个进程同时申请某类资源时, 由分配策略确定资源分配给进程的次序。有时资源分配策略可能是不公平的, 即不能保证等待时间上界的存在。在这种情况下, 即使系统没有发生死锁, 某些进程也可能会长时间等待。当等待时间给进程推进和响应带来明显影响时, 称发生了进程“饥饿”, 当“饥饿”到一定程度的进程所赋予的任务即使完成也不再具有实际意义时, 称该进程被“饿死”。

例如, 当有多个进程需要打印机时, 若系统分配打印机的策略是最短文件优先, 则长文件的打印任务将由于短文件的源源不断到来而被无限期推迟, 导致最终“饥饿”甚至“饿死”。

“饥饿”并不表示系统一定会死锁, 但至少有一个进程的执行被无限期推迟。“饥饿”与死锁的主要差别如下:

- 1) 进入“饥饿”状态的进程可以只有一个, 而因循环等待条件而进入死锁状态的进程却必须大于等于两个。
- 2) 处于“饥饿”状态的进程可以是一个就绪进程, 如静态优先权调度算法时的低优先权进程, 而处于死锁状态的进程则必定是阻塞进程。

3. 银行家算法的工作原理

银行家算法的主要思想是避免系统进入不安全状态。在每次进行资源分配时, 它首先检查系统是否有足够的资源满足要求, 若有则先进行分配, 并对分配后的新状态进行安全性检查。若新状态安全, 则正式分配上述资源, 否则拒绝分配上述资源。这样, 它保证系统始终处于安全状态, 从而避免了死锁现象的发生。

4. 进程同步、互斥的区别和联系

并发进程的执行会产生相互制约的关系: 一种是进程之间竞争使用临界资源, 只能让它们逐个使用, 这种现象称为互斥, 是一种竞争关系; 另一种是进程之间协同完成任务, 在关键点上等待另一个进程发来的消息, 以便协同一致, 是一种协作关系。

5. 作业和进程的关系

进程是系统资源的使用者, 系统的资源大部分都是以进程为单位分配的。而用户使用计算机

是为了实现一串相关的任务，通常把用户要求计算机完成的这一串任务称为作业。

(1) 批处理系统中作业与进程的关系（进程组织）

批处理系统可以通过磁记录设备或卡片机向系统提交批作业，由系统的 SPOOLing 输入进程将作业放入磁盘的输入井，作为后备作业。作业调度程序（一般也作为独立的进程运行）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）。该进程将执行作业控制语言解释程序，解释该作业的作业说明书。父进程在运行过程中可以动态地创建一个或多个子进程，执行说明书中的语句。例如，对一条编译的语句，该进程可以创建一个子进程执行编译程序对用户源程序进行编译。类似地，子进程也可继续创建子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—进程族。当父进程遇到作业说明书中的“撤出作业”语句时，将该作业从运行态改变为完成态，将作业及相关结果送入磁盘上的输出井。作业终止进程负责将输出井中的作业利用打印机输出，回收作业所占用的资源，删除作业有关的数据结构，删除作业在磁盘输出井中的信息等。作业终止进程撤除一道作业后，可向作业调度进程请求进行新的作业调度。至此，一道进入系统运行的作业全部结束。

(2) 分时系统中作业与进程的关系

在分时系统中，作业的提交方法、组织形式均与批处理作业有很大差异。分时系统的用户通过命令语言逐条与系统应答式地输入命令，提交作业。每输入一条（或一组）命令，便直接在系统内部对应一个（或若干）进程。在系统启动时，系统为每个终端设备建立一个进程（称为终端进程），该进程执行命令解释程序，命令解释程序从终端设备读入命令，解释执行用户输入的每条命令。对于每条终端命令，可以创建一个子进程去具体执行。若当前的终端命令是一条后台命令，则可以和下一条终端命令并行处理。各子进程在运行过程中完全可以根据需要创建子孙进程。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。用户本次上机完毕，用户通过一条登出命令即结束上机过程。

分时系统的作业就是用户的一次上机交互过程，可以认为终端进程的创建是一个交互作业的开始，登出命令运行结束代表用户交互作业的终止。

命令解释程序流程扮演着批处理系统中作业控制语言解释程序的角色，只不过命令解释程序是从用户终端接收命令。

(3) 交互地提交批作业

在同时支持交互和批处理的操作系统中，人们可以用交互的方式准备好批作业的有关程序、数据及作业控制说明书。比如，可用交互式系统提供的全屏幕编辑命令编辑好自编的一个天气预报程序，用编译及装配命令将程序变成可执行文件，用调试命令进行程序调试。调试成功后，用户每天都要做如下工作：准备原始天气数据，运行天气预报执行文件处理原始数据，把结果打印出来等。这时，用交互系统提供的全屏幕编辑命令编辑好将要提交的作业控制说明书文件，如 Windows 系统的 bat 文件和 Linux 系统的 sh 文件。然后用一条作业提交命令将作业提交到系统作业队列中。系统有专门的作业调度进程负责从作业队列中选择作业，为被选取的作业创建一个父进程运行命令解释程序，解释执行作业控制说明书文件中的命令。

第3章

内存管理

【考纲内容】

(一) 内存管理基础

内存管理概念；程序装入与链接；逻辑地址与物理地址空间；内存保护
连续分配管理方式

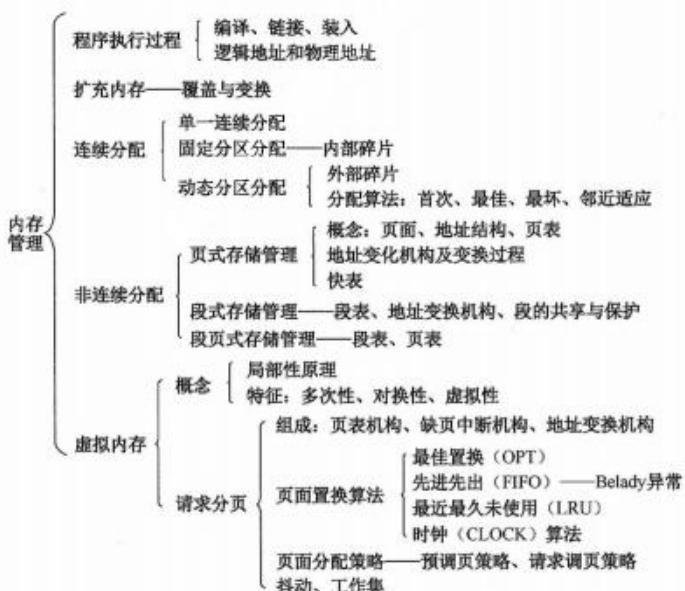
非连续分配管理方式：分页管理方式；分段管理方式；段页式管理方式

(二) 虚拟内存管理

虚拟内存的基本概念；请求分页管理方式；页面置换算法

页面分配策略；工作集；抖动

【知识框架】



【复习提示】

内存管理和进程管理是操作系统的核心内容，需要重点复习。本章围绕分页机制展开：通过分页管理方式在物理内存大小的基础上提高内存的利用率，再进一步引入请求分页管理方式，实现虚拟内存，使内存脱离物理大小的限制，从而提高处理器的利用率。

3.1 内存管理概念

在学习本节时，请读者思考以下问题：

- 1) 为什么要进行内存管理?
- 2) 页式管理中每个页表项大小的下限如何决定?
- 3) 多级页表解决了什么问题? 又会带来什么问题?

在学习经典的管理方法前, 同样希望读者先思考, 自己给出一些内存管理的想法, 并在学习过程中和经典方案进行比较。注意本节给出的内存管理是循序渐进的, 后一种方法通常会解决前一种方法的不足。希望读者多多思考, 比较每种方法的异同, 着重掌握页式管理。

3.1.1 内存管理的基本原理和要求

内存管理 (Memory Management) 是操作系统设计中最重要和最复杂的内容之一。虽然计算机硬件技术一直在飞速发展, 内存容量也在不断增大, 但仍然不可能将所有用户进程和系统所需要的全部程序与数据放入主存, 因此操作系统必须对内存空间进行合理的划分和有效的动态分配。操作系统对内存的划分和动态分配, 就是内存管理的概念。

有效的内存管理在多道程序设计中非常重要, 它不仅可以方便用户使用存储器、提高内存利用率, 还可以通过虚拟技术从逻辑上扩充存储器。

内存管理的功能有:

- 内存空间的分配与回收。由操作系统完成主存储器空间的分配和管理, 使程序员摆脱存储分配的麻烦, 提高编程效率。
- 地址转换。在多道程序环境下, 程序中的逻辑地址与内存中的物理地址不可能一致, 因此存储管理必须提供地址变换功能, 把逻辑地址转换成相应的物理地址。
- 内存空间的扩充。利用虚拟存储技术或自动覆盖技术, 从逻辑上扩充内存。
- 存储保护。保证各道作业在各自的存储空间内运行, 互不干扰。

在进行具体的内存管理之前, 需要了解进程运行的基本原理和要求。

1. 程序装入和链接

创建进程首先要将程序和数据装入内存。将用户源程序变为可在内存中执行的程序, 通常需要以下几个步骤:

- 编译。由编译程序将用户源代码编译成若干目标模块。
- 链接。由链接程序将编译后形成的一组目标模块及所需的库函数链接在一起, 形成一个完整的装入模块。
- 装入。由装入程序将装入模块装入内存运行。

这三步过程如图 3.1 所示。

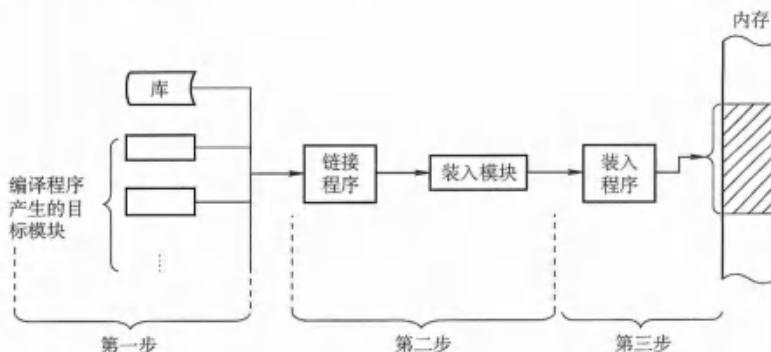


图 3.1 将用户程序变为可在内存中执行的程序的步骤

程序的链接有以下三种方式。

- 静态链接。在程序运行之前，先将各目标模块及它们所需的库函数链接成一个完整的可执行程序，以后不再拆开。
- 装入时动态链接。将用户源程序编译后所得到的一组目标模块，在装入内存时，采用边装入边链接的方式。
- 运行时动态链接。对某些目标模块的链接，是在程序执行中需要该目标模块时才进行的。其优点是便于修改和更新，便于实现对目标模块的共享。

内存的装入模块在装入内存时，同样有以下三种方式：

1) 绝对装入。在编译时，若知道程序将驻留在内存的某个位置，则编译程序将产生绝对地址的目标代码。绝对装入程序按照装入模块中的地址，将程序和数据装入内存。由于程序中的逻辑地址与实际内存地址完全相同，因此不需对程序和数据的地址进行修改。

绝对装入方式只适用于单道程序环境。另外，程序中所用的绝对地址，可在编译或汇编时给出，也可由程序员直接赋予。而通常情况下在程序中采用的是符号地址，编译或汇编时再转换为绝对地址。

2) 可重定位装入。在多道程序环境下，多个目标模块的起始地址（简称始址）通常都从 0 开始，程序中的其他地址都是相对于始址的，此时应采用可重定位装入方式。根据内存的当前情况，将装入模块装入内存的适当位置。装入时对目标程序中指令和数据的修改过程称为重定位，地址变换通常是在装入时一次完成的，所以又称静态重定位，如图 3.2(a) 所示。

静态重定位的特点是，一个作业装入内存时，必须给它分配要求的全部内存空间，若没有足够的内存，则不能装入该作业。此外，作业一旦进入内存，整个运行期间就不能在内存中移动，也不能再申请内存空间。

3) 动态运行时装入，也称动态重定位。程序在内存中若发生移动，则需要采用动态的装入方式。装入程序把装入模块装入内存后，并不立即把装入模块中的相对地址转换为绝对地址，而是把这种地址转换推迟到程序真正要执行时才进行。因此，装入内存后的所有地址均为相对地址。这种方式需要一个重定位寄存器的支持，如图 3.2(b) 所示。

动态重定位的特点如下：可以将程序分配到不连续的存储区中；在程序运行之前可以只装入它的部分代码即可投入运行，然后在程序运行期间，根据需要动态申请分配内存；便于程序段的共享，可以向用户提供一个比存储空间大得多的地址空间。

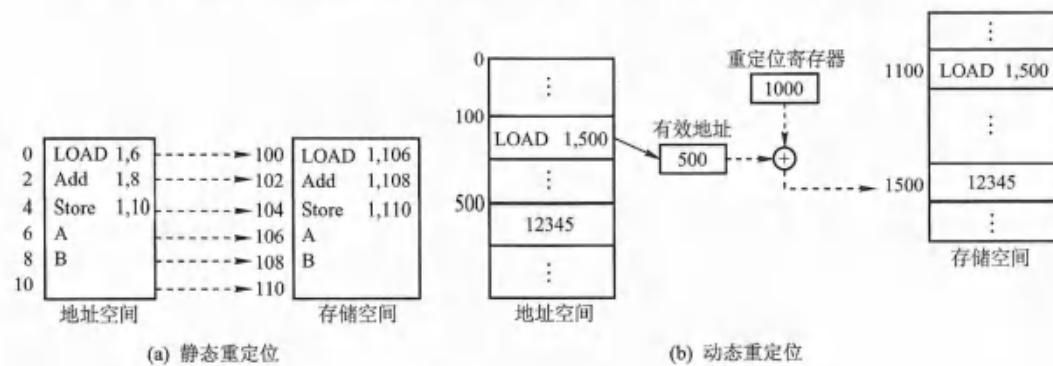


图 3.2 重定位类型

2. 逻辑地址空间与物理地址空间

编译后，每个目标模块都从0号单元开始编址，这称为该目标模块的相对地址（或逻辑地址）。当链接程序将各个模块链接成一个完整的可执行目标程序时，链接程序顺序依次按各个模块的相对地址构成统一的从0号单元开始编址的逻辑地址空间。用户程序和程序员只需知道逻辑地址，而内存管理的具体机制则是完全透明的，只有系统编程人员才会涉及内存管理的具体机制。不同进程可以有相同的逻辑地址，因为这些相同的逻辑地址可以映射到主存的不同位置。

物理地址空间是指内存中物理单元的集合，它是地址转换的最终地址，进程在运行时执行指令和访问数据，最后都要通过物理地址从主存中存取。当装入程序将可执行代码装入内存时，必须通过地址转换将逻辑地址转换成物理地址，这个过程称为地址重定位。

3. 内存保护

内存分配前，需要保护操作系统不受用户进程的影响，同时保护用户进程不受其他用户进程的影响。内存保护可采取两种方法：

- 1) 在CPU中设置一对上、下限寄存器，存放用户作业在主存中的下限和上限地址，每当CPU要访问一个地址时，分别和两个寄存器的值相比，判断有无越界。
- 2) 采用重定位寄存器（或基址寄存器）和界地址寄存器（又称限长寄存器）来实现这种保护。重定位寄存器含最小的物理地址值，界地址寄存器含逻辑地址的最大值。每个逻辑地址值必须小于界地址寄存器；内存管理机构动态地将逻辑地址与界地址寄存器进行比较，若未发生地址越界，则加上重定位寄存器的值后映射成物理地址，再送交内存单元，如图3.3所示。

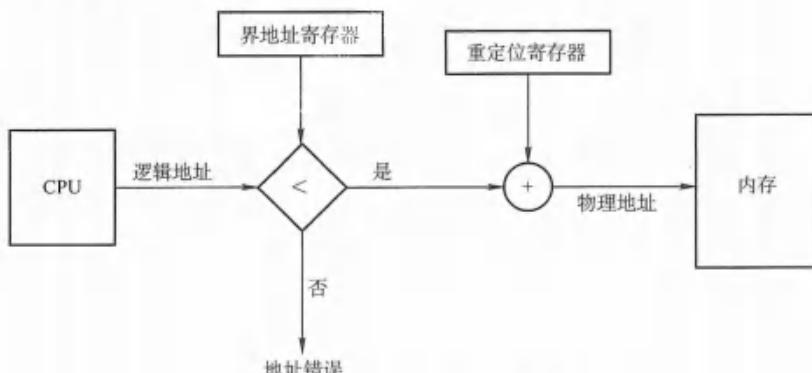


图3.3 重定位和界地址寄存器的硬件支持

实现内存保护需要重定位寄存器和界地址寄存器，因此要注意两者的区别。重定位寄存器是用来“加”的，逻辑地址加上重定位寄存器中的值就能得到物理地址；界地址寄存器是用来“比”的，通过比较界地址寄存器中的值与逻辑地址的值来判断是否越界。

*3.1.2 覆盖与交换^①

覆盖与交换技术是在多道程序环境下用来扩充内存的两种方法。

1. 覆盖

早期的计算机系统中，主存容量很小，虽然主存中仅存放一道用户程序，但存储空间放不下

^① 加“*”号表示新大纲中已删除，仅供学习参考。

用户进程的现象也经常发生，这一矛盾可以用覆盖技术来解决。

覆盖的基本思想如下：由于程序运行时并非任何时候都要访问程序及数据的各个部分（尤其是大程序），因此可把用户空间分成一个固定区和若干覆盖区。将经常活跃的部分放在固定区，其余部分按调用关系分段。首先将那些即将要访问的段放入覆盖区，其他段放在外存中，在需要调用前，系统再将其调入覆盖区，替换覆盖区中原有的段。

覆盖技术的特点是，打破了必须将一个进程的全部信息装入主存后才能运行的限制，但当同时运行程序的代码量大于主存时仍不能运行，此外，内存中能够更新的地方只有覆盖区的段，不在覆盖区中的段会常驻内存。

2. 交换

交换（对换）的基本思想是，把处于等待状态（或在 CPU 调度原则下被剥夺运行权利）的程序从内存移到辅存，把内存空间腾出来，这一过程又称换出；把准备好竞争 CPU 运行的程序从辅存移到内存，这一过程又称换入。第 2 章介绍的中级调度采用的就是交换技术。

例如，有一个 CPU 采用时间片轮转调度算法的多道程序环境。时间片到，内存管理器将刚刚执行过的进程换出，将另一进程换入刚刚释放的内存空间。同时，CPU 调度器可以将时间片分配给其他已在内存中的进程。每个进程用完时间片都与另一进程交换。在理想情况下，内存管理器的交换过程速度足够快，总有进程在内存中可以执行。

有关交换，需要注意以下几个问题：

- 交换需要备份存储，通常是快速磁盘。它必须足够大，并提供对这些内存映像的直接访问。
- 为了有效使用 CPU，需要使每个进程的执行时间比交换时间长，而影响交换时间的主要是转移时间。转移时间与所交换的内存空间成正比。
- 若换出进程，则必须确保该进程完全处于空闲状态。
- 交换空间通常作为磁盘的一整块，且独立于文件系统，因此使用起来可能很快。
- 交换通常在有许多进程运行且内存空间吃紧时开始启动，而在系统负荷降低时就暂停。
- 普通的交换使用不多，但交换策略的某些变体在许多系统（如 UNIX 系统）中仍发挥作用。

交换技术主要在不同进程（或作业）之间进行，而覆盖则用于同一个程序或进程中。由于覆盖技术要求给出程序段之间的覆盖结构，使得其对用户和程序员不透明，所以对于主存无法存放用户程序的矛盾，现代操作系统是通过虚拟内存技术来解决的，覆盖技术则已成为历史；而交换技术在现代操作系统中仍具有较强的生命力。

3.1.3 连续分配管理方式

连续分配方式是指为一个用户程序分配一个连续的内存空间，譬如某用户需要 1GB 的内存空间，连续分配方式就在内存空间中为用户分配一块连续的 1GB 空间。连续分配方式主要包括单一连续分配、固定分区分配和动态分区分配。

1. 单一连续分配

内存在此方式下分为系统区和用户区，系统区仅供操作系统使用，通常在低地址部分；用户区是为用户提供的、除系统区之外的内存空间。这种方式无须进行内存保护。因为内存中永远只有一道程序，因此肯定不会因为访问越界而干扰其他程序。

这种方式的优点是简单、无外部碎片，可以采用覆盖技术，不需要额外的技术支持。缺点是只能用于单用户、单任务的操作系统中，有内部碎片，存储器的利用率极低。

2. 固定分区分配

固定分区分配是最简单的一种多道程序存储管理方式，它将用户内存空间划分为若干固定大小的区域，每个分区只装入一道作业。当有空闲分区时，便可再从外存的后备作业队列中选择适当大小的作业装入该分区，如此循环。

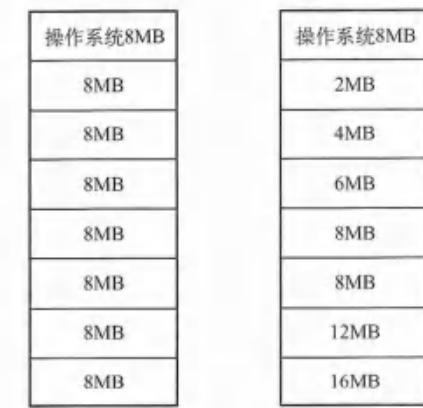
固定分区分配在划分分区时有两种不同的方法，如图 3.4 所示。

- 分区大小相等。用于利用一台计算机去控制多个相同对象的场合，缺乏灵活性。
- 分区大小不等。划分为多个较小的分区、适量的中等分区和少量大分区。

为便于内存分配，通常将分区按大小排队，并为之建立一张分区说明表，其中各表项包括每个分区的始址、大小及状态（是否已分配），如图 3.5(a)所示。当有用户程序要装入时，便检索该表，以找到合适的分区给予分配并将其状态置为“已分配”；未找到合适分区时，则拒绝为该用户程序分配内存。存储空间的分配情况如图 3.5(b)所示。

分区号	大小/KB	起址/KB	状态
1	12	20	已分配
2	32	32	已分配
3	64	64	已分配
4	128	128	已分配

(a) 分区说明表



(a) 大小相等的分区

(b) 大小不等的分区

图 3.4 固定分区分配的两种方法

地址	操作
20KB	操作系统
32KB	作业A
64KB	作业B
128KB	作业C
256KB	

(b) 存储空间分配情况

图 3.5 固定分区说明表和内存分配情况

这种分区方式存在两个问题：一是程序可能太大而放不进任何一个分区中，这时用户不得不使用覆盖技术来使用内存空间；二是主存利用率低，当程序小于固定分区大小时，也占用一个完整的内存分区空间，这样分区内部就存在空间浪费，这种现象称为内部碎片。

固定分区是可用于多道程序设计的最简单的存储分配，无外部碎片，但不能实现多进程共享一个主存区，所以存储空间利用率低。固定分区分配很少用于现在通用的操作系统中，但在某些用于控制多个相同对象的控制系统中仍发挥着一定的作用。

3. 动态分区分配

动态分区分配又称可变分区分配，是一种动态划分内存的分区方法。这种分区方法不预先划分内存，而是在进程装入内存时，根据进程的大小动态地建立分区，并使分区的大小正好适合进程的需要。因此，系统中分区的大小和数目是可变的。

如图 3.6 所示，系统有 64MB 内存空间，其中低 8MB 固定分配给操作系统，其余为用户可用内存。开始时装入前三个进程，它们分别分配到所需的空间后，内存只剩下 4MB，进程 4 无法装入。在某个时刻，内存中没有一个就绪进程，CPU 出现空闲，操作系统就换出进程 2，换入进程 4。由于进程 4 比进程 2 小，这样在主存中就产生了一个 6MB 的内存块。之后 CPU 又出现空闲，而主存无法容纳进程 2，操作系统就换出进程 1，换入进程 2。

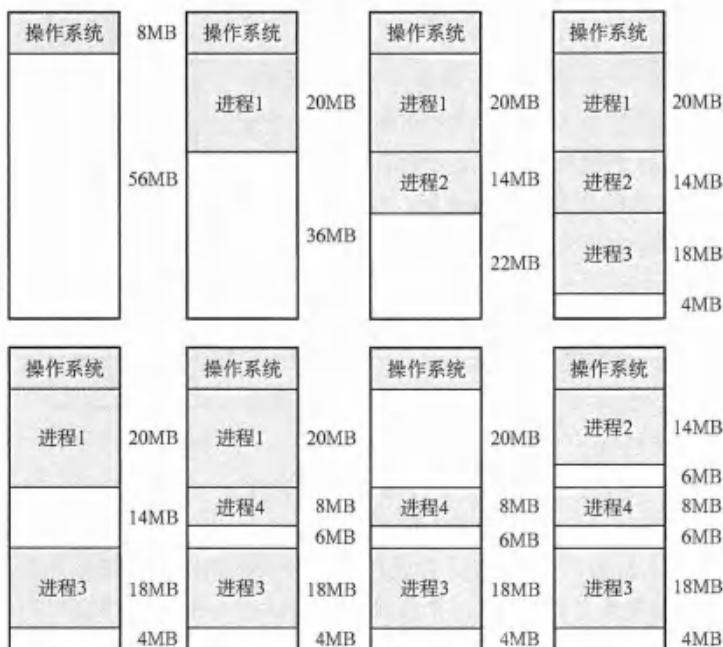


图 3.6 动态分区

动态分区在开始分配时是很好的，但之后会导致内存中出现许多小的内存块。随着时间的推移，内存中会产生越来越多的碎片（图 3.6 中最后的 4MB 和中间的 6MB，且随着进程的换入/换出，很可能会出现更多、更小的内存块），内存的利用率随之下降。这些小的内存块称为外部碎片，指在所有分区外的存储空间会变成越来越多的碎片，这与固定分区中的内部碎片正好相对。克服外部碎片可以通过紧凑（Compaction）技术来解决，即操作系统不时地对进程进行移动和整理。但这需要动态重定位寄存器的支持，且相对费时。紧凑的过程实际上类似于 Windows 系统中的磁盘整理程序，只不过后者是对外存空间的紧凑。

在进程装入或换入主存时，若内存中有多个足够大的空闲块，则操作系统必须确定分配哪个内存块给进程使用，这就是动态分区的分配策略。考虑以下几种算法：

- 1) 首次适应（First Fit）算法。空闲分区以地址递增的次序链接。分配内存时顺序查找，找到大小能满足要求的第一个空闲分区。
- 2) 最佳适应（Best Fit）算法。空闲分区按容量递增的方式形成分区链，找到第一个能满足要求的空闲分区。
- 3) 最坏适应（Worst Fit）算法。又称最大适应（Largest Fit）算法，空闲分区以容量递减的次序链接，找到第一个能满足要求的空闲分区，即挑选出最大的分区。
- 4) 邻近适应（Next Fit）算法。又称循环首次适应算法，由首次适应算法演变而成。不同之处是，分配内存时从上次查找结束的位置开始继续查找。

在这几种方法中，首次适应算法不仅是最简单的，而且通常也是最好和最快的。在 UNIX 系统的最初版本中，就是使用首次适应算法为进程分配内存空间的，它使用数组的数据结构（而非链表）来实现。不过，首次适应算法会使得内存的低地址部分出现很多小的空闲分区，而每次分查找时，都要经过这些分区，因此增加了查找的开销。

邻近适应算法试图解决这个问题。但实际上，它常常导致在内存的末尾分配空间（因为在一遍扫描中，内存前面部分使用后再释放时，不会参与分配）分裂成小碎片。它通常比首次适应算

法的结果要差。

最佳适应算法虽然称为“最佳”，但是性能通常很差，因为每次最佳的分配会留下很小的难以利用的内存块，会产生最多的外部碎片。

最坏适应算法与最佳适应算法相反，它选择最大的可用块，这看起来最不容易产生碎片，但是却把最大的连续内存划分开，会很快导致没有可用的大内存块，因此性能也非常差。

Knuth 和 Shore 分别就前三种方法对内存空间的利用情况做了模拟实验，结果表明：首次适应算法可能比最佳适应法效果好，而它们两者一定比最大适应法效果好。另外要注意，在算法实现时，分配操作中最佳适应法和最大适应法需要对可用块进行排序或遍历查找，而首次适应法和邻近适应法只需要简单查找；在回收操作中，当回收的块与原来的空闲块相邻时（有三种相邻的情况，比较复杂），需要将这些块合并。在算法实现时，使用数组或链表进行管理。除了内存的利用率，这里的算法开销也是操作系统设计需要考虑的一个因素。

三种内存分区管理方式的比较见表 3.1。

表 3.1 三种内存分区管理方式的比较

	作业道数	内部 碎片	外部 碎片	硬件支持	可用空 间管理	解决碎 片方法	解决空 间不足	提高作 业道数
单道连续 分配	1	有	无	界地址寄存器、越界检 查机构	—	—	覆盖	交换
多道固定 连续分配	$\leq N$ (用户空间 划为 N 块)	有	无	① 上下界寄存器、越界 检查机构 ② 地址寄存器、长度 寄存器、动态地址转 换机构	—	—		
多道可变 连续分配	—	无	有		① 数组 ② 链表	紧凑		

以上三种内存分区管理方法有一个共同特点，即用户进程（或作业）在主存中都是连续存放的。这里对它们进行比较和总结。

3.1.4 非连续分配管理方式

非连续分配允许一个程序分散地装入不相邻的内存分区。在连续分配管理方式中，我们发现，即使内存有超过 1GB 的空闲空间，但若没有连续的 1GB 空间，则需要 1GB 空间的作业仍然是无法运行的；但若采用非连续分配管理方式，则作业所要求的 1GB 内存空间可以分散地分配在内存的各个区域，当然，这也需要额外的空间去存储它们（分散区域）的索引，使得非连续分配方式的存储密度低于连续存储方式的。

非连续分配管理方式根据分区的大小是否固定，分为分页存储管理方式和分段存储管理方式。

在分页存储管理方式中，又根据运行作业时是否要把作业的所有页面都装入内存才能运行，分为基本分页存储管理方式和请求分页存储管理方式。下面介绍基本分页存储管理方式。

1. 基本分页存储管理方式

固定分区会产生内部碎片，动态分区会产生外部碎片，这两种技术对内存的利用率都比较低。我们希望内存的使用能尽量避免碎片的产生，这就引入了分页的思想：把主存空间划分为大小相等且固定的块，块相对较小，作为主存的基本单位。每个进程也以块为单位进行划分，进程在执行时，以块为单位逐个申请主存中的块空间。

分页的方法从形式上看，像分区相等的固定分区技术，分页管理不会产生外部碎片。但它又

有本质的不同点：块的大小相对分区要小很多，而且进程也按照块进行划分，进程运行时按块申请主存可用空间并执行。这样，进程只会在为最后一个不完整的块申请一个主存块空间时，才产生主存碎片，所以尽管会产生内部碎片，但这种碎片相对于进程来说也是很小的，每个进程平均只产生半个块大小的内部碎片（也称页内碎片）。

(1) 分页存储的几个基本概念

① 页面和页面大小。进程中的块称为页（Page），内存中的块称为页框（Page Frame，或页帧）。外存也以同样的单位进行划分，直接称为块（Block）。进程在执行时需要申请主存空间，即要为每个页面分配主存中的可用页框，这就产生了页和页框的一一对应。

为方便地址转换，页面大小应是 2 的整数幂。同时页面大小应该适中，页面太小会使进程的页面数过多，这样页表就会过长，占用大量内存，而且也会增加硬件地址转换的开销，降低页面换入/换出的效率；页面过大又会使页内碎片增多，降低内存的利用率。所以页面的大小应该适中，要在空间效率和时间效率之间权衡。

② 地址结构。分页存储管理的逻辑地址结构如图 3.7 所示。

31	...	12	11	...	0
页号 P			页内偏移量 W		

图 3.7 分页存储管理的逻辑地址结构

地址结构包含两部分：前一部分为页号 P ，后一部分为页内偏移量 W 。地址长度为 32 位，其中 0~11 位为页内地址，即每页大小为 4KB；12~31 位为页号，地址空间最多允许 2^{20} 页。

注意，地址结构决定了虚拟内存的寻址空间有多大。在实际问题中，页号、页内偏移、逻辑地址大多都是用十进制数给出的。题目用二进制地址的形式给出时，读者要学会转换。

③ 页表。为了便于在内存中找到进程的每个页面所对应的物理块，系统为每个进程建立一张页表，它记录页面在内存中对应的物理块号，页表一般存放在内存中。

页表是由页表项组成的，初学者容易混淆页表项与地址结构，页表项与地址都由两部分构成，而且第一部分都是页号，但页表项的第二部分是物理内存中的块号，而地址的第二部分是页内偏移；页表项的第二部分与地址的第二部分共同组成物理地址。

在配置页表后，进程执行时，通过查找该表，即可找到每页在内存中的物理块号。可见，页表的作用是实现从页号到物理块号的地址映射，如图 3.8 所示。

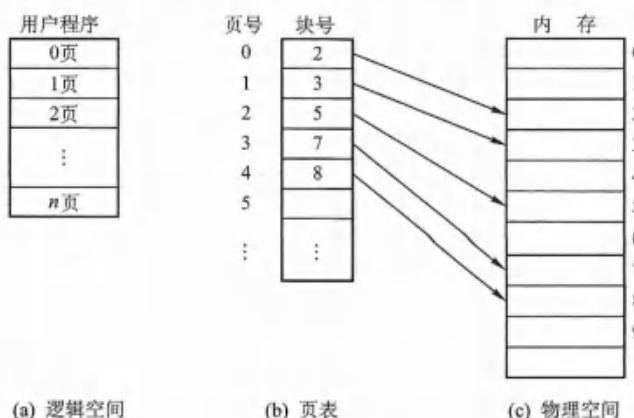


图 3.8 页表的作用

(2) 基本地址变换机构

地址变换机构的任务是将逻辑地址转换为内存中的物理地址。地址变换是借助于页表实现的。图 3.9 给出了分页存储管理系统中的地址变换机构。

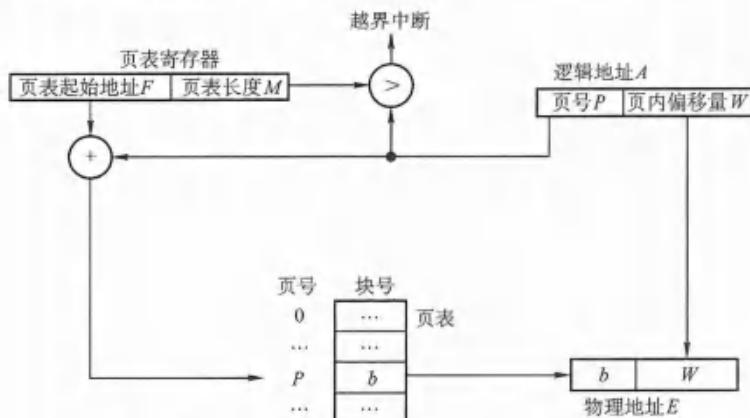


图 3.9 分页存储管理系统中的地址变换机构

在系统中通常设置一个页表寄存器 (PTR)，存放页表在内存的起始地址 F 和页表长度 M 。进程未执行时，页表的始址和长度存放在进程控制块中，当进程执行时，才将页表始址和长度存入页表寄存器。设页面大小为 L ，逻辑地址 A 到物理地址 E 的变换过程如下 (逻辑地址、页号、每页的长度都是十进制数)：

- ① 计算页号 P ($P = A/L$) 和页内偏移量 W ($W = A \% L$)。
- ② 比较页号 P 和页表长度 M ，若 $P \geq M$ ，则产生越界中断，否则继续执行。
- ③ 页表中页号 P 对应的页表项地址 = 页表始址 $F +$ 页号 $P \times$ 页表项长度，取出该页表项内容 b ，即为物理块号。要注意区分页表长度和页表项长度。页表长度的值是指一共有多少页，页表项长度是指页地址占多大的存储空间。
- ④ 计算 $E = b \times L + W$ ，用得到的物理地址 E 去访问内存。

以上整个地址变换过程均是由硬件自动完成的。例如，若页面大小 L 为 1KB，页号 2 对应的物理块为 $b = 8$ ，计算逻辑地址 $A = 2500$ 的物理地址 E 的过程如下： $P = 2500/1K = 2$ ， $W = 2500 \% 1K = 452$ ，查找得到页号 2 对应的物理块的块号为 8， $E = 8 \times 1024 + 452 = 8644$ 。

要再次提醒读者的是，题目中条件用十进制数给出和用二进制数给出的处理过程会稍有不同。同时读者会发现，页式管理只需给出一个整数就能确定对应的物理地址，因为页面大小 L 是固定的。因此，页式管理中地址空间是一维的。

页表项的大小不是随意规定的，而是有所约束的。如何确定页表项的大小？

页表项的作用是找到该页在内存中的位置。以 32 位逻辑地址空间、字节编址单位、一页 4KB 为例，地址空间内一共有 $2^{32}B / 4KB = 1M$ 页，因此需要 $\log_2 1M = 20$ 位才能保证表示范围能容纳所有页面，又因为以字节作为编址单位，即页表项的大小 $\geq \lceil 20/8 \rceil = 3B$ 。所以在这个条件下，为了保证页表项能够指向所有页面，页表项的大小应该大于 3B，当然，也可选择更大的页表项让一个页面能够正好容下整数个页表项，进而方便存储(如取成 4B，一页正好可以装下 1K 个页表项)，或增加一些其他信息。

下面讨论分页管理方式存在的两个主要问题：① 每次访存操作都需要进行逻辑地址到物理地址的转换，地址转换过程必须足够快，否则访存速度会降低；② 每个进程引入页表，用于存储映射机制，页表不能太大，否则内存利用率会降低。

(3) 具有快表的地址变换机构

由上面介绍的地址变换过程可知，若页表全部放在内存中，则存取一个数据或一条指令至少要访问两次内存：第一次是访问页表，确定所存取的数据或指令的物理地址；第二次是根据该地址存取数据或指令。显然，这种方法比通常执行指令的速度慢了一半。

为此，在地址变换机构中增设一个具有并行查找能力的高速缓冲存储器——快表，又称相联存储器（TLB），用来存放当前访问的若干页表项，以加速地址变换的过程。与此对应，主存中的页表常称为慢表。具有快表的地址变换机构如图 3.10 所示。

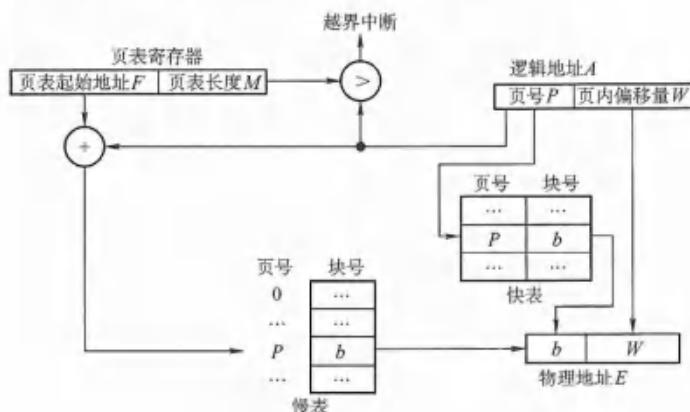


图 3.10 具有快表的地址变换机构

在具有快表的分页机制中，地址的变换过程如下：

- ① CPU 给出逻辑地址后，由硬件进行地址转换，将页号送入高速缓存寄存器，并将此页号与快表中的所有页号进行比较。
- ② 若找到匹配的页号，说明所要访问的页表项在快表中，则直接从中取出该页对应的页框号，与页内偏移量拼接形成物理地址。这样，存取数据仅一次访存便可实现。
- ③ 若未找到匹配的页号，则需要访问主存中的页表，在读出页表项后，应同时将其存入快表，以便后面可能的再次访问。但若快表已满，则必须按照一定的算法对旧的页表项进行替换。

注意：有些处理机设计为快表和慢表同时查找，若在快表中查找成功则终止慢表的查找。

一般快表的命中率可达 90% 以上，这样分页带来的速度损失就可降低至 10% 以下。快表的有效性基于著名的局部性原理，后面讲解虚拟内存时将会具体讨论它。

(4) 两级页表

由于引入了分页管理，进程在执行时不需要将所有页调入内存页框，而只需将保存有映射关系的页表调入内存。但是，我们仍然需要考虑页表的大小。以 32 位逻辑地址空间、页面大小 4KB、页表项大小 4B 为例，若要实现进程对全部逻辑地址空间的映射，则每个进程需要 2^{20} 即约 100 万个页表项。也就是说，每个进程仅页表这一项就需要 4MB 主存空间，这显然是不切实际的。即便不考虑对全部逻辑地址空间进行映射的情况，一个逻辑地址空间稍大的进程，其页表大小也可能是过大的。以一个 40MB 的进程为例，页表项共 40KB ($40\text{MB}/4\text{KB} \times 4\text{B}$)，若将所有页表项内容保存在内存中，则需要 10 个内存页框来保存整个页表。整个进程大小约为 1 万个页面，而实际执行时只需要几十个页面进入内存页框就可运行，但若要求 10 个页面大小的页表必须全部进入内存，则相对实际执行时的几十个进程页面的大小来说，肯定降低了内存利用率；从另一方面来说，这 10 页的页表项也并不需要同时保存在内存中，因为在大多数情况下，映射所需要的页

表项都在页表的同一个页面中。

为了压缩页表，我们进一步延伸页表映射的思想，就可得到二级分页，即使用层次结构的页表：将页表的 10 页空间也进行地址映射，建立上一级页表，用于存储页表的映射关系。这里对页表的 10 个页面进行映射只需要 10 个页表项，所以上一级页表只需要 1 页就已足够（可以存储 $2^{10} = 1024$ 个页表项）。在进程执行时，只需要将这一页的上一级页表调入内存即可，进程的页表和进程本身的页面可在后面的执行中再调入内存。根据上面提到的条件（32 位逻辑地址空间、页面大小 4KB、页表项大小 4B，以字节为编址单位），我们来构造一个适合的页表结构。页面大小为 4KB，页内偏移地址为 $\log_2 4K = 12$ 位，页号部分为 20 位，若不采用分级页表，则仅页表就要占用 $2^{20} \times 4B / 4KB = 1024$ 页，这大大超过了许多进程自身需要的页面，对于内存来说是非常浪费资源的，而且查询页表工作也会变得十分不便、试想若把这些页表放在连续的空间内，查询对应页的物理页号时可以通过页表首地址 + 页号 $\times 4B$ 的形式得到，而这种方法查询起来虽然相对方便，但连续的 1024 页对于内存的要求实在太高，并且上面也说到了其中大多数页面都是不会用到的，所以这种方法并不具有可行性。若不把这些页表放在连续的空间里，则需要一张索引表来告诉我们第几张页表该上哪里去找，这能解决页表的查询问题，且不用把所有的页表都调入内存，只在需要它时才调入（下节介绍的虚拟存储器思想），因此能解决占用内存空间过大的问题。读者也许发现这个方案就和当初引进页表机制的方式一模一样，实际上就是构造一个页表的页表，也就是二级页表。为查询方便，顶级页表最多只能有 1 个页面（一定要记住这个规定），因此顶级页表总共可以容纳 $4KB / 4B = 1K$ 个页表项，它占用的地址位数为 $\log_2 1K = 10$ 位，而之前已经计算出页内偏移地址占用了 12 位，因此一个 32 位的逻辑地址空间就剩下了 10 位，正好使得二级页表的大小在一页之内，这样就得到了逻辑地址空间的格式，如图 3.11 所示。

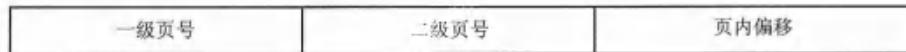


图 3.11 逻辑地址空间的格式

二级页表实际上是在原有页表结构上再加上一层页表，示意结构如图 3.12 所示。

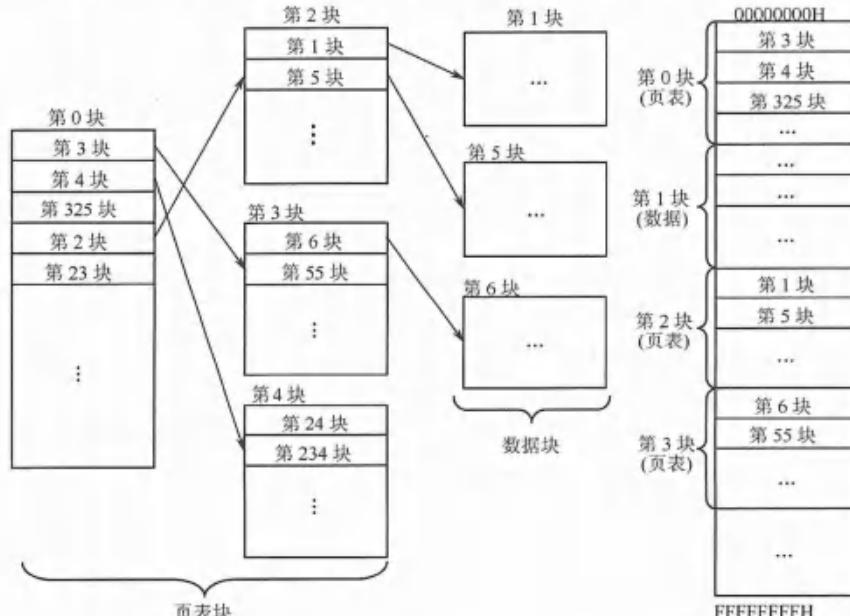


图 3.12 二级页表结构示意图

建立多级页表的目的在于建立索引，以便不用浪费主存空间去存储无用的页表项，也不用盲目地顺序式查找页表项。

2. 基本分段存储管理方式

分页管理方式是从计算机的角度考虑设计的，目的是提高内存的利用率，提升计算机的性能。分页通过硬件机制实现，对用户完全透明。分段管理方式的提出则考虑了用户和程序员，以满足方便编程、信息保护和共享、动态增长及动态链接等多方面的需要。

1) 分段。段式管理方式按照用户进程中的自然段划分逻辑空间。例如，用户进程由主程序、两个子程序、栈和一段数据组成，于是可以把这个用户进程划分为 5 段，每段从 0 开始编址，并分配一段连续的地址空间（段内要求连续，段间不要求连续，因此整个作业的地址空间是二维的），其逻辑地址由段号 S 与段内偏移量 W 两部分组成。

在图 3.13 中，段号为 16 位，段内偏移量为 16 位，因此一个作业最多有 $2^{16} = 65536$ 段，最大段长为 64KB。

31	...	16	15	...	0
段号 S			段内偏移量 W		

图 3.13 分段系统中的逻辑地址结构

在页式系统中，逻辑地址的页号和页内偏移量对用户是透明的，但在段式系统中，段号和段内偏移量必须由用户显式提供，在高级程序设计语言中，这个工作由编译程序完成。

2) 段表。每个进程都有一张逻辑空间与内存空间映射的段表，其中每个段表项对应进程的一段，段表项记录该段在内存中的始址和长度。段表的内容如图 3.14 所示。

段号	段长	本段在主存的始址
----	----	----------

图 3.14 段表的内容

配置段表后，执行中的进程可通过查找段表，找到每段所对应的内存区。可见，段表用于实现从逻辑段到物理内存区的映射，如图 3.15 所示。

3) 地址变换机构。分段系统的地址变换过程如图 3.16 所示。为了实现进程从逻辑地址到物理地址的变换功能，在系统中设置了段表寄存器，用于存放段表始址 F 和段表长度 M 。从逻辑地址 A 到物理地址 E 之间的地址变换过程如下：

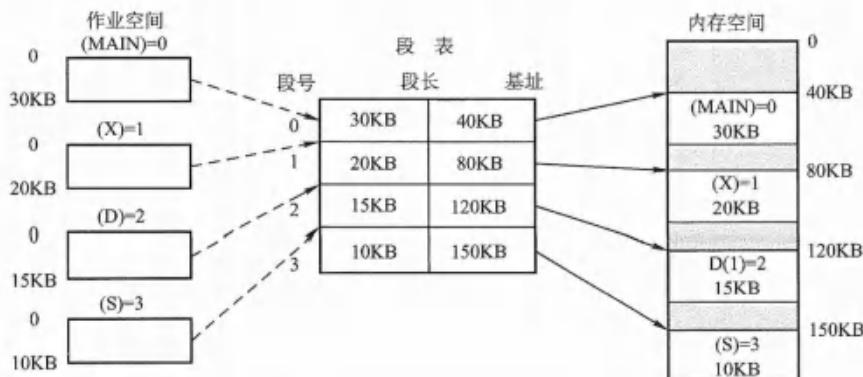


图 3.15 利用段表实现物理内存区映射

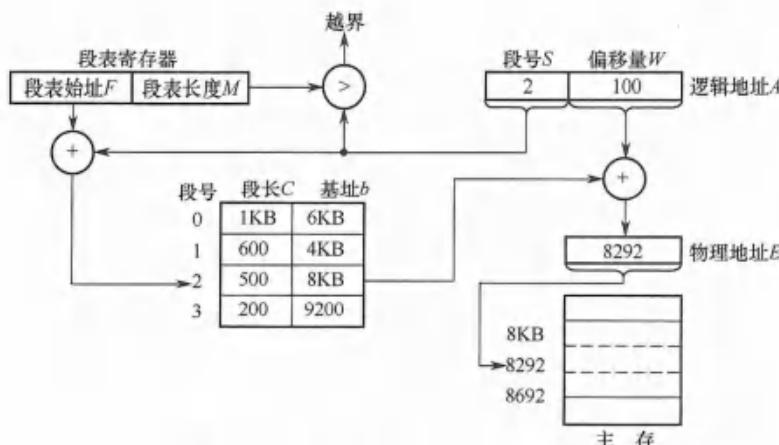


图 3.16 分段系统的地址变换过程

- ① 从逻辑地址 A 中取出前几位为段号 S , 后几位为段内偏移量 W , 注意在段式存储管理的题目中, 逻辑地址一般以二进制数给出, 而在页式存储管理中, 逻辑地址一般以十进制数给出, 读者要具体问题具体分析。
- ② 比较段号 S 和段表长度 M , 若 $S \geq M$, 则产生越界中断, 否则继续执行。
- ③ 段表中段号 S 对应的段表项地址 = 段表始址 $F + S \times$ 段表项长度, 取出该段表项的前几位得到段长 C 。若段内偏移量 $\geq C$, 则产生越界中断, 否则继续执行。从这句话我们可以看出, 段表项实际上只有两部分, 前几位是段长, 后几位是始址。
- ④ 取出段表项中该段的始址 b , 计算 $E = b + W$, 用得到的物理地址 E 去访问内存。
- 4) 段的共享与保护。在分段系统中, 段的共享是通过两个作业的段表中相应表项指向被共享的段的同一个物理副本实现的。当一个作业正从共享段中读取数据时, 必须防止另一个作业修改此共享段中的数据。不能修改的代码称为纯代码或可重入代码(它不属于临界资源), 这样的代码和不能修改的数据可以共享, 而可修改的代码和数据不能共享。

与分页管理类似, 分段管理的保护方法主要有两种: 一种是存取控制保护, 另一种是地址越界保护。地址越界保护将段表寄存器中的段表长度与逻辑地址中的段号比较, 若段号大于段表长度, 则产生越界中断; 再将段表项中的段长和逻辑地址中的段内偏移进行比较, 若段内偏移大于段长, 也会产生越界中断。分页管理中的地址越界保护只需要判断页号是否越界, 页内偏移是不可能越界的。

与页式管理不同, 段式管理不能通过给出一个整数便确定对应的物理地址, 因为每段的长度是不固定的, 无法通过整数除法得出段号, 无法通过求余得出段内偏移, 所以段号和段内偏移一定要显式给出(段号, 段内偏移), 因此分段管理的地址空间是二维的。

3. 段页式管理方式

页式存储管理能有效地提高内存利用率, 而分段存储管理能反映程序的逻辑结构并有利于段的共享。将这两种存储管理方法结合起来, 便形成了段页式存储管理方式。

在段页式系统中, 作业的地址空间首先被分成若干逻辑段, 每段都有自己的段号, 然后将每段分成若干大小固定的页。对内存空间的管理仍然和分页存储管理一样, 将其分成若干和页面大小相同的存储块, 对内存的分配以存储块为单位, 如图 3.17 所示。

在段页式系统中, 作业的逻辑地址分为三部分: 段号、页号和页内偏移量, 如图 3.18 所示。

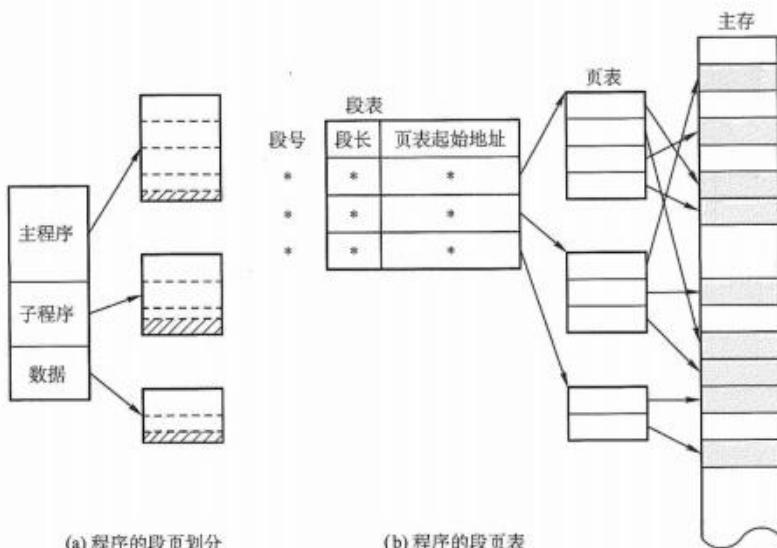


图 3.17 段页式管理方式

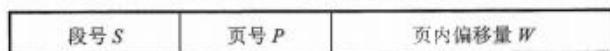


图 3.18 段页式系统的逻辑地址结构

为了实现地址变换，系统为每个进程建立一张段表，每个分段有一张页表。段表表项中至少包括段号、页表长度和页表始址，页表表项中至少包括页号和块号。此外，系统中还应有一个段表寄存器，指出作业的段表始址和段表长度（段表寄存器和页表寄存器的作用都有两个，一是在段表或页表中寻址，二是判断是否越界）。

注意：在一个进程中，段表只有一个，而页表可能有多个。

在进行地址变换时，首先通过段表查到页表始址，然后通过页表找到页帧号，最后形成物理地址。如图 3.19 所示，进行一次访问实际需要三次访问主存，这里同样可以使用快表来加快查找速度，其关键字由段号、页号组成，值是对应的页帧号和保护码。

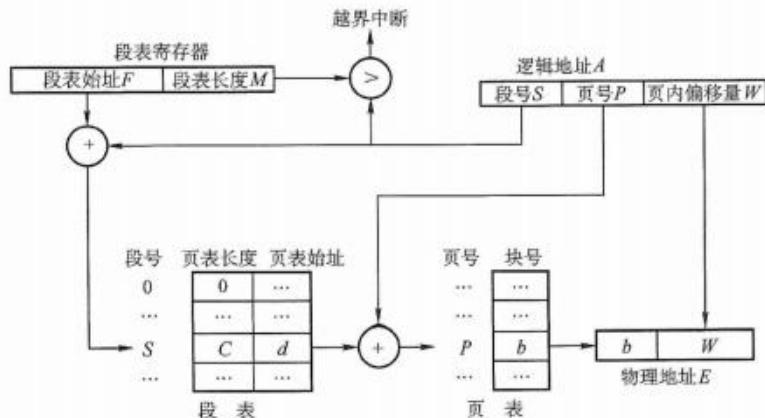


图 3.19 段页式系统的地址变换机构

结合上面对段式管理和页式管理的地址空间的分析，可以得出结论：段页式管理的地址空间是二维的。

3.1.5 本节小结

本节开头提出的问题的参考答案如下。

1) 为什么要进行内存管理?

在单道批处理系统阶段,一个系统在一个时间段内只执行一个程序,内存的分配极其简单,即仅分配给当前运行的进程。引入多道程序的并发执行后,进程之间共享的不仅仅是处理器,还有主存储器。然而,共享主存会形成一些特殊的挑战。若不对内存进行管理,则容易导致内存数据的混乱,以至于限制进程的并发执行。因此,为了更好地支持多道程序并发执行,必须进行内存管理。

2) 页式管理中每个页表项大小的下限如何决定?

页表项的作用是找到该页在内存中的位置。以32位逻辑地址空间、字节编址单位、一页4KB为例,地址空间内共含有 $2^{32}B/4KB = 1M$ 页,需要 $\log_2 1M = 20$ 位才能保证表示范围能容纳所有页面,又因为以字节作为编址单位,即页表项的大小 $\geq \lceil 20/8 \rceil = 3B$ 。所以在这个条件下,为了保证页表项能够指向所有页面,页表项的大小应该大于3B;当然,也可选择更大的页表项大小,让一个页面能够正好容下整数个页表项,以方便存储(例如取成4B,一页正好可以装下1K个页表项),或增加一些其他信息。

3) 多级页表解决了什么问题?又会带来什么问题?

多级页表解决了当逻辑地址空间过大时,页表的长度会大大增加的问题。而采用多级页表时,一次访盘需要多次访问内存甚至磁盘,会大大增加一次访存的时间。

不少读者表示本节的内容难以掌握,实际上本节的内容并不难,只要抓住下列几个关键的线索,本节的所有知识点就能了然于胸。

无论是段式管理、页式管理还是段页式管理,读者都只需要关注三个问题:①逻辑地址结构,②表项结构,③寻址过程。搞清楚这三个问题,就相当于搞清楚了上面几种存储管理方式。再次提醒读者区分逻辑地址结构和表项结构。

3.1.6 本节习题精选

一、单项选择题

- 【2011统考真题】在虚拟内存管理中,地址变换机构将逻辑地址变换为物理地址,形成该逻辑地址的阶段是()。
 - 编辑
 - 编译
 - 链接
 - 装载
- 下面关于存储管理的叙述中,正确的是()。
 - 存储保护的目的是限制内存的分配
 - 在内存为M、有N个用户的分时系统中,每个用户占用M/N的内存空间
 - 在虚拟内存系统中,只要磁盘空间无限大,作业就能拥有任意大的编址空间
 - 实现虚拟内存管理必须有相应硬件的支持
- 在使用交换技术时,若一个进程正在(),则不能交换出主存。
 - 创建
 - I/O操作
 - 处于临界段
 - 死锁
- 在存储管理中,采用覆盖与交换技术的目的是()。
 - 节省主存空间
 - 物理上扩充主存容量
 - 提高CPU效率
 - 实现主存共享
- 【2009统考真题】分区分配内存管理方式的主要保护措施是()。

- A. 界地址保护 B. 程序代码保护 C. 数据保护 D. 栈保护
6. 【2010 统考真题】某基于动态分区存储管理的计算机，其主存容量为 55MB（初始为空），采用最佳适配（Best Fit）算法，分配和释放的顺序为：分配 15MB，分配 30MB，释放 15MB，分配 8MB，分配 6MB，此时主存中最大空闲分区的大小是（ ）。
- A. 7MB B. 9MB C. 10MB D. 15MB
7. 段页式存储管理中，地址映射表是（ ）。
- A. 每个进程一张段表，两张页表
B. 每个进程的每个段一张段表，一张页表
C. 每个进程一张段表，每个段一张页表
D. 每个进程一张页表，每个段一张段表
8. 内存保护需要由（ ）完成，以保证进程空间不被非法访问。
- A. 操作系统 B. 硬件机构
C. 操作系统和硬件机构合作 D. 操作系统或者硬件机构独立完成
9. 存储管理方案中，（ ）可采用覆盖技术。
- A. 单一连续存储管理 B. 可变分区存储管理
C. 段式存储管理 D. 段页式存储管理
10. 在可变分区分配方案中，某一进程完成后，系统回收其主存空间并与相邻空闲区合并，为此需修改空闲区表，造成空闲区数减 1 的情况是（ ）。
- A. 无上邻空闲区也无下邻空闲区
B. 有上邻空闲区但无下邻空闲区
C. 有下邻空闲区但无上邻空闲区
D. 有上邻空闲区也有下邻空闲区
11. 设内存的分配情况如右图所示。若要申请一块 40KB 的内存空间，采用最佳适应算法，则所得到的分区首址为（ ）。
- A. 100K B. 190K
C. 330K D. 410K
12. 某段表的内容见右表，一逻辑地址为(2, 154)，它对应的物理地址为（ ）。
- A. 120K + 2 B. 480K + 154
C. 30K + 154 D. 480K + 2
13. 动态重定位是在作业的（ ）中进行的。
- A. 编译过程 B. 装入过程 C. 链接过程 D. 执行过程
14. 下面的存储管理方案中，（ ）方式可以采用静态重定位。
- A. 固定分区 B. 可变分区 C. 页式 D. 段式
15. 在可变分区管理中，采用拼接技术的目的是（ ）。
- A. 合并空闲区 B. 合并分配区 C. 增加主存容量 D. 便于地址转换
16. 在一页式存储管理系统中，页表内容见右表。若页的大小为 4KB，则地址转换机构将逻辑地址 0 转换成的物理地址为（块号从 0 开始计算）（ ）。
- A. 8192 B. 4096
C. 2048 D. 1024

0	操作系统
100K	占用
180K	占用
190K	占有
280K	占有
330K	占有
390K	占有
410K	占有
512K	

段号	段首址	段长度
0	120K	40K
1	760K	30K
2	480K	20K
3	370K	20K

页号	块号
0	2
1	1
3	3
4	7

17. 不会产生内部碎片的存储管理是()。
 A. 分页式存储管理 B. 分段式存储管理
 C. 固定分区式存储管理 D. 段页式存储管理
18. 多进程在主存中彼此互不干扰的环境下运行，操作系统是通过()来实现的。
 A. 内存分配 B. 内存保护 C. 内存扩充 D. 地址映射
19. 分区管理中采用最佳适应分配算法时，把空闲区按()次序登记在空闲区表中。
 A. 长度递增 B. 长度递减 C. 地址递增 D. 地址递减
20. 首次适应算法的空间分区()。
 A. 按大小递减顺序连在一起 B. 按大小递增顺序连在一起
 C. 按地址由小到大排列 D. 按地址由大到小排列
21. 采用分页或分段管理后，提供给用户的物理地址空间()。
 A. 分页支持更大的物理地址空间 B. 分段支持更大的物理地址空间
 C. 不能确定 D. 一样大
22. 分页系统中的页面是为()。
 A. 用户所感知的 B. 操作系统所感知的
 C. 编译系统所感知的 D. 连接装配程序所感知的
23. 页式存储管理中，页表的始地址存放在()中。
 A. 内存 B. 存储页表 C. 快表 D. 寄存器
24. 对重定位存储管理方式，应()。
 A. 在整个系统中设置一个重定位寄存器 B. 为每道程序设置一个重定位寄存器
 C. 为每道程序设置两个重定位寄存器 D. 为每道程序和数据都设置一个重定位寄存器
25. 采用段式存储管理时，一个程序如何分段是在()时决定的。
 A. 分配主存 B. 用户编程 C. 装作业 D. 程序执行
26. 下面的()方法有利于程序的动态链接。
 A. 分段存储管理 B. 分页存储管理
 C. 可变式分区管理 D. 固定式分区管理
27. 当前编程人员编写好的程序经过编译转换成目标文件后，各条指令的地址编号起始一般定为()，称为()地址。
 1) A. 1 B. 0 C. IP D. CS
 2) A. 绝对 B. 名义 C. 逻辑 D. 实
28. 可重入程序是通过()方法来改善系统性能的。
 A. 改变时间片长度 B. 改变用户数
 C. 提高对换速度 D. 减少对换数量
29. 操作系统实现()存储管理的代价最小。
 A. 分区 B. 分页 C. 分段 D. 段页式
30. 动态分区又称可变式分区，它是系统运行过程中()动态建立的。
 A. 在作业装入时 B. 在作业创建时
 C. 在作业完成时 D. 在作业未装入时
31. 对外存对换区的管理以()为主要目标。

- A. 提高系统吞吐量 B. 提高存储空间的利用率
 C. 降低存储费用 D. 提高换入、换出速度
32. 【2017 统考真题】某计算机按字节编址，其动态分区内存管理采用最佳适应算法，每次分配和回收内存后都对空闲分区链重新排序。当前空闲分区信息如下表所示。

分区始址	20K	500K	1000K	200K
分区大小	40KB	80KB	100KB	200KB

回收始址为 60K、大小为 140KB 的分区后，系统中空闲分区的数量、空闲分区链第一个分区的始址和大小分别是（ ）。

- A. 3, 20K, 380KB B. 3, 500K, 80KB C. 4, 20K, 180KB D. 4, 500K, 80KB
33. 下列关于虚拟存储器的论述中，正确的是（ ）。
- A. 作业在运行前，必须全部装入内存，且在运行过程中也一直驻留内存
 B. 作业在运行前，不必全部装入内存，且在运行过程中也不必一直驻留内存
 C. 作业在运行前，不必全部装入内存，但在运行过程中必须一直驻留内存
 D. 作业在运行前，必须全部装入内存，但在运行过程中不必一直驻留内存
34. 在页式存储管理中选择页面的大小，需要考虑下列（ ）因素。
- I. 页面大的好处是页表比较少
 II. 页面小的好处是可以减少由内碎片引起的内存浪费
 III. 影响磁盘访问时间的主要因素通常不是页面大小，所以使用时优先考虑较大的页面
 A. I 和 III B. II 和 III C. I 和 II D. I、II 和 III
35. 某个操作系统对内存的管理采用页式存储管理方法，所划分的页面大小（ ）。
- A. 要根据内存大小确定 B. 必须相同
 C. 要根据 CPU 的地址结构确定 D. 要依据外存和内存的大小确定
36. 引入段式存储管理方式，主要是为了更好地满足用户的一系列要求。下面选项中不属于这一系列要求的是（ ）。
- A. 方便操作 B. 方便编程
 C. 共享和保护 D. 动态链接和增长
37. 存储管理的目的是（ ）。
- A. 方便用户 B. 提高内存利用率
 C. 方便用户和提高内存利用率 D. 增加内存实际容量
38. 对主存储器的访问，（ ）。
- A. 以块（即页）或段为单位 B. 以字节或字为单位
 C. 随存储器的管理方案不同而异 D. 以用户的逻辑记录为单位
39. 把作业空间中使用的逻辑地址变为内存中的物理地址称为（ ）。
- A. 加载 B. 重定位 C. 物理化 D. 逻辑化
40. 以下存储管理方式中，不适合多道程序设计系统的是（ ）。
- A. 单用户连续分配 B. 固定式分区分配
 C. 可变式分区分配 D. 分页式存储管理方式
41. 在分页存储管理中，主存的分配（ ）。
- A. 以物理块为单位进行 B. 以作业的大小进行
 C. 以物理段进行 D. 以逻辑记录大小进行
42. 在段式分配中，CPU 每次从内存中取一次数据需要（ ）次访问内存。

- A. 1 B. 3 C. 2 D. 4
43. 在段页式分配中, CPU 每次从内存中取一次数据需要()次访问内存。
 A. 1 B. 3 C. 2 D. 4
44. ()存储管理方式提供一维地址结构。
 A. 分段 B. 分页
 C. 分段和段页式 D. 以上答案都不正确
45. 操作系统采用分页存储管理方式, 要求()。
 A. 每个进程拥有一张页表, 且进程的页表驻留在内存中
 B. 每个进程拥有一张页表, 但只有执行进程的页表驻留在内存中
 C. 所有进程共享一张页表, 以节约有限的内存空间, 但页表必须驻留在内存中
 D. 所有进程共享一张页表, 只有页表中当前使用的页面必须驻留在内存中, 以最大限度地节省有限的内存空间
46. 【2009统考真题】一个分段存储管理系统中, 地址长度为32位, 其中段号占8位, 则最大段长是()。
 A. 2^8B B. $2^{16}B$ C. $2^{24}B$ D. $2^{32}B$
47. 在分段存储管理方式中, ()。
 A. 以段为单位, 每段是一个连续存储区 B. 段与段之间必定不连续
 C. 段与段之间必定连续 D. 每段是等长的
48. 段页式存储管理汲取了页式管理和段式管理的长处, 其实现原理结合了页式和段式管理的基本思想, 即()。
 A. 用分段方法来分配和管理物理存储空间, 用分页方法来管理用户地址空间
 B. 用分段方法来分配和管理用户地址空间, 用分页方法来管理物理存储空间
 C. 用分段方法来分配和管理主存空间, 用分页方法来管理辅存空间
 D. 用分段方法来分配和管理辅存空间, 用分页方法来管理主存空间
49. 以下存储管理方式中, 会产生内部碎片的是()。
 I. 分段虚拟存储管理 II. 分页虚拟存储管理
 III. 段页式分区管理 IV. 固定式分区管理
 A. I、II、III B. III、IV C. 仅 II D. II、III、IV
50. 下列关于页式存储的论述中, 正确的是()。
 I. 在页式存储管理中, 若关闭 TLB, 则每当访问一条指令或存取一个操作数时都要访问2次内存
 II. 页式存储管理不会产生内部碎片
 III. 页式存储管理中的页面是为用户所感知的
 IV. 页式存储方式可以采用静态重定位
 A. I、II、IV B. I、IV C. 仅 I D. 全都正确
51. 【2010统考真题】某计算机采用二级页表的分页存储管理方式, 按字节编址, 页大小为 $2^{10}B$, 页表项大小为2B, 逻辑地址结构为

页目录号	页号	页内偏移量
------	----	-------

逻辑地址空间大小为 2^{16} 页, 则表示整个逻辑地址空间的页目录表中包含表项的个数至少是()。

- A. 64 B. 128 C. 256 D. 512

52. 【2014 统考真题】现有一个容量为 10GB 的磁盘分区，磁盘空间以簇为单位进行分配，簇的大小为 4KB，若采用位图法管理该分区的空闲空间，即用一位标识一个簇是否被分配，则存放该位图所需的簇为（ ）个。
- A. 80 B. 320 C. 80K D. 320K
53. 【2014 统考真题】下列选项中，属于多级页表优点的是（ ）。
- A. 加快地址变换速度 B. 减少缺页中断次数
C. 减少页表项所占字节数 D. 减少页表所占的连续内存空间
54. 【2016 统考真题】某进程的段表内容如下所示。

段号	段长	内存起始地址	权限	状态
0	100	6000	只读	在内存
1	200	—	读写	不在内存
2	300	4000	读写	在内存

- 访问段号为 2、段内地址为 400 的逻辑地址时，进行地址转换的结果是（ ）。
- A. 段缺失异常 B. 得到内存地址 4400
C. 越权异常 D. 越界异常
55. 【2019 统考真题】在分段存储管理系统中，用共享段表描述所有被共享的段。若进程 P₁ 和 P₂ 共享段 S，则下列叙述中，错误的是（ ）。
- A. 在物理内存中仅保存一份段 S 的内容
B. 段 S 在 P₁ 和 P₂ 中应该具有相同的段号
C. P₁ 和 P₂ 共享段 S 在共享段表中的段表项
D. P₁ 和 P₂ 都不再使用段 S 时才回收段 S 所占的内存空间
56. 【2019 统考真题】某计算机主存按字节编址，采用二级分页存储管理，地址结构如下：

页目录号（10 位）	页号（10 位）	页内偏移（12 位）
------------	----------	------------

- 虚拟地址 2050 1225H 对应的页目录号、页号分别是（ ）。
- A. 081H, 101H B. 081H, 401H C. 201H, 101H D. 201H, 401H
57. 【2019 统考真题】在下列动态分区分配算法中，最容易产生内存碎片的是（ ）。
- A. 首次适应算法 B. 最坏适应算法
C. 最佳适应算法 D. 循环首次适应算法

二、综合应用题

1. 动态分区与固定分区分配方式相比，是否解决了碎片问题？
2. 某系统的空闲分区见下表，采用可变式分区管理策略，现有如下作业序列：96KB, 20KB, 200KB。若用首次适应算法和最佳适应算法来处理这些作业序列，则哪种算法能满足该作业序列请求？为什么？

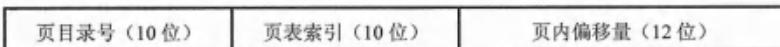
分区号	大小	始址
1	32KB	100K
2	10KB	150K
3	5KB	200K
4	218KB	220K
5	96KB	530K

3. 某操作系统采用段式管理，用户区主存为 512KB，空闲块链入空块表，分配时截取空块的前半部分（小地址部分）。初始时全部空闲。执行申请、释放操作序列 reg(300KB), reg(100KB), release(300KB), reg(150KB), reg(50KB), reg(90KB) 后：
- 1) 采用最先适配，空块表中有哪些空块？（指出大小及始址）
 - 2) 采用最佳适配，空块表中有哪些空块？（指出大小及始址）
 - 3) 若随后又要申请 80KB，针对上述两种情况会产生什么后果？这说明了什么问题？
4. 【2013 统考真题】某计算机主存按字节编址，逻辑地址和物理地址都是 32 位，页表项大小为 4B。请回答下列问题：
- 1) 若使用一级页表的分页存储管理方式，逻辑地址结构为



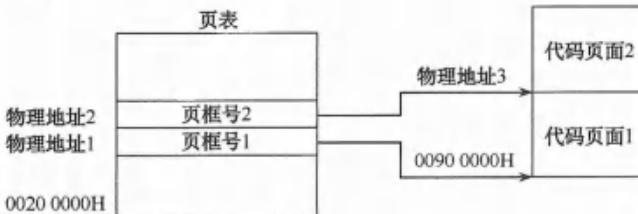
则页的大小是多少字节？页表最大占用多少字节？

- 2) 若使用二级页表的分页存储管理方式，逻辑地址结构为

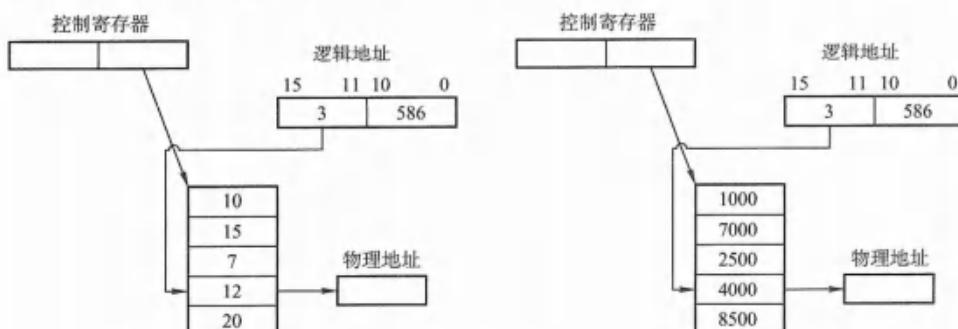


设逻辑地址为 LA，请分别给出其对应的页目录号和页表索引的表达式。

- 3) 采用 1) 中的分页存储管理方式，一个代码段的起始逻辑地址为 0000 8000H，其长度为 8 KB，被装载到从物理地址 0090 0000H 开始的连续主存空间中。页表从主存 0020 0000H 开始的物理地址处连续存放，如下图所示（地址大小自下向上递增）。请计算出该代码段对应的两个页表项的物理地址、这两个页表项中的页框号，以及代码页面 2 的起始物理地址。



5. 下图给出了页式和段式两种地址变换示意（假定段式变换对每段不进行段长越界检查，即段表中无段长信息）。
- 1) 指出这两种变换各属于何种存储管理。
 - 2) 计算出这两种变换所对应的物理地址。



6. 在一个段式存储管理系统中，其段表见下表 A。试求表 B 中的逻辑地址所对应的物理地址。

表 A 段表

段号	内存始址	段长
0	210	500
1	2350	20
2	100	90
3	1350	590
4	1938	95

表 B 逻辑地址

段号	段内位移
0	430
1	10
2	500
3	400
4	112
5	32

7. 页式存储管理允许用户的编程空间为 32 个页面 (每页 1KB), 主存为 16KB。如有一用户程序为 10 页长, 且某时刻该用户程序页表见右表。若分别遇到三个逻辑地址 0AC5H, 1AC5H, 3AC5H 处的操作, 计算并说明存储管理系统将如何处理。
8. 在某页式管理系统中, 假定主存为 64KB, 分成 16 块, 块号为 0, 1, 2, …, 15。设某进程有 4 页, 其页号为 0, 1, 2, 3, 被分别装入主存的第 9, 0, 1, 14 块。
- 该进程的总长度是多大?
 - 写出该进程每页在主存中的始址。
 - 若给出逻辑地址(0, 0), (1, 72), (2, 1023), (3, 99), 请计算出相应的内存地址 (括号内的第一个数为十进制页号, 第二个数为十进制页内地址)。
9. 某操作系统存储器采用页式存储管理, 页面大小为 64B, 假定一进程的代码段的长度为 702B, 页表见表 A, 该进程在快表中的页表见表 B。现进程有如下访问序列: 其逻辑地址为八进制的 0105, 0217, 0567, 01120, 02500。试问给定的这些地址能否进行转换?

表 A 进程页表

页号	页帧号	页号	页帧号
0	F0	6	F6
1	F1	7	F7
2	F2	8	F8
3	F3	9	F9
4	F4	10	F10
5	F5	6	F6

表 B 快表

页号	页帧号
0	F0
1	F1
2	F2
3	F3
4	F4

10. 某一页式系统, 其页表存放在主存中:
- 若对主存的一次存取需 $1.5\mu s$, 问实现一次页面访问时存取时间是多少?
 - 若系统有快表且其平均命中率为 85%, 而页表项在快表中的查找时间为 $0.2\mu s$, 若快表的命中率是 85%, 则有效存取时间是多少? 若快表的命中率为 50%, 则有效存取时间是多少?
11. 在页式、段式和段页式存储管理中, 当访问一条指令或数据时, 各需要访问内存几次? 其过程如何? 假设一个页式存储系统具有快表, 多数活动页表项都可以存在其中。若页表存放在内存中, 内存访问时间是 $1\mu s$, 检索快表的时间为 $0.2\mu s$, 若快表的命中率是 85%, 则有效存取时间是多少? 若快表的命中率为 50%, 则有效存取时间是多少?
12. 在一个分页存储管理系统中, 地址空间分页 (每页 1KB), 物理空间分块, 设主存总容

量是 256KB，描述主存分配情况的位示图如下图所示（0 表示未分配，1 表示已分配），此时作业调度程序选中一个长为 5.2KB 的作业投入内存。

- 1) 为该作业分配内存后（分配内存时，首先分配低地址的内存空间），填写该作业的页表内容。
- 2) 页式存储管理有无零头存在？若有，会存在什么零头？为该作业分配内存后，会产生零头吗？若产生，大小为多少？（提示：这里的零头是指一页中未被使用的部分。）
- 3) 假设一个 64MB 内存容量的计算机，其操作系统采用页式存储管理（页面大小为 4KB），内存分配采用位示图方式管理，请问位示图将占用多大的内存？

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1
1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	1	0	0	0	1	0	1
0	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1

页号	块号（从 0 开始编址）

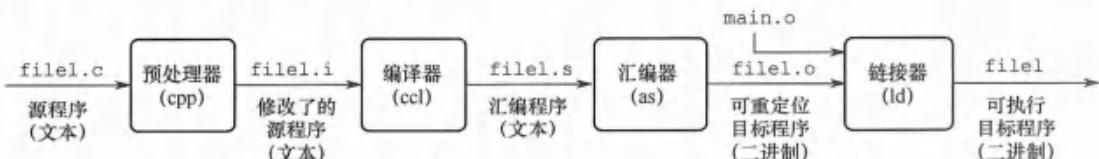
3.1.7 答案与解析

一、单项选择题

1. C

编译后的程序需要经过链接才能装载，而链接后形成的目标程序中的地址也就是逻辑地址。以 C 语言为例：C 程序经过预处理→编译→汇编→链接产生了可执行文件，其中链接的前一步是产生可重定位的二进制目标文件。C 语言采用源文件独立编译的方法，如程序 main.c, file1.c, file2.c, file1.h, file2.h 在链接的前一步生成了 main.o, file1.o, file2.o, 这些目标模块的逻辑地址都从 0 开始，但只是相对于该模块的逻辑地址。链接器将这三个文件、libc 和其库文件链接成一个可执行文件。链接阶段主要完成重定位，形成整个程序的完整逻辑地址空间。

例如，file1.o 的逻辑地址为 0~1023，main.o 的逻辑地址为 0~1023，假设链接时将 file1.o 链接在 main.o 之后，则链接之后 file1.o 对应的逻辑地址应为 1024~2047，整个过程如下图所示。



要注意审题，若题目是“完成该变换过程的阶段是”，则可以选 D。

2. D

选项 A、B 显然错误，选项 C 中编址空间的大小取决于硬件的访存能力，一般由地址总线宽度决定。选项 D 中虚拟内存的管理需要由相关的硬件和软件支持，有请求分页页表机制、缺页中断机构、地址变换机构等。

3. B

进程正在进行 I/O 操作时不能换出主存，否则其 I/O 数据区将被新换入的进程占用，导致错

误。不过可以在操作系统中开辟 I/O 缓冲区，将数据从外设输入或将数据输出到外设的 I/O 活动在系统缓冲区中进行，这时系统缓冲区与外设 I/O 时，进程交换不受限制。

4. A

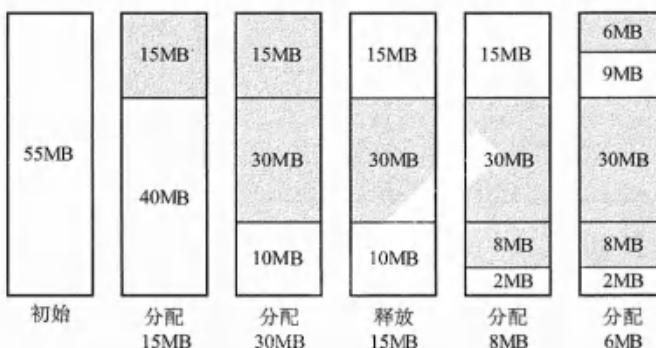
覆盖和交换的提出就是为了解决主存空间不足的问题，但不是在物理上扩充主存，只是将暂时不用的部分换出主存，以节省空间，从而在逻辑上扩充主存。

5. A

每个进程都拥有自己独立的进程空间，若一个进程在运行时所产生的地址在其地址空间之外，则发生地址越界，因此需要进行界地址保护，即当程序要访问某个内存单元时，由硬件检查是否允许，若允许则执行，否则产生地址越界中断。

6. B

最佳适配算法是指每次为作业分配内存空间时，总是找到能满足空间大小需要的最小空闲分区给作业，可以产生最小的内存空闲分区。下图显示了这个过程的主存空间变化。



图中，灰色部分为分配出去的空间，白色部分为空闲区。这样，容易发现，此时主存中最大空闲分区的大小为 9MB。

7. C

段页式系统中，进程首先划分为段，每段再进一步划分为页。

8. C

内存保护是内存管理的一部分，是操作系统的任务，但是出于安全性和效率考虑，必须由硬件实现，所以需要操作系统和硬件机构的合作来完成。

9. A

覆盖技术是早期在单一连续存储管理中使用的扩大存储容量的一种技术，它同样可用于固定分区分配的存储管理。

10. D

将上邻空闲区、下邻空闲区和回收区合并为一个空闲区，因此空闲区数反而减少了一个。而仅有上邻空闲区或下邻空闲区时，空闲区数并不减少。

11. C

最佳适配算法是指，每次为作业分配内存空间时，总是找到能满足空间大小需要的最小空闲分区给作业，可以产生最小的内存空闲分区。从图中可以看出应选择大小为 60KB 的空闲分区，其首地址为 330K。

12. B

段号为 2，其对应的首地址为 480K，段长度为 20K，大于 154，所以逻辑地址(2, 154)对应的物理地址为 480K + 154。

13. D

静态装入是指在编程阶段就把物理地址计算好。

可重定位是指在装入时把逻辑地址转换成物理地址，但装入后不能改变。

动态重定位是指在执行时再决定装入的地址并装入，装入后有可能会换出，所以同一个模块在内存中的物理地址是可能改变的。

动态重定位是指在作业运行过程中执行到一条访存指令时，再把逻辑地址转换为主存中的物理地址，实际中是通过硬件地址转换机制实现的。

14. A

固定分区方式中，作业装入后位置不再改变，可以采用静态重定位。其余三种管理方案均可能在运行过程中改变程序位置，静态重定位不能满足其要求。

15. A

在可变分区管理中，回收空闲区时采用拼接技术对空闲区进行合并。

16. A

按页表内容可知，逻辑地址 0 对应块号 2，页大小为 4KB，因此转换成的物理地址为 $2 \times 4K = 8K = 8192$ 。

17. B

分页式存储管理有内部碎片，分段式存储管理有外部碎片，固定分区存储管理方式有内部碎片，段页式存储管理方式有内部碎片。

18. B

多进程的执行通过内存保护实现互不干扰，如页式管理中有页地址越界保护，段式管理中有段地址越界保护。

19. A

最佳适应算法要求从剩余的空闲分区中选出最小且满足存储要求的分区，空闲区应按长度递增登记在空闲区表中。

20. C

首次适应算法的空闲分区按地址递增的次序排列。

21. C

页表和段表同样存储在内存中，系统提供给用户的物理地址空间为总空间大小减去页表或段表的长度。由于页表和段表的长度不能确定，所以提供给用户的物理地址空间大小也不能确定。

22. B

内存分页管理是在硬件和操作系统层面实现的，对用户、编译系统、连接装配程序等上层是不可见的。

23. D

页表的功能由一组专门的存储器实现，其始址放在页表基址寄存器（PTBR）中。这样才能满足在地址变换时能够较快地完成逻辑地址和物理地址之间的转换。

24. A

为使地址转换不影响到指令的执行速度，必须有硬件地址变换结构的支持，即需在系统中增设一个重定位寄存器，用它来存放程序（数据）在内存中的始址。在执行程序或访问数据时，真正访问的内存地址由相对地址与重定位寄存器中的地址相加而成，这时将始址存入重定位寄存器，之后的地址访问即可通过硬件变换实现。因为系统处理器在同一时刻只能执行一条指令或访问数据，所以为每道程序（数据）设置一个寄存器没有必要（同时也不现实，因为寄存器是很昂贵的硬件，而且程序的道数是无法预估的），而只需在切换程序执行时重置寄存器内容。

25. B

分段是指在用户编程时，将程序按照逻辑划分为几个逻辑段。

26. A

程序的动态链接与程序的逻辑结构相关，分段存储管理将程序按照逻辑段进行划分，因此有利于其动态链接。其他的内存管理方式与程序的逻辑结构无关。

27. B、C

编译后一个目标程序所限定的地址范围称为该作业的逻辑地址空间。换句话说，地址空间仅指程序用来访问信息所用的一系列地址单元的集合。这些单元的编号称为逻辑地址。通常，编译地址都是相对始址“0”的，因而也称逻辑地址为相对地址。

注意：区分编译后形成的逻辑地址和链接后形成的最终逻辑地址。

28. D

可重入程序主要是通过共享来使用同一块存储空间的，或通过动态链接的方式将所需的程序段映射到相关进程中去，其最大的优点是减少了对程序段的调入/调出，因此减少了对换数量。

29. A

实现分页、分段和段页式存储管理需要特定的数据结构支持，如页表、段表等。为了提高性能，还需要硬件提供快存和地址加法器等，代价高。分区存储管理是满足多道程序设计的最简单的存储管理方案，特别适合嵌入式等微型设备。

30. A

动态分区时，在系统启动后，除操作系统占据一部分内存外，其余所有内存空间是一个大空闲区，称为自由空间。若作业申请内存，则从空闲区中划出一个与作业需求量相适应的分区分配给该作业，将作业创建为进程，在作业运行完毕后，再收回释放的分区。

31. D

操作系统在内存管理中为了提高内存的利用率，引入了覆盖和交换技术，即在较小的内存空间中采用重复使用的方法来节省存储空间，但它付出的代价是需要消耗更多的处理器时间，因此实际上是一种以时间换空间的技术。为此，从节省处理器时间来讲，换入、换出速度越快，付出的时间代价就越小，反之就越大，大到一定程度时，覆盖和交换技术就没有意义。

32. B

回收始址为 60K、大小为 140KB 的分区时，它与表中第一个分区和第四个分区合并，成为始址为 20K、大小为 380KB 的分区，剩余 3 个空闲分区。在回收内存后，算法会对空闲分区链按分区大小由小到大进行排序，表中的第二个分区排第一，所以选择 B。

33. B

在非虚拟存储器中，作业必须全部装入内存且在运行过程中也一直驻留内存；在虚拟存储器中，作业不必全部装入内存且在运行过程中也不用一直驻留内存，这是虚拟存储器和非虚拟存储器的主要区别之一。

34. C

页面大，用于管理页面的页表就少，但是页内碎片会比较大；页面小，用于管理页面的页表就大，但是页内碎片小。通过适当的计算可以获得较佳的页面大小和较小的系统开销。

35. B

页式管理中很重要的一个问题就是页面大小如何确定。确定页面大小有很多因素，如进程的平均大小、页表占用的长度等。而一旦确定，所有的页面就是等长的（一般取 2 的整数幂倍），以

便易于系统管理。

36. A

引入段式存储管理方式，主要是为了满足用户的下列要求：方便编程、分段共享、分段保护、动态链接和动态增长。

37. C

存储管理的目的有两个：一个是方便用户，二是提高内存利用率。

38. B

这里是指主存的访问，不是主存的分配。对主存的访问是以字节或字为单位的。例如，在页式管理中，不仅要知道块号，而且要知道页内偏移。

39. B

在一般情况下，一个作业在装入时分配到的内存空间和它的地址空间是不一致的，因此，作业在 CPU 上运行时，其所要访问的指令、数据的物理地址和逻辑地址是不同的。显然，若在作业装入或执行时，不对有关的地址部分加以相应的修改，则会导致错误的结果。这种将作业的逻辑地址变为物理地址的过程称为地址重定位。

40. A

单用户连续分配管理方式只适用于单用户、单任务的操作系统，不适用于多道程序设计。

41. A

在分页存储管理中，逻辑地址分配是按页为单位进行分配的，而主存的分配即物理地址分配是以内存块为单位分配的。

42. C

在段式分配中，取一次数据时先从内存查找段表，再拼成物理地址后访问内存，共需要 2 次内存访问。

43. B

在段页式分配中，取一次数据时先从内存查找段表，再访问内存查找相应的页表，最后拼成物理地址后访问内存，共需要 3 次内存访问。

44. B

分页存储管理中，作业地址空间是一维的，即单一的线性地址空间，程序员只需要一个记忆符来表示地址。在分段存储分配管理中，段之间是独立的，而且段长不定长，而页长是固定的，因此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。简言之，确定一个地址需要几个参数，作业地址空间就是几维的。

45. A

在多个进程并发执行时，所有进程的页表大多数驻留在内存中，在系统中只设置一个页表寄存器（PTR），它存放页表在内存中的始址和长度。平时，进程未执行时，页表的始址和页表长度存放在本进程的 PCB 中，当调度到某进程时，才将这两个数据装入页表寄存器中。每个进程都有一个单独的逻辑地址，有一张属于自己的页表。

46. C

分段存储管理的逻辑地址分为段号和位移量两部分，段内位移的最大值就是最大段长。地址长度为 32 位，段号占 8 位，因此位移量占 $32 - 8 = 24$ 位，因此最大段长为 2^{24} B。

47. A

在分段存储管理方式中，以段为单位进行分配，每段是一个连续存储区，每段不一定等长，段与段之间可连续，也可不连续。

48. B

段页式存储管理兼有页式管理和段式管理的优点，采用分段方法来分配和管理用户地址空间，采用分页方法来管理物理存储空间。但它的开销要比段式和页式管理的开销大。

49. D

只要是固定的分配就会产生内部碎片，其余的都会产生外部碎片。若固定和不固定同时存在（例如段页式），则仍视为固定。分段虚拟存储管理：每段的长度都不一样（对应不固定），所以会产生外部碎片。分页虚拟存储管理：每页的长度都一样（对应固定），所以会产生内部碎片。段页式分区管理：既有固定，又有不固定，以固定为主，所以会有内部碎片。固定式分区管理：很明显固定，会产生内部碎片。

综上分析，II、III、IV 选项会产生内部碎片。

50. C

I 正确：关闭 TLB 后，每当访问一条指令或存取一个操作数时都要先访问页表（内存中），得到物理地址后，再访问一次内存进行相应操作。II 错误：记住，凡是分区固定的都会产生内部碎片，而无外部碎片。III 错误：页式存储管理对于用户是透明的。IV 错误：静态重定位是在程序运行之前由装配程序完成的，必须分配其要求的全部连续内存空间。而页式存储管理方案是将程序离散地分成若干页（块），从而可以将程序装入不连续的内存空间，显然静态重定位不能满足其要求。

51. B

页大小为 2^{10} B，页表项大小为 2B，因此一页可以存放 2^9 个页表项，逻辑地址空间大小为 2^{16} 页，即共需 $2^{16}/2^9 = 2^7 = 128$ 个页面保存页表项，即页目录表中包含表项的个数至少是 128。

52. A

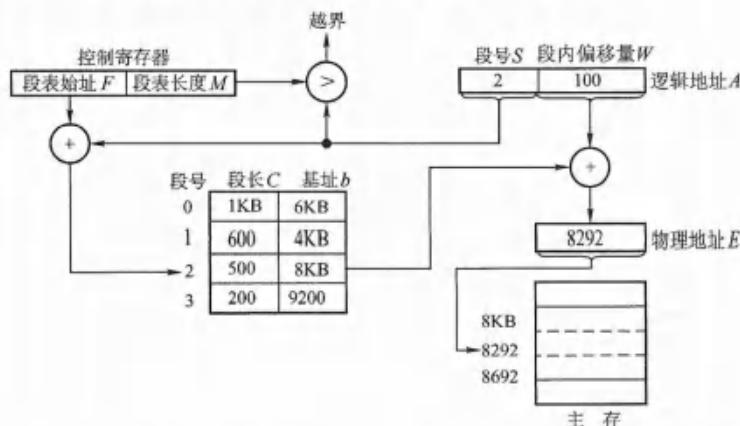
簇的总数为 $10\text{GB}/4\text{KB} = 2.5\text{M}$ ，用一位标识一簇是否被分配，整个磁盘共需要 2.5Mb ，即需要 $2.5\text{M}/8 = 320\text{KB}$ ，因此共需要 $320\text{KB}/4\text{KB} = 80$ 簇，选 A。

53. D

多级页表不仅不会加快地址的变换速度，还会因为增加更多的查表过程，使地址变换速度减慢；也不会减少缺页中断的次数，反而如果访问过程中多级的页表都不在内存中，会大大增加缺页的次数，并不会减少页表项所占的字节数，而多级页表能够减少页表所占的连续内存空间，即当页表太大时，将页表再分级，把每张页表控制在一页之内，减少页表所占的连续内存空间，因此选 D。

54. D

分段系统的逻辑地址 A 到物理地址 E 之间的地址变换过程如下：



① 从逻辑地址 A 中取出前几位为段号 S，后几位为段内偏移量 W，注意段式存储管理的题

目中，逻辑地址一般以二进制数给出，而在页式存储管理中，逻辑地址一般以十进制数给出，读者要具体问题具体分析。

- ② 比较段号 S 和段表长度 M ，若 $S \geq M$ ，则产生越界异常，否则继续执行。
- ③ 段表中段号 S 对应的段表项地址 = 段表始址 F + 段号 $S \times$ 段表项长度 M ，取出该段表项的前几位得到段长 C 。若段内偏移量 $\geq C$ ，则产生越界异常，否则继续执行。从这句话可以看出，段表项实际上只有两部分，前几位是段长，后几位是始址。
- ④ 取出段表项中该段的基址 b ，计算 $E = b + W$ ，用得到的物理地址 E 去访问内存。

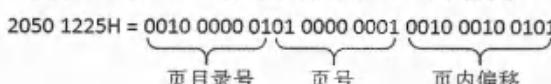
题目中段号为 2 的段长为 300，小于段内地址 400，因此发生越界异常，D 正确。

55. B

段的共享是通过两个作业的段表中相应表项指向被共享的段的同一个物理副本实现的，因此在内存中仅保存一份段 S 的内容，A 正确。段 S 对于进程 P_1, P_2 来说，使用位置可能不同，所以在不同进程中的逻辑段号可能不同，B 错误。段表项存放的是段的物理地址（包括段始址和段长度），对于共享段 S 来说物理地址唯一，C 正确。为了保证进程可以顺利使用段 S ，段 S 必须确保在没有任何进程使用它（可在段表项中设置共享进程计数）后才能被删除，D 正确。

56. A

题中给出的是十六进制地址，首先将它转化为二进制地址，然后用二进制地址去匹配题中对应的地址结构。转换为二进制地址和地址结构的对应关系如下图所示。



前 10 位、11~20 位、21~32 位分别对应页目录号、页号和页内偏移。把页目录号、页号单独拿出，转换为十六进制时缺少的位数在高位补零，0000 1000 0001，0001 0000 0001 分别对应 081H，101H，选项 A 正确。

57. C

最佳适应算法总是匹配与当前大小要求最接近的空闲分区，但是大多数情况下空闲分区的大小不可能完全和当前要求的大小相等，几乎每次分配内存都会产生很小的难以利用的内存块，所以最佳适应算法最容易产生最多的内存碎片，C 正确。

二、综合应用题

1. 解答：

动态分区和固定分区分配方式相比，内存空间的利用率要高一些。但是，总会存在一些分散的较小空闲分区，即外部碎片，它们存在于已分配的分区之间，不能充分利用。可以采用拼接技术加以解决。固定分区分配方式存在内部碎片，而无外部碎片；动态分区分配方式存在外部碎片，无内部碎片。

2. 解答：

采用首次适应算法时，96KB 大小的作业进入 4 号空闲分区，20KB 大小的作业进入 1 号空闲分区，这时空闲分区如下表所示。

分区号	大 小	始 址
1	12KB	120K
2	10KB	150K
3	5KB	200K
4	122KB	316K
5	96KB	530K

此时再无空闲分区可以满足 200KB 大小的作业，所以该作业序列请求无法满足。

采用最佳适应算法时，作业序列分别进入 5, 1, 4 号空闲分区，可以满足其请求。分配处理之后的空闲分区表见下表：

分区号	大 小	始 址
1	12KB	120K
2	10KB	150K
3	5KB	200K
4	18KB	420K

3. 解答：

1) 最先适配的内存分配情况如下图中的(a)所示。

内存中的空块为：

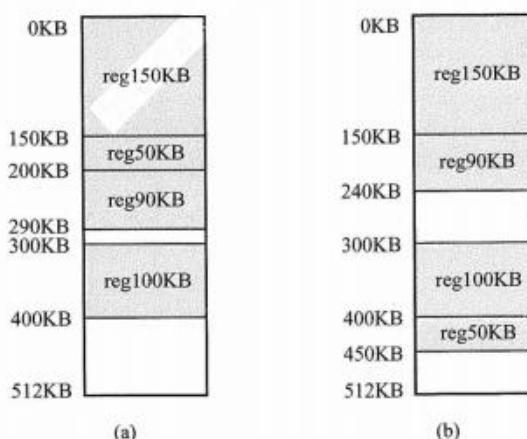
第一块：始址 290K，大小 10KB；第二块：始址 400K，大小 112KB。

2) 最佳适配的内存分配情况如下图中的(b)所示。

内存中的空块为：

第一块：始址 240K，大小 60KB；第二块：超始地址 450K，大小 62KB。

3) 若随后又要申请 80KB，则最先适配算法可以分配成功，而最佳适配算法则没有足够大的空闲区分配。这说明最先适配算法尽可能地使用了低地址部分的空闲区域，留下了高地址部分的大的空闲区，更有可能满足进程的申请。



4. 解答：

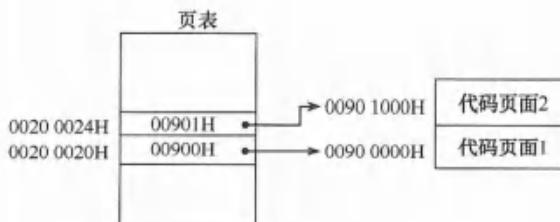
1) 因为主存按字节编址，页内偏移量是 12 位，所以页大小为 $2^{12}B = 4KB$ 。

页表项数为 2^{20} ，因此该一级页表最大为 $2^{20} \times 4B = 4MB$ 。

2) 页目录号可表示为 $((unsigned\ int)(LA)) >> 22 \& 0x3FF$ 。

页表索引可表示为 $((unsigned\ int)(LA)) >> 12 \& 0x3FF$ 。

3) 代码页面 1 的逻辑地址为 0000 8000H，表明其位于第 8 个页处，对应页表中的第 8 个页表项，所以第 8 个页表项的物理地址 = 页表始址 + 8×页表项的字节数 = 0020 0000H + 8×4 = 0020 0020H。由此可得如下图所示的答案。



5. 解答：

- 1) 由题图所示的逻辑地址结构可知：页或段的最大个数为 $2^5 = 32$ 。如果左图是段式管理，那么段始址 12 加上偏移量 586，远超第 1 段的段始址 15，超过第 4 段的段始址 20，所以左图是页式变换，而右图满足段式变换。对于页式管理，由逻辑地址的位移量位数可知，一页的大小为 2KB。
- 2) 对图中的页式地址变换，其物理地址为 $12 \times 2048 + 586 = 25162$ ；对图中的段式地址变换，其物理地址为 $4000 + 586 = 4586$ 。

6. 解答：

- 1) 由段表知，第 0 段内存始址为 210，段长为 500，因此逻辑地址(0, 430)是合法地址，对应的物理地址为 $210 + 430 = 640$ 。
- 2) 由段表知，第 1 段内存始址为 2350，段长为 20，因此逻辑地址(1, 10)是合法地址，对应的物理地址为 $2350 + 10 = 2360$ 。
- 3) 由段表知，第 2 段内存始址为 100，段长为 90，逻辑地址(2, 500)的段内位移 500 超过了段长，因此为非法地址。
- 4) 由段表知，第 3 段内存始址为 1350，段长为 590，因此逻辑地址(3, 400)是合法地址，对应的物理地址为 $1350 + 400 = 1750$ 。
- 5) 由段表知，第 4 段内存始址为 1938，段长为 95，逻辑地址(4, 112)的段内位移 112 超过了段长，因此为非法地址。
- 6) 由段表知，不存在第 5 段，因此逻辑地址(5, 32)为非法地址。

7. 解答：

页面大小为 1KB，所以低 10 位为页内偏移地址；用户编程空间为 32 个页面，即逻辑地址高 5 位为虚页号；主存为 16 个页面，即物理地址高 4 位为物理块号。

逻辑地址 0AC5H 转换为二进制是 000 1010 1100 0101B，虚页号为 2(00010B)，映射至物理块号 4，因此系统访问物理地址 12C5H(01 0010 1100 0101B)。

逻辑地址 1AC5H 转换为二进制是 001 1010 1100 0101B，虚页号为 6(00110B)，不在页面映射表中，会产生缺页中断，系统进行缺页中断处理。

逻辑地址 3AC5H 转换为二进制是 011 1010 1100 0101B，页号为 14，而该用户程序只有 10 页，因此系统产生越界中断。

注意：在将十六进制地址转换为二进制地址时，我们可能会习惯性地写为 16 位，这是容易犯错的细节。例如，题中的逻辑地址为 15 位，物理地址为 14 位。逻辑地址 0AC5H 的二进制表示为 000 1010 1100 0101B，对应物理地址 12C5H 的二进制表示为 01 0010 1100 0101B。这一点应该引起注意。

8. 解答：

- 1) 页面的大小为 $(64/16)KB = 4KB$ ，该进程共有 4 页，所以该进程的总长度为 $4 \times 4KB = 16KB$ 。
- 2) 页面大小为 4KB，因此低 12 位为页内偏移地址；主存分为 16 块，因此内存物理地址高

4 位为主存块号。

页号为 0 的页面被装入主存的第 9 块，因此该地址在内存中的始址为 **1001 0000 0000 0000B**，即 **9000H**。

页号为 1 的页面被装入主存的第 0 块，因此该地址在内存中的始址为 **0000 0000 0000 0000B**，即 **0000H**。

页号为 2 的页面被装入主存的第 1 块，因此该地址在内存中的始址为 **0001 0000 0000 0000B**，即 **1000H**。

页号为 3 的页面被装入主存的第 14 块，因此该地址在内存中的始址为 **1110 0000 0000 0000B**，即 **E000H**。

3) 逻辑地址为(0, 0)，因此内存地址为 $(9, 0) = \text{1001 } 0000 \ 0000 \ 0000B$ ，即 **9000H**。

逻辑地址为(1, 72)，因此内存地址为 $(0, 72) = \text{0000 } 0000 \ 0100 \ 1000B$ ，即 **0048H**。

逻辑地址为(2, 1023)，因此内存地址为 $(1, 1023) = \text{0001 } 0011 \ 1111 \ 1111B$ ，即 **13FFH**。

逻辑地址为(3, 99)，因此内存地址为 $(14, 99) = \text{1110 } 0000 \ 0110 \ 0011B$ ，即 **E063H**。

9. 解答：

要注意题目中的逻辑地址使用哪种进制的数给出，若是十进制，则一般通过整数除法和求余得到页号和页内偏移，若用其他进制给出，则一般转换成二进制，然后按照地址结构划分为页号部分和页内偏移部分，再把页号和页内偏移计算出来。

页面大小为 64B，因此页内位移为 6 位，进程代码段长度为 702B，因此需要 11 个页面，编号为 0~10。

- 1) 八进制逻辑地址 0105 的二进制表示为 0 0100 0101B。逻辑页号为 1，此页号可在快表中查找到，得页帧号为 F1；页内位移为 5，因此物理地址为(F1, 5)。
- 2) 八进制逻辑地址 0217 的二进制表示为 0 1000 1111B。逻辑页号为 2，此页号可在快表中查找到，得页帧号为 F2；页内位移为 15，因此物理地址为(F2, 15)。
- 3) 八进制逻辑地址 0567 的二进制表示为 1 0111 0111B。逻辑页号为 5，此页号不在快表中，在内存页表中可以查找到，得页帧号为 F5；页内位移为 55，因此物理地址为(F5, 55)。
- 4) 八进制逻辑地址 01120 的二进制表示为 0010 0101 0000B。逻辑页号为 9，此页号不在快表中，在内存页表中可以查找到，得页帧号为 F9；页内位移为 16，因此物理地址为(F9, 16)。
- 5) 八进制逻辑地址 02500 的二进制表示为 0101 0100 0000B。逻辑页号为 21，此页号已超过页表的最大页号 10，因此产生越界中断。

注意：根据题中条件无法得知逻辑地址位数，所以在其二进制表示中，其位数并不一致，只是根据八进制表示进行转换。若已知逻辑地址空间大小或位数，则二进制表示必须保持一致。

10. 解答：

页表在主存时，实现一次存取需要访问主存两次：第一次是访问页表，获得所需访问数据所在页面的物理地址；第二次才是根据这个物理地址存取数据。

1) 因为页表在主存，所以 CPU 必须访问主存两次，即实现一次页面访问的存取时间是

$$1.5 \times 2 = 3\mu\text{s}$$

2) 系统增加快表后，在快表中找到页表项的概率为 85%，所以实现一次页面访问的存取时间为

$$0.85 \times (0 + 1.5) + (1 - 0.85) \times 2 \times 1.5 = 1.725\mu\text{s}$$

11. 解答：

1) 在页式存储管理中，访问指令或数据时，首先要访问内存中的页表，查找到指令或数据

所在页面对应的页表项，然后根据页表项查找访问指令或数据所在的内存页面。需要访问内存 2 次。

段式存储管理同理，需要访问内存 2 次。

段页式存储管理，首先要访问内存中的段表，然后访问内存中的页表，最后访问指令或数据所在的内存页面，需要访问内存 3 次。

对于比较复杂的情况，如多级页表，若页表划分为 N 级，则需要访问内存 $N + 1$ 次。若系统中有快表，则在快表命中时，只需要访问内存 1 次。

2) 按 1) 中的访问过程分析，有效存取时间为

$$(0.2 + 1) \times 85\% + (0.2 + 1 + 1) \times (1 - 85\%) = 1.35\mu s$$

3) 同理可计算得

$$(0.2 + 1) \times 50\% + (0.2 + 1 + 1) \times (1 - 50\%) = 1.7\mu s$$

从结果可以看出，快表的命中率对访存时间影响非常大。当命中率从 85% 降低到 50% 时，有效存取时间增加 $0.35\mu s$ 。因此在页式存储系统中，应尽可能地提高快表的命中率，从而提高系统效率。

注意：在有快表的分页存储系统中，计算有效存取时间时，需注意访问快表与访问内存的时间关系。通常的系统中，先访问快表，未命中时再访问内存；在有些系统中，快表与内存的访问同时进行，当快表命中时就停止对内存的访问。这里题中未具体指明，我们按照前者进行计算。但若题中有具体的说明，计算时就应注意区别。

12. 解答：

1) 位示图利用二进制的一位来表示磁盘中一个盘块的使用情况，其值为“0”时表示对应盘块空闲，为“1”时表示已分配，地址空间分页，每页为 1KB，对应的盘块大小也为 1KB，主存总容量为 256KB，可分成 256 个盘块，长 5.2KB 的作业需要占用 6 页空间，假设页号与物理块号都从 0 开始，则根据位示图可得到页表内容如下：

页 号	块 号
0	21
1	27
2	28
3	29
4	34
5	35

- 2) 页式存储管理中有零头的存在，会存在内零头。为该作业分配内存后，会产生零头，因为此作业大小为 5.2KB，占 6 页，前 5 页满，最后一页只占 0.2KB 的空间，因此零头大小为 $1KB - 0.2KB = 0.8KB$ 。
- 3) 64MB 内存，一页大小为 4KB，共可分成 $64K \times 1K / 4K = 2^{14}$ 个物理盘块，在位示图中每个盘块占 1 位，共占 2^{14} 位空间，因为 $1B = 8$ 位，所以此位示图共占 2KB 空间的内存。

3.2 虚拟内存管理

在学习本节时，请读者思考以下问题：

- 1) 为什么要引入虚拟内存？

2) 虚拟内存空间的大小由什么因素决定?

3) 虚拟内存是怎么解决问题的? 会带来什么问题?

读者要掌握虚拟内存解决问题的思想, 并了解几种替换算法的优劣, 熟练掌握虚实地址的变换方法。

3.2.1 虚拟内存的基本概念

1. 传统存储管理方式的特征

3.1 节讨论的各种内存管理策略都是为了同时将多个进程保存在内存中, 以便允许进行多道程序设计。它们都具有以下两个共同的特征:

- 1) 一次性。作业必须一次性全部装入内存后, 才能开始运行。这会导致两种情况: ① 当作业很大而不能全部被装入内存时, 将使该作业无法运行; ② 当大量作业要求运行时, 由于内存不足以容纳所有作业, 只能使少数作业先运行, 导致多道程序度的下降。
- 2) 驻留性。作业被装入内存后, 就一直驻留在内存中, 其任何部分都不会被换出, 直至作业运行结束。运行中的进程会因等待 I/O 而被阻塞, 可能处于长期等待状态。

由以上分析可知, 许多在程序运行中不用或暂时不用的程序(数据)占据了大量的内存空间, 而一些需要运行的作业又无法装入运行, 显然浪费了宝贵的内存资源。

2. 局部性原理

要真正理解虚拟内存技术的思想, 首先须了解著名的局部性原理。Bill Joy (SUN 公司 CEO) 说过: “在研究所时, 我经常开玩笑地说高速缓存是计算机科学中唯一重要的思想。事实上, 高速缓存技术确实极大地影响了计算机系统的设计。”快表、页高速缓存及虚拟内存技术从广义上讲, 都属于高速缓存技术。这个技术所依赖的原理就是局部性原理。局部性原理既适用于程序结构, 又适用于数据结构(更远地讲, Dijkstra 关于“goto 语句有害”的著名论文也出于对程序局部性原理的深刻认识和理解)。

局部性原理表现在以下两个方面:

- 1) 时间局部性。程序中的某条指令一旦执行, 不久后该指令可能再次执行; 某数据被访问过, 不久后该数据可能再次被访问。产生时间局部性的典型原因是程序中存在着大量的循环操作。
- 2) 空间局部性。一旦程序访问了某个存储单元, 在不久后, 其附近的存储单元也将被访问, 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之内, 因为指令通常是顺序存放、顺序执行的, 数据也一般是以向量、数组、表等形式簇聚存储的。

时间局部性通过将近来使用的指令和数据保存到高速缓冲存储器中, 并使用高速缓存的层次结构实现。空间局部性通常使用较大的高速缓存, 并将预取机制集成到高速缓存控制逻辑中实现。虚拟内存技术实际上建立了“内存-外存”的两级存储器结构, 利用局部性原理实现高速缓存。

3. 虚拟存储器的定义和特征

基于局部性原理, 在程序装入时, 将程序的一部分装入内存, 而将其余部分留在外存, 就可启动程序执行。在程序执行过程中, 当所访问的信息不在内存时, 由操作系统将所需要的部分调入内存, 然后继续执行程序。另一方面, 操作系统将内存中暂时不使用的内容换出到外存上, 从而腾出空间存放将要调入内存的信息。这样, 系统好像为用户提供了一个比实际内存大得多的存储器, 称为虚拟存储器。

之所以将其称为虚拟存储器, 是因为这种存储器实际上并不存在, 只是由于系统提供了部分

装入、请求调入和置换功能后（对用户完全透明），给用户的感觉是好像存在一个比实际物理内存大得多的存储器。虚拟存储器的大小由计算机的地址结构决定，并不是内存和外存的简单相加。虚拟存储器有以下三个主要特征：

- 1) 多次性。多次性是指无须在作业运行时一次性地全部装入内存，而允许被分成多次调入内存运行。
- 2) 对换性。对换性是指无须在作业运行时一直常驻内存，而允许在作业的运行过程中，进行换进和换出。
- 3) 虚拟性。虚拟性是指从逻辑上扩充内存的容量，使用户所看到的内存容量远大于实际的内存容量。

4. 虚拟内存技术的实现

虚拟内存技术允许将一个作业分多次调入内存。采用连续分配方式时，会使相当一部分内存空间都处于暂时或“永久”的空闲状态，造成内存资源的严重浪费，而且也无法从逻辑上扩大内存容量。因此，虚拟内存的实现需要建立在离散分配的内存管理方式的基础上。

虚拟内存的实现有以下三种方式：

- 请求分页存储管理。
- 请求分段存储管理。
- 请求段页式存储管理。

不管哪种方式，都需要有一定的硬件支持。一般需要的支持有以下几个方面：

- 一定容量的内存和外存。
- 页表机制（或段表机制），作为主要的数据结构。
- 中断机构，当用户程序要访问的部分尚未调入内存时，则产生中断。
- 地址变换机构，逻辑地址到物理地址的变换。

3.2.2 请求分页管理方式

请求分页系统建立在基本分页系统基础之上，为了支持虚拟存储器功能而增加了请求调页功能和页面置换功能。请求分页是目前最常用的一种实现虚拟存储器的方法。

在请求分页系统中，只要求将当前需要的一部分页面装入内存，便可以启动作业运行。在作业执行过程中，当所要访问的页面不在内存中时，再通过调页功能将其调入，同时还可通过置换功能将暂时不用的页面换出到外存上，以便腾出内存空间。

为了实现请求分页，系统必须提供一定的硬件支持。除了需要一定容量的内存及外存的计算机系统，还需要有页表机制、缺页中断机构和地址变换机构。

1. 页表机制

请求分页系统的页表机制不同于基本分页系统，请求分页系统在一个作业运行之前不要求全部一次性调入内存，因此在作业的运行过程中，必然会出现要访问的页面不在内存中的情况，如何发现和处理这种情况是请求分页系统必须解决的两个基本问题。为此，在请求页表项中增加了4个字段，如图3.20所示。

页号	物理块号	状态位 P	访问字段 A	修改位 M	外存地址
----	------	-------	--------	-------	------

图3.20 请求分页系统中的页表项

增加的4个字段说明如下：

- 状态位 P 。用于指示该页是否已调入内存，供程序访问时参考。
- 访问字段 A 。用于记录本页在一段时间内被访问的次数，或记录本页最近已有多长时间未被访问，供置换算法换出页面时参考。
- 修改位 M 。标识该页在调入内存后是否被修改过。
- 外存地址。用于指出该页在外存上的地址，通常是物理块号，供调入该页时参考。

2. 缺页中断机构

在请求分页系统中，每当所要访问的页面不在内存中时，便产生一个缺页中断，请求操作系统将所缺的页调入内存。此时应将缺页的进程阻塞（调页完成唤醒），若内存中有空闲块，则分配一个块，将要调入的页装入该块，并修改页表中的相应页表项，若此时内存中没有空闲块，则要淘汰某页（若被淘汰页在内存期间被修改过，则要将其写回外存）。

缺页中断作为中断，同样要经历诸如保护 CPU 环境、分析中断原因、转入缺页中断处理程序、恢复 CPU 环境等几个步骤。但与一般的中断相比，它有以下两个明显的区别：

- 在指令执行期间而非一条指令执行完后产生和处理中断信号，属于内部中断。
- 一条指令在执行期间，可能产生多次缺页中断。

3. 地址变换机构

请求分页系统中的地址变换机构，是在分页系统地址变换机构的基础上，为实现虚拟内存，又增加了某些功能而形成的。

如图 3.21 所示，在进行地址变换时，先检索快表：

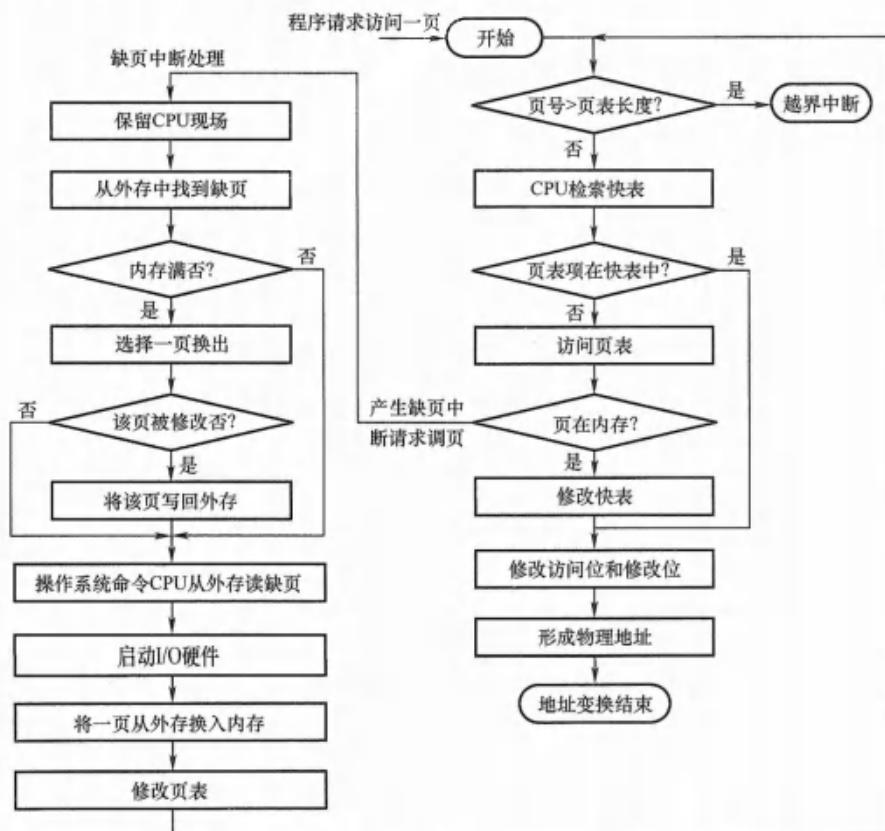


图 3.21 请求分页中的地址变换过程

- 若找到要访问的页，则修改页表项中的访问位（写指令还需要重置修改位），然后利用页表项中给出的物理块号和页内地址形成物理地址。
- 若未找到该页的页表项，则应到内存中去查找页表，再对比页表项中的状态位 P ，看该页是否已调入内存，未调入则产生缺页中断，请求从外存把该页调入内存。

3.2.3 页面置换算法（决定应该换入哪页、换出哪页）

进程运行时，若其访问的页面不在内存中而需将其调入，但内存已无空闲空间时，就需要从内存中调出一页程序或数据，送入磁盘的对换区。

选择调出页面的算法就称为页面置换算法。好的页面置换算法应有较低的页面更换频率，也就是说，应将以后不会再访问或以后较长时间内不会再访问的页面先调出。

常见的置换算法有以下 4 种。

1. 最佳 (OPT) 置换算法

最佳 (Optimal, OPT) 置换算法选择的被淘汰页面是以后永不使用的页面，或是在最长时间内不再被访问的页面，以便保证获得最低的缺页率。然而，由于人们目前无法预知进程在内存下的若干页面中哪个是未来最长时间内不再被访问的，因而该算法无法实现。

最佳置换算法可用来评价其他算法。假定系统为某进程分配了三个物理块，并考虑有页面号引用串 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1。进程运行时，先将 7, 0, 1 三个页面依次装入内存。进程要访问页面 2 时，产生缺页中断，根据最佳置换算法，选择将第 18 次访问才需调入的页面 7 淘汰。然后，访问页面 0 时，因为它已在内存中，所以不必产生缺页中断。访问页面 3 时，又会根据最佳置换算法将页面 1 淘汰……以此类推，如图 3.22 所示，从图中可以看出采用最佳置换算法时的情况。

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块 1	7	7	7	2		2		2		2		2		2				7		
物理块 2		0	0	0		0		4		0		0		0				0		
物理块 3			1	1		3		3		3		1		1				1		
缺页否	√	√	√	√		√		√		√		√		√				√		

图 3.22 利用最佳置换算法时的置换图

最长时间不被访问和以后被访问次数最小是不同的概念，初学者在理解 OPT 算法时千万不要混淆。

可以看到，发生缺页中断的次数为 9，页面置换的次数为 6。

2. 先进先出 (FIFO) 页面置换算法

优先淘汰最早进入内存的页面，即在内存中驻留时间最久的页面。该算法实现简单，只需把调入内存的页面根据先后次序链接成队列，设置一个指针总指向最早的页面。但该算法与进程实际运行时的规律不适应，因为在进程中，有的页面经常被访问。

这里仍用上面的实例采用 FIFO 算法进行页面置换。进程访问页面 2 时，把最早进入内存的页面 7 换出。然后访问页面 3 时，把 2, 0, 1 中最先进入内存的页面 0 换出。由图 3.23 可以看出，利用 FIFO 算法时进行了 12 次页面置换，比最佳置换算法正好多一倍。

FIFO 算法还会产生所分配的物理块数增大而页故障数不减反增的异常现象，这由 Belady 于

1969 年发现，因此称为 Belady 异常。只有 FIFO 算法可能出现 Belady 异常，LRU 和 OPT 算法永远不会出现 Belady 异常。

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块 1	7	7	7	2		2	2	4	4	4	0			0	0			7	7	7
物理块 2		0	0	0		3	3	3	2	2	2			1	1			1	0	0
物理块 3			1	1		1	0	0	0	3	3			3	2			2	2	1
缺页否	√	√	√	√		√	√	√	√	√	√			√	√			√	√	√

图 3.23 利用 FIFO 置换算法时的置换图

如图 3.24 所示，页面访问顺序为 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4。若采用 FIFO 置换算法，当分配的物理块为 3 个时，缺页次数为 9 次；当分配的物理块为 4 个时，缺页次数为 10 次。分配给进程的物理块增多，但缺页次数不减反增。

访问页面	3	2	1	0	3	2	4	3	2	1	0	4							
物理块 1	3	3	3	0	0	0	4					4	4						
物理块 2		2	2	2	3	3	3					1	1						
物理块 3			1	1	1	2	2					2	0						
缺页否	√	√	√	√	√	√	√	√				√	√						
物理块 1*	3	3	3	3				4	4	4	4	0	0						
物理块 2*		2	2	2				2	3	3	3	3	4						
物理块 3*			1	1				1	1	2	2	2	2						
物理块 4*				0				0	0	0	1	1	1						
缺页否	√	√	√	√				√	√	√	√	√	√						

图 3.24 Belady 异常

3. 最近最久未使用 (LRU) 置换算法

选择最近最长时间未访问过的页面予以淘汰，它认为过去一段时间内未访问过的页面，在最近的将来可能也不会被访问。该算法为每个页面设置一个访问字段，来记录页面自上次被访问以来所经历的时间，淘汰页面时选择现有页面中值最大的予以淘汰。

再对上面的实例采用 LRU 算法进行页面置换，如图 3.25 所示。进程第一次对页面 2 访问时，将最近最久未被访问的页面 7 置换出去。然后在访问页面 3 时，将最近最久未使用的页面 1 换出。

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块 1	7	7	7	2		2		4	4	4	0			1	1			1		
物理块 2		0	0	0		0		0	0	3	3			3	0		0			
物理块 3			1	1		3		3	2	2	2			2	2		7			
缺页否	√	√	√	√		√		√	√	√	√			√	√		√			

图 3.25 LRU 页面置换算法时的置换图

在图 3.25 中，前 5 次置换的情况与最佳置换算法相同，但两种算法并无必然联系。实际上，LRU 算法根据各页以前的情况，是“向前看”的，而最佳置换算法则根据各页以后的使用情况，是“向后看”的。

LRU 算法的性能较好，但需要寄存器和栈的硬件支持。LRU 是堆栈类的算法。理论上可以证明，堆栈类算法不可能出现 Belady 异常。FIFO 算法基于队列实现，不是堆栈类算法。

4. 时钟 (CLOCK) 置换算法

LRU 算法的性能接近于 OPT 算法，但实现起来比较困难，且开销大；FIFO 算法实现简单，但性能差。因此，操作系统的设计师尝试了很多算法，试图用比较小的开销接近 LRU 算法的性能，这类算法都是 CLOCK 算法的变体。因为算法要循环扫描缓冲区，像时钟的指针一样转动，所以称为 CLOCK 算法。

简单的 CLOCK 算法给每帧关联一个附加位，称为使用位。当某页首次装入主存时，将该帧的使用位设置为 1；当该页随后再被访问到时，其使用位也被置为 1。对于页替换算法，用于替换的候选帧集合可视为一个循环缓冲区，并有一个指针与之相关联。当某一页被替换时，该指针被设置成指向缓冲区中的下一帧。当需要替换一页时，操作系统扫描缓冲区，以查找使用位被置为 0 的一帧。每当遇到一个使用位为 1 的帧时，操作系统就将该位重新置为 0；若在这个过程开始时，缓冲区中所有帧的使用位均为 0，则选择遇到的第一个帧替换；若所有帧的使用位均为 1，则指针在缓冲区中完整地循环一周，把所有使用位都置为 0，并停留在最初的位置上，替换该帧中的页。由于该算法循环检查各页面的情况，因此称 CLOCK 算法，又称最近未用 (Not Recently Used, NRU) 算法。

CLOCK 算法的性能比较接近 LRU 算法，而通过增加使用的位数目，可以使得 CLOCK 算法更加高效。在使用位的基础上再增加一个修改位，则得到改进型 CLOCK 置换算法。这样，每帧都处于以下 4 种情况之一：

- 1) 最近未被访问，也未被修改 ($u = 0, m = 0$)。
- 2) 最近被访问，但未被修改 ($u = 1, m = 0$)。
- 3) 最近未被访问，但被修改 ($u = 0, m = 1$)。
- 4) 最近被访问，被修改 ($u = 1, m = 1$)。

算法执行如下操作步骤：

- 1) 从指针的当前位置开始，扫描帧缓冲区。在这次扫描过程中，对使用位不做任何修改。选择遇到的第一个帧 ($u = 0, m = 0$) 用于替换。
- 2) 若第 1) 步失败，则重新扫描，查找 ($u = 0, m = 1$) 的帧。选择遇到的第一个这样的帧用于替换。在这个扫描过程中，对每个跳过的帧，把它的使用位设置成 0。
- 3) 若第 2) 步失败，则指针将回到它的最初位置，且集合中所有帧的使用位均为 0。重复第 1) 步，并且若有必要，重复第 2) 步，以便可以找到供替换的帧。

改进型 CLOCK 算法优于简单 CLOCK 算法的地方在于替换时首选没有变化的页。由于修改过的页在被替换之前必须写回，因而这样做会节省时间。

有些读者会认为 CLOCK 算法和改进型 CLOCK 算法记忆起来不易。为方便记忆，我们将其总结如下。

操作系统中任何经过优化而有效的页面置换算法都有一个原则，即尽可能保留曾经使用过的页面，而淘汰未使用的页面，认为这样可以在总体上减少换页次数。CLOCK 算法只考虑到是否被访问过，因此被访问过的当然尽可能留下，未使用过的就淘汰；而改进型 CLOCK 算法对使用过的页面又做了细分，分为使用过但未修改过和使用过且修改过。因此，若有未使用过的页面，则当然首先把它换出，若全部页面都使用过，则当然优先把未修改过的页面换出。

为帮助读者理解，这里举一个例子。假设系统给某进程分配了 5 个页框，刚开始，进程依次

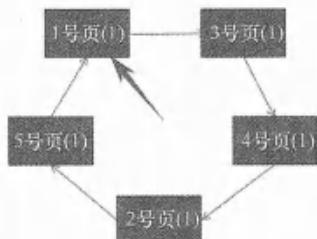


图 3.26 刚开始扫描时指针指向的页面

访问 1, 3, 4, 2, 5 号页面，系统会将这些页面连成一个循环队列，刚开始扫描指针指向第一个被访问的页面（即 1 号页），如图 3.26 所示。

图 3.26 中，小括号内的数字就是使用位。接下来，若进程请求访问 6 号页面，则由于此时分配给进程的 5 个页框都被使用，因此必须选择一个页面置换出去。按照 CLOCK 置换算法的规则，在第一轮扫描中，指针扫过的页面的使用位应置为 0。第一轮扫描的过程如图 3.27 所示。

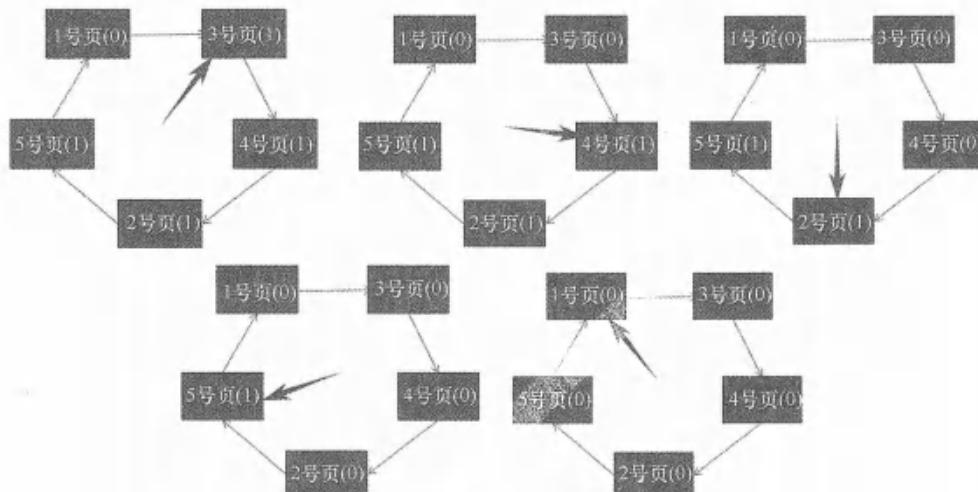
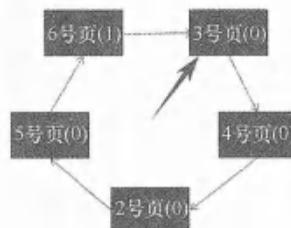


图 3.27 第一轮扫描的过程

第一轮扫描中，未找到使用位为 0 的页面，因此需要进行第二轮扫描。第二轮扫描中，1 号页面的使用位为 0，因此将 1 号页面换出，将 6 号页面换入，将 6 号页的访问位设置为 1，并将扫描指针后移（若下次需要换出页面，则从 3 号页面开始扫描），如图 3.28 所示。

注意一个小细节：假设 1 号页面原先占有的是 x 号物理块（页框），则 6 号页面换入内存后也放在 x 号物理块中。

图 3.28 第二轮扫描后指针指向的页面



3.2.4 页面分配策略

1. 驻留集大小

对于分页式的虚拟内存，在进程准备执行时，不需要也不可能把一个进程的所有页都读入主存。因此，操作系统必须决定读取多少页，即决定给特定的进程分配几个页框。给一个进程分配的物理页框的集合就是这个进程的驻留集。需要考虑以下几点：

- 1) 分配给一个进程的存储量越小，任何时候驻留在主存中的进程数就越多，从而可以提高处理机的时间利用效率。
- 2) 若一个进程在主存中的页数过少，则尽管有局部性原理，页错误率仍然会相对较高。
- 3) 若页数过多，则由于局部性原理，给特定的进程分配更多的主存空间对该进程的错误率没有明显的影响。

基于这些因素，现代操作系统通常采用三种策略：

- 1) 固定分配局部置换。它为每个进程分配一定数目的物理块，在整个运行期间都不改变。若进程在运行中发生缺页，则只能从该进程在内存中的页面中选出一页换出，然后调入需要的页面。实现这种策略时，难以确定应为每个进程分配的物理块数目：太少会频繁出现缺页中断，太多又会使 CPU 和其他资源利用率下降。
- 2) 可变分配全局置换。这是最易于实现的物理块分配和置换策略，它为系统中的每个进程分配一定数目的物理块，操作系统自身也保持一个空闲物理块队列。当某进程发生缺页时，系统从空闲物理块队列中取出一个物理块分配给该进程，并将欲调入的页装入其中。这种方法比固定分配局部置换更加灵活，可以动态增加进程的物理块，但也存在弊端，如它会盲目地给进程增加物理块，从而导致系统多道程序的并发能力下降。
- 3) 可变分配局部置换。它为每个进程分配一定数目的物理块，当某个进程发生缺页时，只允许从该进程在内存的页面中选出一页换出，因此不会影响其他进程的运行。若进程在运行中频繁地缺页，则系统再为该进程分配若干物理块，直至该进程缺页率趋于适当程度；反之，若进程运行中的缺页率特别低，则可适当减少分配给该进程的物理块。比起可变分配全局置换，这种方法不仅可以动态增加进程物理块的数量，还能动态减少进程物理块的数量，在保证进程不会过多地调页的同时，也保持了系统的多道程序并发能力。当然它需要更复杂的实现，也需要更大的开销，但对比频繁地换入/换出所浪费的计算机资源，这种牺牲是值得的。

页面分配策略在 2015 年的统考选择题中出现过，考查的是这三种策略的名称。往年很多读者看到这里时，由于认为不是重点，复习时便一带而过，最后在考试中失分。在这种基础题上失分是十分可惜的。再次提醒读者，考研成功的秘诀在于“反复多次”和“全面”。

2. 调入页面的时机

为确定系统将进程运行时所缺的页面调入内存的时机，可采取以下两种调页策略：

- 1) 预调页策略。根据局部性原理，一次调入若干相邻的页可能会比一次调入一页更高效。但若调入的一批页面中大多数都未被访问，则又是低效的。因此，需要采用以预测为基础的预调页策略，将预计在不久之后会被访问的页面预先调入内存。但目前预调页的成功率仅约 50%。因此这种策略主要用于进程的首次调入，由程序员指出应先调入哪些页。
- 2) 请求调页策略。进程在运行中需要访问的页面不在内存而提出请求，由系统将所需页面调入内存。由这种策略调入的页一定会被访问，且这种策略比较易于实现，因此在目前的虚拟存储器中大多采用此策略。它的缺点是每次只调入一页，调入/调出页面数多时会花费过多的 I/O 开销。

预调入实际上就是运行前的调入，请求调页实际上就是运行期间调入。一般情况下，两种调页策略会同时使用。

3. 从何处调入页面

请求分页系统中的外存分为两部分：用于存放文件的文件区和用于存放对换页面的对换区。对换区通常采用连续分配方式，而文件区采用离散分配方式，因此对换区的磁盘 I/O 速度比文件区的更快。这样，从何处调入页面就存在三种情况：

- 1) 系统拥有足够的对换区空间。可以全部从对换区调入所需页面，以提高调页速度。为此，在进程运行前，需将与该进程有关的文件从文件区复制到对换区。
- 2) 系统缺少足够的对换区空间。凡不会被修改的文件都直接从文件区调入；而当换出这些

页面时，由于它们未被修改而不必再将它们换出。但对于那些可能被修改的部分，在将它们换出时须调到对换区，以后需要时再从对换区调入（因为读的速度比写的速度快）。

- 3) UNIX 方式。与进程有关的文件都放在文件区，因此未运行过的页面都应从文件区调入。曾经运行过但又被换出的页面，由于放在对换区，因此下次调入时应从对换区调入。进程请求的共享页面若被其他进程调入内存，则无须再从对换区调入。

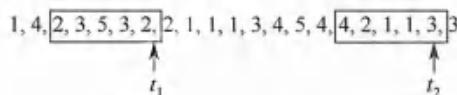
3.2.5 抖动

在页面置换过程中，一种最糟糕的情形是，刚刚换出的页面马上又要换入主存，刚刚换入的页面马上又要换出主存，这种频繁的页面调度行为称为抖动或颠簸。若一个进程在换页上用的时间多于执行时间，则这个进程就在颠簸。

频繁发生缺页中断（抖动）的主要原因是，某个进程频繁访问的页面数目高于可用的物理页帧数目。虚拟内存技术可在内存中保留更多的进程以提高系统效率。在稳定状态，几乎主存的所有空间都被进程块占据，处理机和操作系统可以直接访问到尽可能多的进程。然而，如果管理不当，那么处理机的大部分时间都将用于交换块，即请求调入页面的操作，而不是执行进程的指令，因此会大大降低系统效率。

3.2.6 工作集

工作集是指在某段时间间隔内，进程要访问的页面集合。基于局部性原理，可以用最近访问过的页面来确定工作集。一般来说，工作集 W 可由时间 t 和工作集窗口大小 Δ 来确定。例如，某进程对页面的访问次序如下：



假设系统为该进程设定的工作集窗口大小 Δ 为 5，则在 t_1 时刻，进程的工作集为 {2, 3, 5}；在 t_2 时刻，进程的工作集为 {1, 2, 3, 4}。

实际应用中，工作集窗口会设置得很大，即对于局部性好的程序，工作集大小一般会比工作集窗口 Δ 小很多。工作集反映了进程在接下来的一段时间内很有可能会频繁访问的页面集合，因此，若分配给进程的物理块小于工作集大小，则该进程就很有可能频繁缺页，所以为了防止这种抖动现象，一般来说分配给进程的物理块数（即驻留集大小）要大于工作集大小。

工作集模型的原理是，让操作系统跟踪每个进程的工作集，并为进程分配大于其工作集的物理块。落在工作集内的页面需要调入驻留集中，而落在工作集外的页面可从驻留集中换出。若还有空闲物理块，则可以再调一个进程到内存以增加多道程序数。若所有进程的工作集之和超过了可用物理块的总数，则操作系统会暂停一个进程，将其页面调出并将其物理块分配给其他进程，防止出现抖动现象。

3.2.7 地址翻译

本小节引入一个实例来说明虚实地址的变换过程，考虑到统考试题近来出现了学科综合的趋势，这里结合“计算机组成原理”中的 Cache 部分进行讲解。对于不参加统考的读者，可以看到翻译出实地址为止，对于参加统考却还没有复习计算机组成原理的读者，可在复习完“计算机组成原理”后，再回来看本章的内容。

设某系统满足以下条件：

- 有一个 TLB 与一个 data Cache
 - 存储器以字节为编址单位
 - 虚拟地址 14 位
 - 物理地址 12 位
 - 页面大小为 64B
 - TLB 为四路组相联，共有 16 个条目
 - data Cache 是物理寻址、直接映射的，行大小为 4B，共有 16 组
- 写出访问地址为 0x03d4, 0x00f1 和 0x0229 的过程。

因为本系统以字节编址，页面大小为 64B，则页内偏移地位为 $\log_2(64B/1B) = 6$ 位，所以虚拟页号为 $14 - 6 = 8$ 位，物理页号为 $12 - 6 = 6$ 位。因为 TLB 为四路组相联，共有 16 个条目，则 TLB 共有 $16/4 = 4$ 组，因此虚拟页号中低 $\log_2 4 = 2$ 位就为组索引，高 6 位就为 TLB 标记。又因为 Cache 行大小为 4B，因此物理地址中低 $\log_2 4 = 2$ 位为块偏移，Cache 共有 16 组，可知接下来 $\log_2 16 = 4$ 位为组索引，剩下高 6 位作为标记。地址结构如图 3.29 所示。

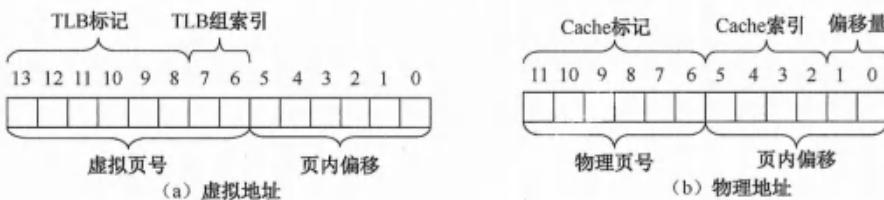


图 3.29 地址结构

TLB、页表、data Cache 内容如表 3.2、表 3.3 及表 3.4 所示。

表 3.2 TLB

位	标记	物理页号	有效位	标记位	物理页号	有效位
0	03	-	0	09	0D	1
	00	-	0	07	02	1
1	03	2D	1	02	-	0
	04	-	0	0A	-	0
2	02	-	0	08	-	0
	06	-	0	03	-	0
3	07	-	0	03	0D	1
	0A	34	1	02	-	0

表 3.3 部分页表

虚拟页号	物理页号	有效位	虚拟页号	物理页号	有效位
00	28	1	08	-	0
01	-	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	-	0
04	-	0	0C	-	0
05	16	1	0D	2D	1
06	-	0	0E	11	1
07	-	0	0F	0D	1

表 3.4 data Cache 内容

索引	标记位	有效位	块 0	块 1	块 2	块 3
0	19	1	99	11	23	11
1	15	0	-	-	-	-
2	1B	1	00	02	04	08
3	36	0	-	-	-	-
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D

(续表)

索引	标记位	有效位	块 0	块 1	块 2	块 3
6	31	0	-	-	-	-
7	16	1	11	C2	DF	03
8	24	1	3A	00	51	89
9	2D	0	-	-	-	-
A	2D	1	93	15	DA	3B
B	0B	0	-	-	-	-
C	02	0	-	-	-	-
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	-	-	-	-

先把十六进制的虚拟地址 0x03d4, 0x00f1 和 0x0229 转化为二进制形式，如表 3.5 所示。

表 3.5 虚拟地址结构

虚拟地址	TLB 标记								组索引							
	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0x03d4	0	0	0	0	1	1	1	1	0	1	0	1	0	0	0	0
0x00f1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	
0x0229	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	
	虚拟页号								页内偏移							

得到每个地址的组索引和 TLB 标记，接下来就要找出每个地址的页面在不在主存中，若在主存中，则还要找出物理地址。

对于 0x03d4，组索引为 3，TLB 标记为 0x03，查 TLB，第 3 组中正好有标记为 03 的项，有效位为 1，可知页面在主存中，对应的物理页号为 0d (001101)，再拼接页内地址 010100，可得物理地址为 0x354 (001101010100)。

对于 0x00f1，组索引为 3，TLB 标记为 0x00，查 TLB，第 3 组中没有标记为 00 的项，再去找页表，虚拟页号为 0x03，页表第 3 行的有效位为 1，可知页面在主存中，物理页号为 02(0000010)，再拼接页内地址 110001，可得物理地址为 0xb1 (000010110001)。

对于 0x0229，组索引为 0，TLB 标记为 0x02，查 TLB，第 0 组中没有标记为 02 的项，再去找页表，虚拟页号为 0x08，页表第 8 行的有效位为 0，页面不在主存中，产生缺页中断。

找出在主存中的页面的物理地址后，就要通过物理地址访问数据，接下来要找该物理地址的内容在不在 Cache 中，物理地址结构如表 3.6 所示。

表 3.6 物理地址结构

物理地址	Cache 标记								Cache 索引				偏移	
	11	10	9	8	7	6	5	4	3	2	1	0		
0x354	0	0	1	1	0	1	0	1	0	1	0	0	0	0
0xb1	0	0	0	0	1	0	1	1	0	0	0	0	1	
	物理页号								页内偏移					

对于 0x354, Cache 索引为 5, Cache 标记为 0x0d, 对照 Cache 中索引为 5 的行, 标记正好为 0d, 有效位为 1, 可知该块在 Cache 中, 偏移 0, 即块 0, 可得虚拟地址 0x03d4 的内容为 36H。

对于 0x0b1, Cache 索引为 c, Cache 标记为 0x02, 对照 Cache 中索引为 c 的行, 有效位为 0, 可知该块不在 Cache 中, 要去主存中查找物理页号为 2、偏移为 0x31 的内容。

以上例子基本覆盖了从虚拟地址到 Cache 查找内容的所有可能出现的情况, 读者务必要掌握此节的内容, 查找顺序是从 TLB 到页表 (TLB 不命中), 再到 Cache 和主存, 最后到外存。

3.2.8 本节小结

本节开头提出的问题的参考答案如下。

1) 为什么要引入虚拟内存?

上一节提到过, 多道程序并发执行不仅使进程之间共享了处理器, 而且同时共享了主存。然而, 随着对处理器需求的增长, 进程的执行速度会以某种合理平滑的方式慢下来。但是, 若同时运行的进程太多, 则需要很多的内存, 当一个程序没有内存空间可用时, 那么它甚至无法运行。所以, 在物理上扩展内存相对有限的条件下, 应尝试以一些其他可行的方式在逻辑上扩充内存。

2) 虚拟内存 (虚存) 空间的大小由什么因素决定?

虚存的容量要满足以下两个条件:

- ① 虚存的实际容量 \leq 内存容量和外存容量之和, 这是硬件的硬性条件规定的, 若虚存的实际容量超过了这个容量, 则没有相应的空间来供虚存使用。
- ② 虚存的最大容量 \leq 计算机的地址位数能容纳的最大容量。假设地址是 32 位的, 按字节编址, 一个地址代表 1B 存储空间, 则虚存的最大容量 $\leq 4\text{GB}$ (2^{32}B)。这是因为若虚存的最大容量超过 4GB, 则 32 位的地址将无法访问全部虚存, 也就是说 4GB 以后的空间被浪费了, 相当于没有一样, 没有任何意义。

实际虚存的容量是取条件①和②的交集, 即两个条件都要满足, 仅满足一个条件是不行的。

3) 虚拟内存是怎么解决问题的? 会带来什么问题?

虚拟内存使用外存上的空间来扩充内存空间, 通过一定的换入/换出, 使得整个系统在逻辑上能够使用一个远远超出其物理内存大小的内存容量。因为虚拟内存技术调换页面时需要访问外存, 会导致平均访存时间增加, 若使用了不合适的替换算法, 则会大大降低系统性能。

本节学习了 4 种页面置换算法, 要把它们与处理机调度算法区分开。当然, 这些调度算法之间也是有联系的, 它们都有一个共同点, 即通过一定的准则决定资源的分配对象。在处理机调度算法中这些准则比较多, 有优先级、响应比、时间片等, 而在页面调度算法中就比较简单, 即是否被用到过或近段时间内是否经常使用。在操作系统中, 几乎每类资源都会有相关的调度算法, 读者通过将这些调度算法作为线索, 可把整个操作系统的课程连成一个整体。

3.2.9 本节习题精选

一、单项选择题

1. 【2012 统考真题】下列关于虚拟存储器的叙述中, 正确的是 ()。
 - A. 虚拟存储只能基于连续分配技术
 - B. 虚拟存储只能基于非连续分配技术
 - C. 虚拟存储容量只受外存容量的限制
 - D. 虚拟存储容量只受内存容量的限制
2. 请求分页存储管理中, 若把页面尺寸增大一倍而且可容纳的最大页数不变, 则在程序顺序执行时缺页中断次数会 ()。
 - A. 增加
 - B. 减少

- C. 不变 D. 可能增加也可能减少

3. 进程在执行中发生了缺页中断，经操作系统处理后，应让其执行（ ）指令。
A. 被中断的前一条 B. 被中断的那一条
C. 被中断的后一条 D. 启动时的第一条

4. 【2011 统考真题】在缺页处理过程中，操作系统执行的操作可能是（ ）。
I. 修改页表 II. 磁盘 I/O III. 分配页框
A. 仅 I、II B. 仅 II C. 仅 III D. I、II 和 III

5. 【2013 统考真题】若用户进程访问内存时产生缺页，则下列选项中，操作系统可能执行的操作是（ ）。
I. 处理越界错 II. 置换页 III. 分配内存
A. 仅 I、II B. 仅 II、III C. 仅 I、III D. I、II 和 III

6. 虚拟存储技术是（ ）。
A. 补充内存物理空间的技术 B. 补充内存逻辑空间的技术
C. 补充外存空间的技术 D. 扩充输入/输出缓冲区的技术

7. 以下不属于虚拟内存特征的是（ ）。
A. 一次性 B. 多次性 C. 对换性 D. 离散性

8. 为使虚存系统有效地发挥其预期的作用，所运行的程序应具有的特性是（ ）。
A. 该程序不应含有过多的 I/O 操作
B. 该程序的大小不应超过实际的内存容量
C. 该程序应具有较好的局部性
D. 该程序的指令相关性不应过多

9. （ ）是请求分页存储管理方式和基本分页存储管理方式的区别。
A. 地址重定向 B. 不必将作业全部装入内存
C. 采用快表技术 D. 不必将作业装入连续区域

10. 下面关于请求页式系统的页面调度算法中，说法错误的是（ ）。
A. 一个好的页面调度算法应减少和避免抖动现象
B. FIFO 算法实现简单，选择最先进入主存储器的页面调出
C. LRU 算法基于局部性原理，首先调出最近一段时间内最长时间未被访问过的页面
D. CLOCK 算法首先调出一段时间内被访问次数多的页面

11. 考虑页面置换算法，系统有 m 个物理块供调度，初始时全空，页面引用串长度为 p ，包含了 n 个不同的页号，无论用什么算法，缺页次数不会少于（ ）。
A. m B. p C. n D. $\min(m, n)$

12. 在请求分页存储管理中，若采用 FIFO 页面淘汰算法，则当可供分配的页帧数增加时，缺页中断的次数（ ）。
A. 减少 B. 增加
C. 无影响 D. 可能增加也可能减少

13. 设主存容量为 1MB，外存容量为 400MB，计算机系统的地址寄存器有 32 位，那么虚拟存储器的最大容量是（ ）。
A. 1MB B. 401MB C. $1MB + 2^{32}MB$ D. $2^{32}B$

14. 虚拟存储器的最大容量（ ）。
A. 为内外存容量之和 B. 由计算机的地址结构决定

- C. 是任意的 D. 由作业的地址空间决定
15. 某虚拟存储器系统采用页式内存管理，使用 LRU 页面替换算法，考虑页面访问地址序列为 1 8 1 7 8 2 7 2 1 8 3 8 2 1 3 1 7 1 3 7。假定内存容量为 4 个页面，开始时是空的，则页面失效次数是（ ）。
- A. 4 B. 5 C. 6 D. 7
16. 导致 LRU 算法实现起来耗费高的原因是（ ）。
- A. 需要硬件的特殊支持 B. 需要特殊的中断处理程序
C. 需要在页表中标明特殊的页类型 D. 需要对所有的页进行排序
17. 在虚拟存储器系统的页表项中，决定是否会发生页故障的是（ ）。
- A. 合法位 B. 修改位 C. 页类型 D. 保护码
18. 在页面置换策略中，（ ）策略可能引起抖动。
- A. FIFO B. LRU C. 没有一种 D. 所有
19. 虚拟存储管理系统的基础是程序的（ ）理论。
- A. 动态性 B. 虚拟性 C. 局部性 D. 全局性
20. 使用（ ）方法可以实现虚拟存储。
- A. 分区合并 B. 覆盖、交换 C. 快表 D. 段合并
21. 请求分页存储管理的主要特点是（ ）。
- A. 消除了页内零头 B. 扩充了内存
C. 便于动态链接 D. 便于信息共享
22. 在请求分页存储管理的页表中增加了若干项信息，其中修改位和访问位供（ ）参考。
- A. 分配页面 B. 调入页面 C. 置换算法 D. 程序访问
23. 产生内存抖动的主要原因是（ ）。
- A. 内存空间太小 B. CPU 运行速度太慢
C. CPU 调度算法不合理 D. 页面置换算法不合理
24. 在页面置换算法中，存在 Belady 现象的算法是（ ）。
- A. 最佳页面置换算法 (OPT) B. 先进先出置换算法 (FIFO)
C. 最近最久未使用算法 (LRU) D. 最近未使用算法 (NRU)
25. 页式虚拟存储管理的主要特点是（ ）。
- A. 不要求将作业装入主存的连续区域
B. 不要求将作业同时全部装入主存的连续区域
C. 不要求进行缺页中断处理
D. 不要求进行页面置换
26. 提供虚拟存储技术的存储管理方法有（ ）。
- A. 动态分区存储管理 B. 页式存储管理
C. 请求段式存储管理 D. 存储覆盖技术
27. 在计算机系统中，快表用于（ ）。
- A. 存储文件信息 B. 与主存交换信息
C. 地址变换 D. 存储通道程序
28. 在虚拟分页存储管理系统中，若进程访问的页面不在主存中，且主存中没有可用的空闲帧时，系统正确的处理顺序为（ ）。
- A. 决定淘汰页 → 页面调出 → 缺页中断 → 页面调入

- B. 决定淘汰页→页面调入→缺页中断→页面调出
 C. 缺页中断→决定淘汰页→页面调出→页面调入
 D. 缺页中断→决定淘汰页→页面调入→页面调出
29. 已知系统为 32 位实地址，采用 48 位虚拟地址，页面大小为 4KB，页表项大小为 8B。假设系统使用纯页式存储，则要采用（ ）级页表，页内偏移（ ）位。
 A. 3, 12 B. 3, 14 C. 4, 12 D. 4, 14
30. 下列说法中，正确的是（ ）。
 I. 先进先出 (FIFO) 页面置换算法会产生 Belady 现象
 II. 最近最少使用 (LRU) 页面置换算法会产生 Belady 现象
 III. 在进程运行时，若其工作集页面都在虚拟存储器内，则能够使该进程有效地运行，否则会出现频繁的页面调入/调出现象
 IV. 在进程运行时，若其工作集页面都在主存储器内，则能够使该进程有效地运行，否则会出现频繁的页面调入/调出现象
 A. I、III B. I、IV C. II、III D. II、IV
31. 测得某个采用按需调页策略的计算机系统的部分状态数据为：CPU 利用率为 20%，用于交换空间的磁盘利用率为 97.7%，其他设备的利用率为 5%。由此判断系统出现异常，这种情况下（ ）能提高系统性能。
 A. 安装一个更快的硬盘 B. 通过扩大硬盘容量增加交换空间
 C. 增加运行进程数 D. 加内存条来增加物理空间容量
32. 假定有一个请求分页存储管理系统，测得系统各相关设备的利用率为：CPU 的利用率为 10%，磁盘交换区的利用率为 99.7%，其他 I/O 设备的利用率为 5%。下面（ ）措施将可能改进 CPU 的利用率。
 I. 增大内存的容量 II. 增大磁盘交换区的容量
 III. 减少多道程序的度数 IV. 增加多道程序的度数
 V. 使用更快速的磁盘交换区 VI. 使用更快速的 CPU
 A. I、II、III、IV B. I、III C. II、III、V D. II、VI
33. 【2011 统考真题】当系统发生抖动时，可以采取的有效措施是（ ）。
 I. 撤销部分进程 II. 增加磁盘交换区的容量
 III. 提高用户进程的优先级
 A. 仅 I B. 仅 II C. 仅 III D. 仅 I、II
34. 【2014 统考真题】下列措施中，能加快虚实地址转换的是（ ）。
 I. 增大快表 (TLB) 容量 II. 让页表常驻内存
 III. 增大交换区 (swap)
 A. 仅 I B. 仅 II C. 仅 I、II D. 仅 II、III
35. 【2014 统考真题】在页式虚拟存储管理系统中，采用某些页面置换算法会出现 Belady 异常现象，即进程的缺页次数会随着分配给该进程的页框个数的增加而增加。下列算法中，可能出现 Belady 异常现象的是（ ）。
 I. LRU 算法 II. FIFO 算法 III. OPT 算法
 A. 仅 II B. 仅 I、II C. 仅 I、III D. 仅 II、III
36. 【2016 统考真题】某系统采用改进型 CLOCK 置换算法，页表项中字段 A 为访问位，M 为修改位。A = 0 表示页最近没有被访问，A = 1 表示页最近被访问过。M = 0 表示页未

被修改过, $M=1$ 表示页被修改过。按 (A, M) 所有可能的取值, 将页分为 $(0, 0), (1, 0), (0, 1)$ 和 $(1, 1)$ 四类, 则该算法淘汰页的次序为 ()。

- A. $(0, 0), (0, 1), (1, 0), (1, 1)$
 B. $(0, 0), (1, 0), (0, 1), (1, 1)$
 C. $(0, 0), (0, 1), (1, 1), (1, 0)$
 D. $(0, 0), (1, 1), (0, 1), (1, 0)$

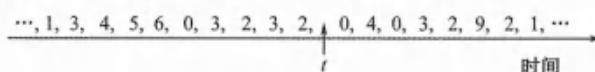
37. 【2015 统考真题】在请求分页系统中, 页面分配策略与页面置换策略不能组合使用的是 ()。

- A. 可变分配, 全局置换
 B. 可变分配, 局部置换
 C. 固定分配, 全局置换
 D. 固定分配, 局部置换

38. 【2015 统考真题】系统为某进程分配了 4 个页框, 该进程已访问的页号序列为 $2, 0, 2, 9, 3, 4, 2, 8, 2, 4, 8, 4, 5$ 。若进程要访问的下一页的页号为 7, 依据 LRU 算法, 应淘汰页的页号是 ()。

- A. 2 B. 3 C. 4 D. 8

39. 【2016 统考真题】某进程访问页面的序列如下所示。



若工作集的窗口大小为 6, 则在 t 时刻的工作集为 ()。

- A. $\{6, 0, 3, 2\}$
 B. $\{2, 3, 0, 4\}$
 C. $\{0, 4, 3, 2, 9\}$
 D. $\{4, 5, 6, 0, 3, 2\}$

40. 【2019 统考真题】某系统采用 LRU 页面置换算法和局部置换策略, 若系统为进程 P 预分配了 4 个页框, 进程 P 访问页号的序列为 $0, 1, 2, 7, 0, 5, 3, 5, 0, 2, 7, 6$, 则进程访问上述页的过程中, 产生页面置换的总次数是 ()。

- A. 3 B. 4 C. 5 D. 6

41. 【2020 统考真题】下列因素中, 影响请求分页系统有效(平均)访存时间的是 ()。

- I. 缺页率 II. 磁盘读写时间 III. 内存访问时间
 IV. 执行缺页处理程序的 CPU 时间
 A. 仅 II、III B. 仅 I、IV C. 仅 I、III、IV D. I、II、III 和 IV

二、综合应用题

- 覆盖技术与虚拟存储技术有何本质上的不同? 交换技术与虚拟存储技术中使用的调入/调出技术有何相同与不同之处?
- 假定某操作系统存储器采用页式存储管理, 一个进程在相联存储器中的页表项见表 A, 不在相联存储器的页表项见表 B。

表 A 相联存储器中的页表

页号	页帧号
0	f1
1	f2
2	f3
3	f4

表 B 内存中的页表

页号	页帧号
4	f5
5	f6
6	f7
7	f8
8	f9
9	f10

注: 只列出不在相联存储器中的页表项。

假定该进程长度为 320B，每页 32B。现有逻辑地址（八进制）为 101, 204, 576，若上述逻辑地址能转换成物理地址，说明转换的过程，并指出具体的物理地址；若不能转换，说明其原因。

3. 某分页式虚拟存储系统，用于页面交换的磁盘的平均访问及传输时间是 20ms。页表保存在主存中，访问时间为 1μs，即每引用一次指令或数据，需要访问内存两次。为改善性能，可以增设一个关联寄存器，若页表项在关联寄存器中，则只需访问一次内存。假设 80% 的访问的页表项在关联寄存器中，剩下的 20% 中，10% 的访问（即总数的 2%）会产生缺页。请计算有效访问时间。
4. 在页式虚存管理系统中，假定驻留集为 m 个页帧（初始所有页帧均为空），在长为 p 的引用串中具有 n 个不同页号 ($n > m$)，对于 FIFO、LRU 两种页面置换算法，试给出页故障数的上限和下限，说明理由并举例说明。
5. 【2009 统考真题】请求分页管理系统中，假设某进程的页表内容如下表所示。

页面大小为 4KB，一次内存的访问时间是 100ns，一次快表(TLB)的访问时间是 10ns，处理一次缺页的平均时间为 10^8 ns (已含更新 TLB 和页表的时间)，进程的驻留集大小固定为 2，采用最近最少使用(LRU)置换算法和局部淘汰策略。假设：① TLB 初始为空；② 地址转换时先访问 TLB，若 TLB 未命中，再访问页表（忽略访问页表后的 TLB 更新时间）；③ 有效位为 0 表示页面不在内存，产生缺页中断，缺页中断处理后，返回到产生缺页中断的指令处重新执行。设有虚地址访问序列 2362H, 1565H, 25A5H，请问：

- 1) 依次访问上述三个虚拟地址，各需多少时间？给出计算过程。
- 2) 基于上述访问序列，虚地址 1565H 的物理地址是多少？请说明理由。
6. 在一个请求分页存储管理系统中，一个作业的页面走向为 4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5，当分配给作业的物理块数分别为 3 和 4 时，试计算采用下述页面淘汰算法时的缺页率（假设开始执行时主存中没有页面），并比较结果。
 - 1) 最佳置换算法。
 - 2) 先进先出置换算法。
 - 3) 最近最久未使用算法。
7. 一个页式虚拟存储系统，其并发进程数固定为 4 个。最近测试了它的 CPU 利用率和用于页面交换的磁盘的利用率，得到的结果就是下列 3 组数据中的一组。针对每组数据，说明系统发生了什么事情。增加并发进程数能提升 CPU 的利用率吗？页式虚拟存储系统有用吗？
 - 1) CPU 利用率为 13%；磁盘利用率为 97%。
 - 2) CPU 利用率为 87%；磁盘利用率为 3%。
 - 3) CPU 利用率为 13%；磁盘利用率为 3%。
8. 现有一请求页式系统，页表保存在寄存器中。若有一个可用的空页或被置换的页未被修改，则它处理一个缺页中断需要 8ms；若被置换的页已被修改，则处理一缺页中断因增加写回外存时间而需要 20ms，内存的存取时间为 1μs。假定 70% 被置换的页被修改过，为保证有效存取时间不超过 2μs，可接受的最大缺页中断率是多少？
9. 已知系统为 32 位实地址，采用 48 位虚拟地址，页面大小为 4KB，页表项大小为 8B，每

页号	页框 (Page Frame) 号	有效位 (存在位)
0	101H	1
1	—	0
2	254H	1

段最大为 4GB。

- 1) 假设系统使用纯页式存储，则要采用多少级页表？页内偏移多少位？
- 2) 假设系统采用一级页表，TLB 命中率为 98%，TLB 访问时间为 10ns，内存访问时间为 100ns，并假设当 TLB 访问失败时才开始访问内存，问平均页面访问时间是多少？
- 3) 若是二级页表，页面平均访问时间是多少？
- 4) 上题中，若要满足访问时间小于 120ns，则命中率至少需要为多少？
- 5) 若系统采用段页式存储，则每用户最多可以有多少个段？段内采用几级页表？
10. 在一个请求分页系统中，采用 LRU 页面置换算法时，假如一个作业的页面走向为 1, 3, 2, 1, 1, 3, 5, 1, 3, 2, 1, 5，当分配给该作业的物理块数分别为 3 和 4 时，试计算在访问过程中发生的缺页次数和缺页率。
11. 一进程已分配到 4 个页帧，见下表（编号为十进制，从 0 开始）。当进程访问第 4 页时，产生缺页中断，请分别用 FIFO（先进先出）、LRU（最近最少使用）、改进型 CLOCK 算法，决定缺页中断服务程序选择换出的页面。

虚拟页号	页帧	装入时间	最近访问时间	访问位	修改位
2	0	60	161	0	1
1	1	130	160	0	0
0	2	26	162	1	0
3	3	20	163	1	1

12. 在页式虚拟管理的页面替换算法中，对于任何给定的驻留集大小，在什么样的访问情况下，FIFO 与 LRU 替换算法一样（即被替换的页面和缺页情况完全一样）？
13. 【2012 统考真题】某请求分页系统的页面置换策略如下：从 0 时刻开始扫描，每隔 5 个时间单位扫描一轮驻留集（扫描时间忽略不计）且本轮未被访问过的页框将被系统回收，并放入空闲页框链尾，其中内容在下一次分配之前不清空。当发生缺页时，若该页曾被使用过且还在空闲页链表中，则重新放回进程的驻留集中；否则，从空闲页框链表头部取出一个页框。
忽略其他进程的影响和系统开销。初始时进程驻留集为空。目前系统空闲页的页框号依次为 32, 15, 21, 41。进程 P 依次访问的<虚拟页号, 访问时刻>为<1, 1>, <3, 2>, <0, 4>, <0, 6>, <1, 11>, <0, 13>, <2, 14>。请回答下列问题：
 - 1) 当虚拟页为<0, 4>时，对应的页框号是什么？
 - 2) 当虚拟页为<1, 11>时，对应的页框号是什么？说明理由。
 - 3) 当虚拟页为<2, 14>时，对应的页框号是什么？说明理由。
 - 4) 这种方法是否适合于时间局部性好的程序？说明理由。
14. 某系统有 4 个页框，某个进程的页面使用情况见下表，问采用 FIFO、LRU、简单 CLOCK 和改进型 CLOCK 置换算法，将会替换哪一页？

页号	装入时间	上次引用时间	R	M
0	126	279	0	0
1	230	260	1	0
2	120	272	1	1
3	160	280	1	1

- 其中， R 是读标志位， M 是修改标志位。
15. 有一个矩阵 $\text{int } A[100, 100]$ 以行优先方式进行存储。计算机采用虚拟存储系统，物理内存共有三页，其中一页用来存放程序，其余两页用于存放数据。假设程序已在内存中占一页，其余两页空闲。若每页可存放 200 个整数，程序 1、程序 2 执行的过程中各会发生多少次缺页？每页只能存放 100 个整数时，会发生多少次缺页？以上结果说明了什么问题？

程序 1：

```
for(i=0; i<100; i++)
    for(j=0; j<100; j++)
        A[i,j]=0;
```

程序 2：

```
for(j=0; j<100; j++)
    for(i=0; i<100; i++)
        A[i,j]=0;
```

16. 【2010 统考真题】设某计算机的逻辑地址空间和物理地址空间均为 64KB，按字节编址。若某进程最多需要 6 页（Page）数据存储空间，页的大小为 1KB，操作系统采用固定分配局部置换策略为此进程分配 4 个页框（Page Frame），见下表。在装入时刻 260 前，该进程的访问情况也见下表（访问位即使用位）。

页号	页框号	装入时刻	访问位
0	7	130	1
1	4	230	1
2	2	200	1
3	9	160	1

当该进程执行到时刻 260 时，要访问逻辑地址为 17CAH 的数据。回答下列问题：

- 1) 该逻辑地址对应的页号是多少？
- 2) 若采用先进先出（FIFO）置换算法，则该逻辑地址对应的物理地址是多少？要求给出计算过程。若采用时钟（Clock）置换算法，则该逻辑地址对应的物理地址是多少？要求给出计算过程。设搜索下一页的指针沿顺时针方向移动，且当前指向 2 号页框，如下图所示。



17. 【2017 统考真题】假定 2017 年题 44^①给出的计算机 M 采用二级分页虚拟存储管理方式，

① 本题是 2017 年统考真题，关联的题干信息太多，请在王道论坛下载 2017 年电子版真题。

虚拟地址格式如下：

页目录号（10位）	页表索引（10位）	页内偏移量（12位）
-----------	-----------	------------

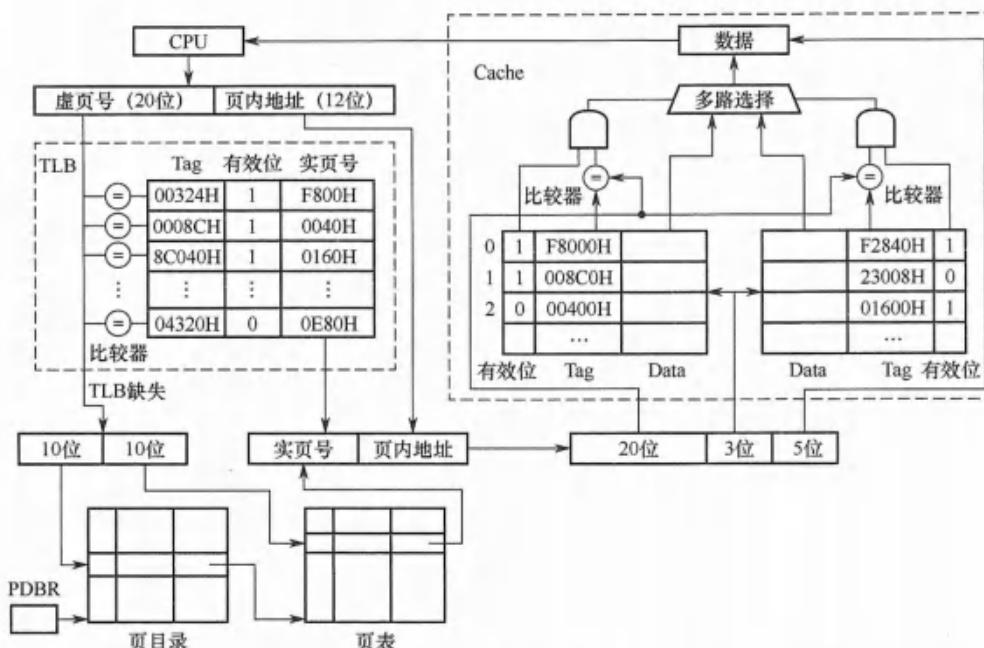
请针对 2017 年题 43 的函数 f1 和题 44 中的机器指令代码，回答下列问题。

- 1) 函数 f1 的机器指令代码占多少页？
 - 2) 取第一条指令 (push ebp) 时，若在进行地址变换的过程中需要访问内存中的页目录和页表，则会分别访问它们各自的第几个表项（编号从 0 开始）？
 - 3) M 的 I/O 采用中断控制方式。若进程 P 在调用 f1 前通过 scanf() 获取 n 的值，则在执行 scanf() 的过程中，进程 P 的状态会如何变化？CPU 是否会进入内核态？
18. Gribble 公司正在开发一款 64 位的计算机体系结构，也就是说，在访问内存时，最多可以使用 64 位的地址。假设采用的是虚拟页式存储管理，现在要为这款机器设计相应的地址映射机制。
- 1) 假设页面的大小是 4KB，每个页表项的长度是 4B，而且必须采用三级页表结构，每级页表结构中的每个页表都必须正好存放在一个物理页面中，请问在这种情形下，如何实现地址的映射？具体来说，对于给定的一个虚拟地址，应该把它划分为几部分，每部分的长度分别是多少，功能是什么？另外，采用这种地址映射机制后，可以访问的虚拟地址空间有多大？（提示：64 位地址并不一定全部用上。）
 - 2) 假设每个页表项的长度变成了 8B，而且必须采用四级页表结构，每级页表结构中的页表都必须正好存放在一个物理页面中，请问在这种情形下，系统能够支持的最大页面大小是多少？此时，虚拟地址应该如何划分？
19. 【2015 统考真题】某计算机系统按字节编址，采用二级页表的分页存储管理方式，虚拟地址格式如下所示：

10 位	10 位	12 位
页目录号	页表索引	页内偏移量

请回答下列问题：

- 1) 页和页框的大小各为多少字节？进程的虚拟地址空间大小为多少页？
 - 2) 若页目录项和页表项均占 4B，则进程的页目录和页表共占多少页？写出计算过程。
 - 3) 若某指令周期内访问的虚拟地址为 0100 0000H 和 0111 2048H，则进行地址转换时共访问多少个二级页表？说明理由。
20. 【2018 统考真题】某计算机采用页式虚拟存储管理方式，按字节编址，CPU 进行存储访问的过程如下图所示，回答下列问题。
- 1) 某虚拟地址对应的页目录号为 6，在相应的页表中对应的页号为 6，页内偏移量为 8，该虚拟地址的十六进制表示是什么？
 - 2) 寄存器 PDBR 用于保存当前进程的页目录起始地址，该地址是物理地址还是虚拟地址？进程切换时，PDBR 的内容是否会变化？说明理由。同一进程的线程切换时，PDBR 的内容是否会变化？说明理由。
 - 3) 为了支持改进型 CLOCK 置换算法，需要在页表项中设置哪些字段？



21.【2020 统考真题】某 32 位系统采用基于二级页表的请求分页存储管理方式，按字节编址，页目录项和页表项长度均为 4 字节，虚拟地址结构如下所示。

页目录号（10 位）	页号（10 位）	页内偏移量（12 位）
------------	----------	-------------

某 C 程序中数组 $a[1024][1024]$ 的起始虚拟地址为 1080 0000H，数组元素占 4 字节，该程序运行时，其进程的页目录起始物理地址为 0020 1000H，请回答下列问题。

- 1) 数组元素 $a[1][2]$ 的虚拟地址是什么？对应的页目录号和页号分别是什么？对应的页目录项的物理地址是什么？若该目录项中存放的页框号为 00301H，则 $a[1][2]$ 所在页对应的页表项的物理地址是什么？
- 2) 数组 a 在虚拟地址空间中所占的区域是否必须连续？在物理地址空间中所占的区域是否必须连续？
- 3) 已知数组 a 按行优先方式存放，若对数组 a 分别按行遍历和按列遍历，则哪种遍历方式的局部性更好？

3.2.10 答案与解析

一、单项选择题

1. B

装入程序时，只将程序的一部分装入内存，而将其余部分留在外存，就可以启动程序执行。采用连续分配方式时，会使相当一部分内存空间都处于暂时或“永久”的空闲状态，造成内存资源的严重浪费，也无法从逻辑上扩大内存容量，因此虚拟内存的实现只能建立在离散分配的内存管理的基础上。有以下三种实现方式：① 请求分页存储管理；② 请求分段存储管理；③ 请求段页式存储管理。虚拟存储器容量既不受外存容量限制，又不受内存容量限制，而是由 CPU 的寻址范围决定的。

2. B

在请求分页存储器中，由于页面尺寸增大，存放程序需要的页帧数就会减少，因此缺页中断

的次数也会减少。

3. B

缺页中断是访存指令引起的，说明所要访问的页面不在内存中，进行缺页中断处理并调入所要访问的页后，访存指令显然应该重新执行。

4. D

缺页中断调入新页面，肯定要修改页表项和分配页框，所以 I、III 可能发生，同时内存没有页面，需要从外存读入，会发生磁盘 I/O。

5. B

用户进程访问内存时缺页，会发生缺页中断。发生缺页中断时，系统执行的操作可能是置换页面或分配内存。系统内没有越界错误，不会进行越界出错处理。

6. B

虚拟存储技术并未实际扩充内存、外存，而是采用相关技术相对地扩充主存。

7. A

多次性、对换性和离散性是虚拟内存的特征，一次性则是传统存储系统的特征。

8. C

虚拟存储技术基于程序的局部性原理。局部性越好，虚拟存储系统越能更好地发挥作用。

9. B

请求分页存储管理方式和基本分页存储管理方式的区别是，前者采用虚拟技术，因此开始运行时，不必将作业全部一次性装入内存，而后者不是。

10. D

CLOCK 算法选择将最近未使用的页面置换出去，因此又称 NRU 算法。

11. C

无论采用什么页面置换算法，每种页面第一次访问时不可能在内存中，必然发生缺页，所以缺页次数大于等于 n 。

12. D

请求分页存储管理中，若采用 FIFO 页面淘汰算法，可能会产生当驻留集增大时页故障数不减反增的 Belady 异常。然而，还有另外一种情况。例如，页面序列为 1, 2, 3, 1, 2, 3，当页帧数为 2 时产生 6 次缺页中断，当页帧数为 3 时产生 3 次缺页中断。所以在请求分页存储管理中，若采用 FIFO 页面淘汰算法，则当可供分配的页帧数增加时，缺页中断的次数可能增加，也可能减少。

13. D

虚拟存储器的最大容量是由计算机的地址结构决定的，与主存容量和外存容量没有必然的联系，其虚拟地址空间为 $2^{32}B$ 。

14. B

虽然从实际使用来说，虚拟存储器能使得进程的可用内存扩大到内外存容量之和，但进程的内存寻址仍由计算机的地址结构决定，这就决定了虚拟存储器理论上的最大容量。比如，64 位系统环境下，虚拟内存技术使得进程可用内存空间达 $2^{64}B$ ，但外存显然是达不到这个大小的。

15. C

利用 LRU 置换算法时的置换如下图所示。

访问页面	1	8	1	7	8	2	7	2	1	8	3	8	2	1	3	1	7	1	3	7
物理块 1	1	1		1		1				1						1				
物理块 2		8		8		8				8						7				
物理块 3				7		7				3						3				
物理块 4						2				2						2				
缺页否	√	√		√		√				√					√					

分别在访问第 1 个、第 2 个、第 4 个、第 6 个、第 11 个、第 17 个页面时产生中断，共产生 6 次中断。

16. D

LRU 算法需要对所有页最近一次被访问的时间进行记录，查找时间最久的进行替换，这涉及排序，对置换算法而言，开销太大。为此需要在页表项中增加 LRU 位，选项 A 可看作是“耗费高”这一结果，选项 D 才是造成选项 A 的原因。

17. A

页表项中的合法位信息显示本页面是否在内存中，即决定了是否会发生页面故障。

18. D

抖动是进程的页面置换过程中，频繁的页面调度（缺页中断）行为，所有的页面调度策略都不可能完全避免抖动。

19. C

基于局部性原理：在程序装入时，不必将其全部读入内存，而只需将当前需要执行的部分页或段读入内存，就可让程序开始执行。在程序执行过程中，若需执行的指令或访问的数据尚未在内存（称为缺页或缺段）中，则由处理器通知操作系统将相应的页或段调入内存，然后继续执行程序。由于程序具有局部性，虚拟存储管理在扩充逻辑地址空间的同时，对程序执行时内存调换的代价很小。

20. B

虚拟存储扩充内存的基本方法是将一些页或段从内存中调入、调出，而调入、调出的基本手段是覆盖与交换。

21. B

请求分页存储管理就是为了解决内存容量不足而使用的方法，它基于局部性原理实现了以时间换取空间的目的。它的主要特点自然是间接扩充了内存。

22. C

当需要置换页面时，置换算法根据修改位和访问位选择调出内存的页面。

23. D

内存抖动是指频繁地引起主存页面淘汰后又立即调入，调入后又很快淘汰的现象。这是由页面置换算法不合理引起的一种现象，是页面置换算法应当尽量避免的。

24. B

FIFO 是队列类算法，有 Belady 现象；选项 C、D 均为堆栈类算法，理论上可以证明不会出现 Belady 现象。

25. B

页式虚拟存储管理的主要特点是，不要求将作业同时全部装入主存的连续区域，一般只装入 10%~30%。不要求将作业装入主存连续区域是所有离散式存储管理（包括页式存储管理）的特

点；页式虚拟存储管理需要进行缺页中断处理和页面置换。

26. C

虚拟存储技术是基于页或段从内存的调入/调出实现的，需要有请求机制的支持。

27. C

计算机系统中，为了提高系统的存取速度，在地址映射机制中增加一个小容量的硬件部件——快表（又称相联存储器），用来存放当前访问最频繁的少数活动页面的页号。快表查找内存块的物理地址消耗的时间大大降低，使得系统效率得到很大提高。

28. C

根据缺页中断的处理流程，产生缺页中断后，首先去内存寻找空闲物理块，若内存没有空闲物理块，使用相应的页面置换算法决定淘汰页面，然后调出该淘汰页面，最后调入该进程需要访问的页面。

29. C

页面大小为 4KB，因此页内偏移为 12 位。系统采用 48 位虚拟地址，因此虚页号 $48 - 12 = 36$ 位。采用多级页表时，最高级页表项不能超出一页大小；每页能容纳的页表项数为 $4KB/8B = 512 = 2^9$ ， $36/9 = 4$ ，因此应采用 4 级页表，最高级页表项正好占据一页空间，所以本题选择 C。

30. B

I 正确：例如，使用先进先出（FIFO）页面置换算法，页面引用串为 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 时，分配 3 帧时产生 9 次缺页中断，分配 4 帧时产生 10 次缺页中断。II 错误：最近最少使用（LRU）页面置换算法没有这样的问题。III 错误。IV 正确：若页面在内存中，不会产生缺页中断，即不会出现页面的调入/调出，不是虚拟存储器（包括作为虚拟内存那部分硬盘）。综上分析，I、IV 正确。

31. D

用于交换空间的磁盘利用率已达 97.7%，其他设备的利用率为 5%，CPU 的利用率为 20%，说明在任务作业不多的情况下交换操作非常频繁，因此判断物理内存严重短缺。

32. B

I 正确：增大内存的容量。增大内存可使每个程序得到更多的页面，能减少缺页率，进而减少换入/换出过程，可提高 CPU 的利用率。II 错误：增大磁盘交换区的容量。因为系统实际已处于频繁的换入/换出过程中，不是因为磁盘交换区容量不够，因此增大磁盘交换区的容量无用。III 正确：减少多道程序的度数。可以提高 CPU 的利用率，因为从给定的条件知道磁盘交换区的利用率为 99.7%，说明系统现在已经处于频繁的换入/换出过程中，可减少主存中的程序。IV 错误：增加多道程序的度数。系统处于频繁的换入/换出过程中，再增加主存中的用户进程数，只能导致系统的换入/换出更频繁，使性能更差。V 错误：使用更快速的磁盘交换区。因为系统现在处于频繁的换入/换出过程中，即使采用更快的磁盘交换区，其换入/换出频率也不会改变，因此没用。VI 错误：使用更快速的 CPU。系统处于频繁的换入/换出过程中，CPU 处于空闲状态，利用率不高，提高 CPU 的速度无济于事。综上分析：I、III 可以改进 CPU 的利用率。

33. A

在具有对换功能的操作系统中，通常把外存分为文件区和对换区。前者用于存放文件，后者用于存放从内存换出的进程。抖动现象是指刚刚被换出的页很快又要被访问，为此又要换出其他页，而该页又很快被访问，如此频繁地置换页面，以致大部分时间都花在页面置换上，导致系统性能下降。撤销部分进程可以减少所要用到的页面数，防止抖动。对换区大小和进程优先级都与抖动无关。

34. C

虚实地址转换是指逻辑地址和物理地址的转换。增大快表容量能把更多的表项装入快表，会加快虚实地址转换的平均速率；让页表常驻内存可以省去一些不在内存中的页表从磁盘上调入的过程，也能加快虚实地址转换；增大交换区对虚实地址转换速度无影响，因此 I、II 正确，选 C。

35. A

只有 FIFO 算法会导致 Belady 异常，选 A。

36. A

改进型 CLOCK 置换算法执行的步骤如下：

- 1) 从指针的当前位置开始，扫描帧缓冲区。在这次扫描过程中，对使用位不做任何修改。
选择遇到的第一个帧 ($A = 0, M = 0$) 用于替换。
- 2) 若第 1) 步失败，则重新扫描，查找 ($A = 0, M = 1$) 的帧。选择遇到的第一个这样的帧用于替换。在这个扫描过程中，对每个跳过的帧，将其使用位设置成 0。
- 3) 若第 2) 步失败，则指针将回到它的最初位置，并且集合中所有帧的使用位均为 0。重复第 1) 步，并在有必要时重复第 2) 步，这样将可以找到供替换的帧。

因此，该算法淘汰页的次序为 $(0, 0), (0, 1), (1, 0), (1, 1)$ ，即 A 正确。

37. C

对各进程进行固定分配时页面数不变，不可能出现全局置换。而 A、B、D 是现代操作系统中常见的 3 种策略。

38. A

可以采用书中常规的解法思路，也可以采用便捷法。对页号序列从后往前计数，直到数到 4 (页框数) 个不同的数字为止，这个停止的数字就是要淘汰的页号（最近最久未使用的页），题中为页号 2。

39. A

在任一时刻 t ，都存在一个集合，它包含所有最近 k 次（该题窗口大小为 6）内存访问所访问过的页面。这个集合 $w(k, t)$ 就是工作集。题中最近 6 次访问的页面分别为 6, 0, 3, 2, 3, 2，去除重复的页面，形成的工作集为 {6, 0, 3, 2}。

40. C

最近最久未使用 (LRU) 算法每次执行页面置换时会换出最近最久未使用过的页面。第一次访问 5 页面时，会把最久未被使用的 1 页面换出，第一次访问 3 页面时，会把最久未访问的 2 页面换出。具体的页面置换情况如下图所示。

访问页面	0	1	2	7	0	5	3	5	0	2	7	6
物理块 1	0	0	0	0	0	0	0	0	0	0	0	0
物理块 2		1	1	1	1	5	5	5	5	5	5	6
物理块 3			2	2	2	2	3	3	3	3	7	7
物理块 4				7	7	7	7	7	7	2	2	2
缺页否	√	√	√	√		√	√			√	√	√

需要注意的是，题中间的是页置换次数，而不是缺页次数，所以前 4 次缺页未换页的情况不考虑在内，答案为 5 次，因此选 C。

41. D

I 影响缺页中断的频率，缺页率越高，平均访存时间越长；II 和 IV 影响缺页中断的处理时间，

中断处理时间越长，平均访存时间越长；III 影响访问页表和访问目标物理地址的时间，故 I、II、III 和 IV 均正确。

二、综合应用题

1. 解答：

- 1) 覆盖技术与虚拟存储技术最本质的不同在于，覆盖程序段的最大长度要受内存容量大小的限制，而虚拟存储器中程序的最大长度不受内存容量的限制，只受计算机地址结构的限制。另外，覆盖技术中的覆盖段由程序员设计，且要求覆盖段中的各个覆盖具有相对独立性，不存在直接联系或相互交叉访问；而虚拟存储技术对用户的程序段没有这种要求。
- 2) 交换技术就是把暂时不用的某个程序及数据从内存移到外存中，以便腾出必要的内存空间，或把指定的程序或数据从外存读到内存中的一种内存扩充技术。交换技术与虚存中使用的调入/调出技术的主要相同点是，都要在内存与外存之间交换信息。交换技术与虚存中使用的调入/调出技术的主要区别是：交换技术调入/调出整个进程，因此一个进程的大小要受内存容量大小的限制；而虚存中使用的调入/调出技术在内存和外存之间来回传递的是页面或分段，而不是整个进程，从而使得进程的地址映射具有更大的灵活性，且允许进程的大小比可用的内存空间大。

2. 解答：

一页的大小为 32B，逻辑地址结构为：低 5 位为页内位移，其余高位为页号。

101 （八进制）= 00100001 （二进制），则页号为 2，在相联存储器中，对应的页帧号为 f3，即物理地址为(f3, 1)。

204 （八进制）= 010000100 （二进制），则页号为 4，不在相联存储器中，查内存的页表得页帧号为 f5，即物理地址为(f5, 4)，并用其更新相联存储器中的一项。

576 （八进制）= 101111110 （二进制），则页号为 11，已超出页表范围，即产生越界中断。

3. 解答：

- 1) 80% 的访问的页表项在关联寄存器中，访问耗时 $1\mu s$ 。
- 2) 18% 的访问的页表项不在关联寄存器中，但在内存中，耗时 $(1+1)\mu s$ 。
- 3) 2% 的访问产生缺页中断，访问耗时 $(1\mu s + 1\mu s + 20ms)$ 。

从而有效访问时间为 $80\% \times 1 + 18\% \times 2 + 2\% \times (1 \times 2 + 20 \times 1000) = 401.2\mu s$ 。

注意：针对 3) 中的耗时情况，有些读者可能对缺页中断的页面调度过程有些模糊，导致此情况下的访问耗时计算出现偏差。题目中已经明确说明“页表保存在主存”，意味着物理页号一定可以通过查找内存获得并计算出相应的物理地址。即若关联寄存器命中（可以取得页框号，再结合逻辑地址中的偏移量，便可算得对应的物理地址。至此，已经获得物理地址），则仅访问一次内存（根据物理地址取得相应的页面，耗时 $1\mu s$ ）即可；若未命中，则还要从主存中【*注：若地址变换是通过查找内存中的页表完成的，则还应将这次所查到的页表项存入关联寄存器中。】取出相应的页表项（耗时 $1\mu s$ ），再根据得到的物理地址访问内存取得对应的页面（耗时 $1\mu s$ ）。但是，无论关联寄存器是否命中，欲访问的页面【非页表（Page）】不一定在主存中，即若页面不在内存中，则产生缺页中断，操作系统需要启动磁盘将缺页调入内存。题目中明确说明“剩下的 20% 中，10% 的访问（即总数的 2%）会产生缺页”，其中“剩下的 20%”意味着是通过查找内存（关联寄存器未命中）得到物理地址的（耗时 $1\mu s$ ），缺页中断（耗时 $20ms$ ）后，将缺页调入主存，此时“系统恢复缺页中断发生前的状态，将程序指令器重新指向引起缺页中断的指令，重新执行

该指令”，这时页表项已在关联寄存器中（对此不解的读者请回看上文的“*注”处），则根据取得的物理地址仅访存一次即可取得对应的页面（耗时 $1\mu s$ ）。

注意：建议读者结合《计算机组成原理考研复习指导》中的“虚拟存储器”小节进行复习。会总结的读者在完成本节习题后应会注意到：该题、第 5 题（2009 真题）中的 1)、第 9 题、第 10 题中的 2) 都是同一类题，望读者在完成本题的基础上总结出求解该类题型的方法，以便在以后遇到类似的题目时得心应手。读者应做到基础扎实、注意细节、触类旁通。

4. 解答：

发生页故障的原因是，当前访问的页不在主存，需要将该页调入主存。此时不管主存中是否已满（已满则先调出一页），都要发生一次页故障，即无论怎样安排， n 个不同的页号在首次进入主存时必须要发生一次页故障，总共发生 n 次，这是页故障数的下限。虽然不同的页号数为 n 小于等于总长度 p （访问串可能会有一些页重复出现），但驻留集 $m < n$ ，所以可能会有某些页进入主存后又被调出主存，当再次访问时又发生一次页故障的现象，即有些页可能会出现多次页故障。最差的情况是每访问一个页号时，该页都不在主存中，这样共发生 p 次故障。

因此，对于 FIFO、LRU 置换算法，页故障数的上限均为 p ，下限均为 n 。例如，当 $m = 3, p = 12, n = 4$ 时，有访问串 1 1 1 2 2 3 3 3 4 4 4 4，则页故障数为 4，这是下限 n 的情况。又如，有访问串 1 2 3 4 1 2 3 4 1 2 3 4，则页故障数为 12，这是上限 p 的情况。

5. 解答：

1) 根据页式管理的工作原理，应先考虑页面大小，以便将页号和页内位移分解出来。页面大小为 4KB，即 2^{12} ，得到页内位移占虚地址的低 12 位，页号占剩余高位。可得三个虚地址的页号 P 如下（十六进制的一位数字转换成二进制的 4 位数字，因此十六进制的低三位正好为页内位移，最高位为页号）：

2362H: $P = 2$ ，访问快表 10ns，因初始为空，访问页表 100ns 得到页框号，合成物理地址后访问主存 100ns，共计 $10ns + 100ns + 100ns = 210ns$ 。

1565H: $P = 1$ ，访问快表 10ns，落空，访问页表 100ns 落空，进行缺页中断处理 $10^8 ns$ ，访问快表 10ns，合成物理地址后访问主存 100ns，共计 $10ns + 100ns + 10^8 ns + 10ns + 100ns = 100000220ns$ 。

25A5H: $P = 2$ ，访问快表，因第一次访问已将该页号放入快表，因此花费 10ns 便可合成物理地址，访问主存 100ns，共计 $10ns + 100ns = 110ns$ 。

2) 当访问虚地址 1565H 时，产生缺页中断，合法驻留集为 2，必须从页表中淘汰一个页面，根据题目的置换算法，应淘汰 0 号页面，因此 1565H 的对应页框号为 101H。由此可得 1565H 的物理地址为 101565H。

6. 解答：

1) 根据页面走向，使用最佳置换算法时，页面置换情况见下表。

物理块数为 3 时：

走向	4	3	2	1	4	3	5	4	3	2	1	5
块 1	4	4	4	4	4	4	4	4	4	2	2	2
块 2		3	3	3	3	3	3	3	3	3	1	1
块 3			2	1	1	1	5	5	5	5	5	5
缺页	√	√	√	√			√			√	√	

缺页率为 $7/12$ 。

物理块数为 4 时：

走向	4	3	2	1	4	3	5	4	3	2	1	5
块1	4	4	4	4	4	4	4	4	4	4	1	1
块2		3	3	3	3	3	3	3	3	3	3	3
块3			2	2	2	2	2	2	2	2	2	2
块4				1	1	1	5	5	5	5	5	5
缺页	√	√	√	√			√			√		

缺页率为 6/12。

由上述结果可以看出，增加分配作业的内存块数可以降低缺页率。

- 2) 根据页面走向，使用先进先出页面淘汰算法时，页面置换情况见下表。

物理块数为 3 时：

走向	4	3	2	1	4	3	5	4	3	2	1	5
块1	4	4	4	1	1	1	5	5	5	5	5	5
块2		3	3	3	4	4	4	4	4	2	2	
块3			2	2	2	3	3	3	3	3	1	
缺页	√	√	√	√	√	√	√			√	√	

缺页率为 9/12。

物理块数为 4 时：

走向	4	3	2	1	4	3	5	4	3	2	1	5
块1	4	4	4	4	4	4	5	5	5	5	1	1
块2		3	3	3	3	3	3	4	4	4	4	5
块3			2	2	2	2	2	2	3	3	3	3
块4				1	1	1	1	1	1	2	2	2
缺页	√	√	√	√			√	√	√	√	√	√

缺页率为 10/12。

由上述结果可以看出，对先进先出算法而言，增加分配作业的内存块数反而使缺页率上升，即出现 Belady 现象。

- 3) 根据页面走向，使用最近最久未使用页面淘汰算法时，页面置换情况见下表。

物理块数为 3 时：

走向	4	3	2	1	4	3	5	4	3	2	1	5
块1	4	4	4	1	1	1	5	5	5	2	2	2
块2		3	3	3	4	4	4	4	4	4	1	1
块3			2	2	2	3	3	3	3	3	3	5
缺页	√	√	√	√	√	√	√			√	√	√

缺页率为 10/12。

物理块数为 4 时：

走向	4	3	2	1	4	3	5	4	3	2	1	5
块1	4	4	4	4	4	4	4	4	4	4	4	5
块2		3	3	3	3	3	3	3	3	3	3	3
块3			2	2	2	2	5	5	5	5	1	1
块4				1	1	1	1	1	1	2	2	2
缺页	√	√	√	√			√			√	√	√

缺页率为 $8/12$ 。

由上述结果可以看出，增加分配作业的内存块数可以降低缺页率。

7. 解答：

- 1) 系统出现“抖动”现象。这时若再增加并发进程数，反而会恶化系统性能。页式虚拟存储系统因“抖动”现象而未能充分发挥功用。
- 2) 系统正常。不需要采取什么措施。
- 3) CPU 没有充分利用。应该增加并发进程数。

8. 解答：

在缺页中断处理完成，调入请求页面后，还需 $1\mu s$ 的存取访问，即

- 1) 当未缺页时，直接访问内存，用时 $1\mu s$ 。
- 2) 当缺页时，若未修改，则用时 $8ms + 1\mu s$ 。
- 3) 当缺页时，而且修改了，则用时 $20ms + 1\mu s$ 。

因此，设最大缺页中断率为 p ，有 $(1 - p) \times 1\mu s + (1 - 70\%) \times p \times (1\mu s + 8ms) + 70\% \times p \times (1\mu s + 20ms) = 2\mu s$ ，即 $1\mu s + (1 - 70\%) \times p \times 8ms + 70\% \times p \times 20ms = 2\mu s$ ，解得 $p = 0.00006$ 。

9. 解答：

- 1) 页面大小为 4KB，因此页内偏移为 12 位。系统采用 48 位虚拟地址，因此虚页号为 $48 - 12 = 36$ 位。采用多级页表时，最高级页表项不能超出一页大小；每页能容纳的页表项数为 $4KB/8B = 512 = 2^9$ ， $36/9 = 4$ ，因此应采用 4 级页表，最高级页表项正好占据一页空间。
- 2) 系统进行页面访问操作时，首先读取页面对应的页表项，有 98% 的概率可以在 TLB 中直接读取到，然后进行地址转换，访问内存读取页面；若 TLB 未命中，则要通过一次内存访问来读取页表项。页面平均访问时间为

$$98\% \times (10 + 100) + (1 - 98\%) \times (10 + 100 + 100) = 112ns$$

- 3) 二级页表的平均访问时间计算同理：

$$98\% \times (10 + 100) + (1 - 98\%) \times (10 + 100 + 100 + 100) = 114ns$$

- 4) 设快表命中率为 p ，则应满足

$$p \times (10 + 100) + (1 - p) \times (10 + 100 + 100 + 100) \leq 120ns$$

解得 $p \geq 95\%$ 。

- 5) 系统采用 48 位虚拟地址，每段最大为 4GB，因此段内地址为 32 位，段号为 $48 - 32 = 16$ 位。每个用户最多可以有 2^{16} 段。段内采用页式地址，与 1) 中计算同理， $(32 - 12)/9$ ，取上整为 3，因此段内应采用 3 级页表。

注意：在采用多级页表的页式存储管理中，若快表命中，则只需要一次访问内存操作即可存取指令或数据，这一点需要注意和理解。以本题 1) 中假设的条件为例，不考虑分段时，需要 4 级页表。若快表未命中，则需要从虚拟地址的高位起，每 9 位逐级访问各级页表，第 5 次才能访问到指令或数据所在的内存页面。

若快表命中，则首先考虑快表中的实际内容：快表存放经常被访问的页面对应的页表项，页表项中是完整的 $48 - 12 = 36$ 位页面号，所以根据快表可以直接对虚拟地址进行转换。因此多级页表中，快表命中时同样只需要一次访问内存操作。根本原因在于，快表提供了进行地址转换的完整的页面号，而不是某一级的页面号。

10. 解答：

- 1) 物理块数为 3 时，缺页情况见下表：

访问串	1	3	2	1	1	3	5	1	3	2	1	5
内存	1	1	1	1	1	1	1	1	1	1	1	1
	3	3	3	3	3	3	3	3	3	3	3	5
		2	2	2	2	5	5	5	2	2	2	
缺页	√	√	√			√			√		√	

缺页次数为 6，缺页率为 $6/12 = 50\%$ 。

2) 物理块数为 4 时，缺页情况见下表：

访问串	1	3	2	1	1	3	5	1	3	2	1	5
内存	1	1	1	1	1	1	1	1	1	1	1	1
	3	3	3	3	3	3	3	3	3	3	3	3
		2	2	2	2	2	2	2	2	2	2	2
						5	5	5	5	5	5	5
缺页	√	√	√			√			√		√	

缺页次数为 4，缺页率为 $4/12 = 33\%$ 。

注意：当分配给作业的物理块数为 4 时，注意到作业请求页面序列中只有 4 个页面，可以直接得出缺页次数为 4，而不需要按表列出缺页情况。

11. 解答：

- 1) FIFO 算法：按照先进先出规则，最先进入的页帧号应最先替换，因此访问第 4 页时，缺页中断程序应选择的是第 3 号页帧。由于该页帧的修改位是 1，在换出主存后应先进行回写，即重新保存。
- 2) LRU 算法：最近一次访问时间离当前最远的页帧应被选择换出，因此缺页中断程序选择的是 1 号页帧。
- 3) 改进型 CLOCK 算法：第一轮扫描淘汰访问位和修改位都为 0 的页面，因此淘汰 1 号页面。

12. 解答：

由于驻留集大小任意，现要求两种算法的替换页面和缺页情况完全一样，就意味着要求 FIFO 与 LRU 的置换选择一致。FIFO 替换最早进入主存的页面，LRU 替换上次访问以来最久未被访问的页面，这两个页面一致。就是说，最先进入主存的页面在此次缺页之前不能再被访问，这样该页面也就同时是最久未被访问的页面。

例如，合法驻留集大小为 4 时，对访问串 1, 2, 3, 4, 1, 2, 5，当 5 号页面调入主存时，应在 1, 2, 3, 4 页中选择一个替换，FIFO 选择 1，LRU 选择 3。原因在于 1 号页面虽然最先进入主存，但由于其进入主存后又被再次访问，所以它不是最久未被访问的页面。若去掉对 1 号页面的第二次访问，则 FIFO 与 LRU 的替换选择就会相同。同理，当 5 号页面调入主存后，若再访问新的 6 号页面，则 2 号页面会遇到同样的问题。因此，以此类推，访问串中的所有页面号都应不同，但要注意到，连续访问相同页面时不影响后面的替换选择，所以对访问串的要求是：不连续的页面号均不相同。

13. 解答：

- 1) 页框号为 21。因为起始驻留集为空，而 0 页对应的页框为空闲链表中的第三个空闲页框 (21)，其对应的页框号为 21。
- 2) 页框号为 32。理由：因 $11 > 10$ ，因此发生第三轮扫描，页号为 1 的页框在第二轮已处于

空闲页框链表中，此刻该页又被重新访问，因此应被重新放回驻留集中，其页框号为 32。

3) 页框号为 41。理由：因为第 2 页从来没有被访问过，它不在驻留集中，因此从空闲页框链表中取出链表头的页框 41，页框号为 41。

4) 合适。理由：程序的时间局部性越好，从空闲页框链表中重新取回的机会越大，该策略的优势越明显。

14. 解答：

1) FIFO 置换算法选择最先进入内存的页面进行替换。由表中装入时间可知，第 2 页最先进入内存，因此 FIFO 置换算法将选择第 2 页替换。

2) LRU 置换算法选择最近最长时间未使用的页面进行替换。由表中的上次引用时间可知，第 1 页是最长时间未使用的页面，因此 LRU 置换算法将选择第 1 页替换。

3) 简单 CLOCK 置换算法从上一次位置开始扫描，选择第一个访问位为 0 的页面进行替换。由表中的 R(读)标志位可知，依次扫描 2, 3, 0(按装入顺序)，页面 0 未被访问，扫描结束，因此简单 CLOCK 置换算法将选择第 0 页替换。

4) 改进型 CLOCK 置换算法从上一次的位置开始扫描，首先寻找未被访问和修改的页面。

由表中的 R(读)标志位和 M(修改)标志位可知，只有页面 0 满足 $R=0$ 和 $M=0$ ，因此改进型 CLOCK 置换算法将选择第 0 页替换。

15. 解答：

程序 1 按行优先的顺序访问数组元素，与数组在内存中存放的顺序一致，每个内存页面可存放 200 个数组元素。这样，程序 1 每访问两行数组元素就产生一次缺页中断，所以程序 1 的执行过程会发生 50 次缺页。

程序 2 按列优先的顺序访问数组元素，由于每个内存页面存放两行数组元素，因此程序 2 每访问两个数组元素就产生一次缺页中断，整个执行过程会发生 5000 次缺页。

若每页只能存放 100 个整数，则每页仅能存放一行数组元素，同理可以计算出：程序 1 的执行过程产生 100 次缺页；程序 2 的执行过程产生 10000 次缺页。

以上说明缺页的次数与内存中数据存放的方式及程序执行的顺序有很大关系；同时说明，当缺页中断次数不多时，减小页面大小影响并不大，但缺页中断次数很多时，减小页面大小会带来很严重的影响。

16. 解答：

1) 由于该计算机的逻辑地址空间和物理地址空间均为 $64KB = 2^{16}B$ ，按字节编址，且页的大小为 $1K = 2^{10}$ ，因此逻辑地址和物理地址的地址格式均为

页号/页框号（6 位）	页内偏移量（10 位）
-------------	-------------

$17CAH = 0001\ 0111\ 1100\ 1010B$ ，可知该逻辑地址的页号为 $000101B = 5$ 。

2) 采用 FIFO 置换算法，与最早调入的页面即 0 号页面置换，其所在的页框号为 7，于是对应的物理地址为 $0001\ 1111\ 1100\ 1010B = 1FCAH$ 。

3) 采用 CLOCK 置换算法，首先从当前位置(2 号页框)开始顺时针寻找访问位为 0 的页面，当指针指向的页面的访问位为 1 时，就把该访问位清“0”，指针遍历一周后，回到 2 号页框，此时 2 号页框的访问位为 0，置换该页框的页面，于是对应的物理地址为 $0000\ 1011\ 1100\ 1010B = 0BCAH$ 。

17. 解答：

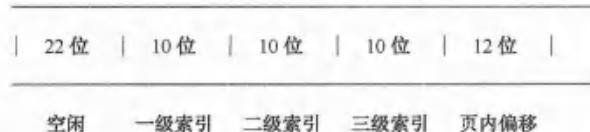
1) 函数 f1 的代码段中，所有指令的虚拟地址的高 20 位相同，因此 f1 的机器指令代码在同

一页中，仅占用1页。页目录号用于寻找页目录的表项，该表项包含页表的位置。页表索引用于寻找页表的表项，该表项包含页的位置。

- 2) push ebp 指令的虚拟地址的最高10位(页目录号)为00 0000 0001，中间10位(页表索引)为00 0000 0001，所以取该指令时访问了页目录的第2个表项(1号表项)，在对应的页表中访问了第2个表项(1号表项)。
- 3) 在执行 scanf() 的过程中，进程 P 因等待输入而从执行态变为阻塞态。输入结束时，P 被中断处理程序唤醒，变为就绪态。P 被调度程序调度，变为运行态。CPU 状态会从用户态变为内核态。

18. 解答：

- 1) 页面大小为4KB，每个页表项大小为4B，因此在每个页表当中，共有1024个页表项，对于每个层次的页表来说，都满足这一点，这样每级页表的索引均为10位，由于页面大小为4KB，所以页内偏移地址为12位。逻辑地址被划分为5个部分：



可访问的虚拟地址空间大小为 $2^{42}B = 4TB$ 。

- 2) 假定一个页面的大小为 2^Y ，即页内偏移地址为 Y 位，每个页面可以包含 $2^Y/8 = 2^{(Y-3)}$ 个页表项，因此每级页表的索引位为 $Y-3$ 位，共有4级页表，所以 $4(Y-3) + Y \leq 64$ ， $Y \leq 15.2$ ，因此 $Y=15$ 。所以最大的页面大小为 $2^{15}B = 32KB$ 。

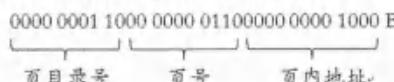
总结：求解这类题目的关键是清楚地划分逻辑地址，清楚地划分了逻辑地址的每个部分，这类题目就很容易求解。

19. 解答：

- 1) 页和页框大小均为4KB。进程的虚拟地址空间大小为 $2^{32}/2^{12} = 2^{20}$ 页。
- 2) $(2^{10} \times 4)/2^{12}$ (页目录所占页数) + $(2^{20} \times 4)/2^{12}$ (页表所占页数) = 1025 页。
- 3) 需要访问一个二级页表。因为虚拟地址 0100 0000H 和 0111 2048H 的最高10位的值都是4，访问的是同一个二级页表。

20. 解答：

- 1) 由图可知，地址总长度为32位，高20位为虚页号，低12位为页内地址，且虚页号高10位为页目录号，低10位为页号。展开成二进制表示为



因此十六进制表示为 0180 6008H。

- 2) PDBR 为页目录基址地址寄存器 (Page-Directory Base Register)，其存储页目录表物理内存基地址。进程切换时，PDBR 的内容会变化；同一进程的线程切换时，PDBR 的内容不会变化。每个进程的地址空间、页目录和 PDBR 的内容存在一一对应的关系。进程切换时，地址空间发生了变化，对应的页目录及其起始地址也相应变化，因此需要用进程切换后当前进程的页目录起始地址刷新 PDBR。同一进程中的线程共享该进程的地址空间，其线程发生切换时，地址空间不变，线程使用的页目录不变，因此 PDBR 的内容也不变。
- 3) 改进型 CLOCK 置换算法需要用到使用位和修改位，所以需要设置访问字段 (使用位)

和修改字段（脏位）。

21. 解答：

- 1) ① 页面大小 $= 2^{12}B = 4096B = 4KB$ 。每个数组元素 4B，每个页面可以存放 $4KB/4B = 1024$ 个数组元素，正好是数组的一行，数组 a 按行优先方式存放。1080 0000H 的虚页号为 10800H，因此 a[0] 行存放在虚页号为 10800H 的页面中，a[1] 行存放在页号为 10801H 的页面中。a[1][2] 的虚拟地址为 $10801\ 0000H + 4 \times 2 = 10801\ 008H$ 。
 ② 转换为二进制 0001000010 0000000001 000000001000，根据虚拟地址结构可知，对应的页目录号为 042H，页号为 001H。
 ③ 进程的页目录表起始地址为 0020 1000H，每个页目录项长 4B，因此 042H 号页目录项的物理地址是 $0020\ 1000H + 4 \times 42H = 0020\ 1108H$ 。
 ④ 页目录项存放的页框号为 00301H，二级页表的起始地址为 00301 0000H，因此 a[1][2] 所在页的页号为 001H，每个页表项 4B，因此对应的页表项物理地址是 $00301\ 0000H + 001H \times 4 = 00301\ 004H$ 。
- 2) 根据数组的随机存取特点，数组 a 在虚拟地址空间中所占的区域必须连续，由于数组 a 不止占用一页，相邻逻辑页在物理上不一定相邻，因此数组 a 在物理地址空间中所占的区域可以不连续。
- 3) 由 1) 可知每个页面正好可以存放一行的数组元素，“按行优先方式存放”意味着数组的同一行的所有元素都存放在同一个页面中，同一列的各个元素都存放在不同的页面中，因此数组 a 按行遍历的局部性较好。

3.3 本章疑难点

分页管理方式和分段管理方式在很多地方是相似的，比如在内存中都是不连续的、都有地址变换机构来进行地址映射等。但两者也存在许多区别，表 3.7 列出了分页管理方式和分段管理方式各方面的对比。

表 3.7 分页管理方式和分段管理方式的比较

	分 页	分 段
目的	页是信息的物理单位，分页是为实现离散分配方式，以削减内存的外零头，提高内存的利用率。或者说，分页仅是由于系统管理的需要而不是用户的需要	段是信息的逻辑单位，它含有一组意义相对完整的信息。分段的目的是能更好地满足用户的需要
长度	页的大小固定且由系统决定，由系统把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的，因而在系统中只能有一种大小的页面	段的长度不固定，决定于用户所编写的程序，通常由编译程序在对程序进行编译时，根据信息的性质来划分
地址空间	作业地址空间是一维的，即单一的线性地址空间，程序员利用一个记号即可表示一个地址	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
碎片	有内部碎片，无外部碎片	有外部碎片，无内部碎片
“共享”和“动态链接”	不容易实现	容易实现