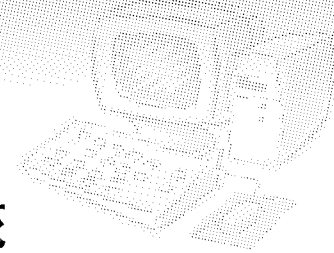


第3章　操作系统

本章学习目标：

- 掌握操作系统的基本概念，掌握进程调度的原则及相关算法。
- 掌握存储管理的主要功能，虚拟存储器的概念。
- 掌握文件管理的主要功能。
- 掌握设备的分类，掌握设备管理的主要功能。

本章主要介绍操作系统技术的发展和最基本、最重要的概念，包括操作系统的定义、功能、特征；进程的定义和特征，进程各种状态之间的转换，进程控制原语操作的意义，进程互斥与同步的概念；存储管理的主要功能；文件的分类，文件管理的主要功能；设备管理的主要功能及 Windows 操作系统的发展历程。

3.1　操作系统概述

3.1.1　操作系统的定义及作用

1. 操作系统的概念

操作系统是计算机系统中的一个系统软件，它是这样一些程序模块的集合——它们管理和控制计算机系统中的软件与硬件资源，合理地组织计算机工作流程，以便有效地利用这些资源为用户提供一个功能强大、使用方便和可扩展的工作环境，从而在计算机与其用户之间起到接口的作用。

注意　简单地说，操作系统是一组控制和管理计算机硬件与软件资源、合理地对各类作业进行调度、方便用户使用的程序的集合。

2. 操作系统的作用

(1) 作为扩展计算机功能的操作系统，是用户与计算机系统之间的接口。一台完全无软件的计算机系统称为裸机，即便其性能再强，相对于用户来讲，如果要面对计算机的指令集、存储组织、I/O 总线结构的编程则是十分困难的。对于一般程序员也并不想涉足硬件编

程的种种具体细节，而希望针对数据结构抽象地使用硬件。如果我们在裸机上覆盖一层I/O设备管理软件，用户便可以利用这层I/O设备管理软件提供给用户的接口来进行数据的输入/输出，那么用户此时看到的计算机是一台功能强大、使用方便的计算机。

(2) 操作系统是系统资源的管理者。从作为机器功能扩充的观点看，操作系统是为用户提供基本的方便的接口，这是一种自顶向下的观点或是自内向外的观点。但是从用户的角度出发或自底向上的观点来看，操作系统则用来管理一个复杂计算机系统的各个部分。

提示 从这个角度来看，操作系统的任务是在相互竞争的程序之间有序地控制对处理器、存储器以及其他I/O接口设备的分配。监视各种资源，随时记录它们的状态；实施某种策略以决定谁获得资源，何时获得，获得多少；分配资源供需求者使用；回收资源，以便再分配。

当计算机上覆盖了操作系统后，便为用户提供了一台功能显著增强，使用更加方便，效率明显提高的虚拟计算机。

注意 操作系统为用户完成所有与"硬件相关、应用无关"的工作，以给用户提供方便、效率、安全、可扩展性、开放性。

3.1.2 操作系统的功能、特性

1. 操作系统的功能

操作系统是用户与计算机硬件之间的接口，操作系统是对计算机硬件系统的第一次扩充，用户通过操作系统来使用计算机系统。换句话来说，操作系统紧靠着计算机硬件并在其基础上提供了许多新的设施和能力，从而使得用户能够方便、可靠、安全、高效地操纵计算机硬件和运行自己的程序。例如，改造各种硬件设施，使之更容易使用；提供原语或系统调用，扩展机器的指令系统。而这些功能到目前为止还难以由硬件直接实现。操作系统还合理组织计算机的工作流程，协调各个部件有效工作，为用户提供一个良好的运行环境。经过操作系统改造和扩充过的计算机不但功能更强，使用也更为方便，用户可以直接调用操作系统提供的许多功能，而无须了解许多软硬件使用细节。

操作系统可以提供虚拟计算机(Virtual Machine)。许多年以前，人们就认识到必须找到某种方法把硬件的复杂性与用户隔离开来，经过不断的探索和研究，目前采用的方法是在计算机裸机上加上一层又一层的软件来组成整个计算机系统，同时，为用户提供一个容易理解和便于程序设计的接口。在操作系统中，类似地把硬件细节隐藏并把它与用户隔离开的情况处处可见。

操作系统还是计算机系统的资源管理者。在计算机系统中，能分配给用户使用的各种硬件和软件设施总称为资源。资源包括两大类：硬件资源和信息资源。其中，硬件资源分为处理器、存储器、I/O设备等，I/O设备又分为输入型设备、输出型设备和存储型设备；信息资源则分为程序和数据等。资源管理是操作系统的一项主要任务，而控制程序执行、扩充及其功能、屏蔽使用细节、方便用户使用、组织合理工作流程、改善人机界面等等都可以从资源管理的角度去理解。下面就从资源管理的观点来看操作系统具有的几

个主要功能。

(1) 处理器管理

处理器管理的第一项工作是处理中断事件，硬件只能发现中断事件，捕捉它并产生中断信号，但不能进行处理。配置了操作系统，就能对中断事件进行处理。

处理器管理的第二项工作是处理器调度。在单用户单任务的情况下，处理器仅为一个用户的一个任务所独占，处理器管理的工作十分简单。但在多道程序或多用户的情况下，组织多个作业或任务执行时，就要解决处理器的调度、分配和回收等问题。近年来设计出各种各样的多处理器系统，处理器管理就更加复杂。为了实现处理器管理的功能，操作系统引入了进程(Process)的概念，处理器的分配和执行都是以进程为基本单位。随着并行处理技术的发展，为了进一步提高系统并行性，使并发执行单位的粒度变细，操作系统又引入了线程(Thread)的概念。对处理器的管理可归结为对进程和线程的管理，包括：①进程控制和管理；②进程同步和互斥；③进程通信；④进程死锁；⑤处理器调度，又分高级调度、中级调度、低级调度等；⑥线程控制和管理。

注意 正是由于操作系统对处理器的管理策略不同，其提供的作业处理方式也就不同，例如，批处理方式、分时处理方式、实时处理方式等，从而，呈现在用户面前，成为具有不同性质和不同功能的操作系统。

(2) 存储器管理

存储器管理的主要任务是管理存储器资源，为多道程序运行提供有力的支撑。存储管理的主要功能包括：①存储分配。存储管理将根据用户程序的需要给它分配存储器资源。②存储共享。存储管理能让主存中的多个用户程序实现存储资源的共享，以提高存储器的利用率。③存储保护。存储管理要把各个用户程序相互隔离起来互不干扰，更不允许用户程序访问操作系统的程序和数据，从而保护用户程序存放在存储器中的信息不被破坏。④存储扩充。由于物理内存容量有限，难以满足用户程序的需求，存储管理还应该能从逻辑上来扩充内存储器，为用户提供一个比内存实际容量大得多的编程空间，方便用户编程和使用。

注意 操作系统的存储器管理功能与硬件存储器的组织结构密切相关，操作系统设计者应根据硬件情况和用户使用需要，采用各种相应的有效存储器资源分配策略和保护措施。

(3) 设备管理

设备管理的主要任务是管理各类外围设备，完成用户提出的 I/O 请求，加快 I/O 信息的传送速度，发挥 I/O 设备的并行性，提高 I/O 设备的利用率；以及提供每种设备的设备驱动程序和中断处理程序，向用户屏蔽硬件使用细节。为实现这些任务，设备管理应该具有以下功能：①提供外围设备的控制与处理；②提供缓冲区的管理；③提供外围设备的分配；④提供共享型外围设备的驱动；⑤实现虚拟设备。

提示 从分配的角度看，外部设备可分为共享设备(可以同时为多个用户服务，例如磁盘机)和独占设备(在一段时间内只能为一个用户服务，如打印机)。对于独占设备，系统可以按照一定策略把它轮流分配给请求使用的用户，也可以采用虚拟设备的方法，例如将

行式打印机作为虚拟设备，用户的打印输出申请由操作系统先转换成写盘操作，将待打印信息暂时存盘，到适当时候由操作系统控制，成批向打印机输出，这种方法也叫假脱机打印。它提高了设备效率，也避免了在高峰时间因输出操作而过多占用 CPU 时间。

(4) 文件管理

上述三种管理是针对计算机硬件资源的管理，文件管理则是对系统的信息资源的管理。在现代计算机中，通常把程序和数据以文件形式存储在外存储器上供用户使用，这样，外存储器上保存了大量文件，对这些文件如不能采取良好的管理方式，就会导致混乱或破坏，造成严重后果。为此，在操作系统中配置了文件管理，它的主要任务是对用户文件和系统文件进行有效管理，实现按名存取；实现文件的共享、保护和保密，保证文件的安全性；并提供给用户一套能方便使用文件的操作和命令。

提示 具体来说，文件管理要完成以下任务：①提供文件逻辑组织方法；②提供文件物理组织方法；③提供文件的存取方法；④提供文件的使用方法；⑤实现文件的目录管理；⑥实现文件的存取控制；⑦实现文件的存储空间管理。

(5) 用户接口

操作系统的重要目标是方便用户使用计算机。操作系统内核通过系统调用向应用程序提供接口，方便用户进程对文件和目录的操作、申请和释放内存、对各类设备进行 I/O 操作以及对进程进行控制。操作系统通过用户接口提供对文件系统的操作命令，提供系统维护、系统开发接口，以及向用户提供有关信息。操作系统的用户接口有三类：程序接口、命令行接口和图形接口。

2. 操作系统的特性

(1) 并发性

并发(Concurrence)是指两个或多个事件在同一时间间隔内发生。并发与并行是有区别的。并行是指两个或多个事件在同一时刻发生。计算机内有多个进程并行执行，这只有在多 CPU 的系统中才能实现。

提示 在单 CPU 的计算机系统中，多个进程是不可能同时执行的。并发是从宏观上看多个进程的活动，这些进程在串行并交错地运行，由操作系统负责这些进程间的运行切换。

进程的并发执行，有效地改善了系统资源的利用率和提高了系统的吞吐量，复杂操作系统必须具有控制和管理各种并发活动的能力。

(2) 共享性

共享(Sharing)是指多个用户或进程共享系统的软、硬件资源。共享可以提高各种系统设备和系统软件的使用效率。

并发和共享是操作系统的两个最基本的特征，这两者之间又是互为存在条件的。资源共享是以进程并发为前提条件的，若系统不允许进程并发执行，自然不存在资源共享问题。若系统不能对资源共享实施有效的管理，也必将影响到进程的并发执行，甚至根本无法并发执行。

(3) 虚拟性

操作系统向用户提供了比直接使用裸机要简单方便得多的高级抽象服务方式，从而对用户隐藏了对硬件操作的复杂性。在操作系统中，虚拟是指把一个物理实体变为若干个逻辑上的对应物。物理实体是实实在在的，而后者是虚拟的，是在逻辑上存在的。在操作系统中利用了多种虚拟技术，分别实现虚拟处理机、虚拟内存和虚拟设备等。

(4) 不确定性

不确定性又称异步性，是指同样一个数据集的同一个程序在同样的计算机环境下运行，每次执行的顺序和所需的时间都不相同，即进程的执行顺序和执行时间具有不确定性。进程的执行是以异步方式进行的。每个进程在何时执行、多个进程间的执行顺序以及完成每道进程所需的时间都是不确定和不可预知的，进程是以人们不可预知的速度向前推进的。

注意 操作系统的不确定性不是指程序执行结果的不确定性。

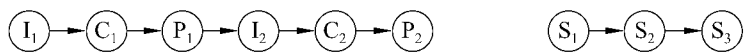
3.2 进 程 管 理

3.2.1 程序的顺序执行和并发执行

1. 程序的顺序执行和并发执行

(1) 程序的顺序执行

在多道程序设计出现以前，程序的最大特性是顺序性，即程序的顺序执行。一个程序通常由若干个程序段组成，它们必须按照某种先后次序执行，前一个操作执行完后，才能执行后继操作，这种计算过程即程序的顺序执行过程。如图 3-1(a)所示为程序的顺序执行示意图，其中，输入为 I，计算为 C，打印为 P。



(a) 程序的顺序执行示意图　　(b) 三条语句的顺序执行示意图

图 3-1　程序的顺序执行示意图

【实例 3-1】 在进行计算时，总须先输入用户的程序和数据，然后进行计算，最后才能打印计算结果。如图 3-1(b)所示为三条语句的顺序执行示意图。

解：S_1：　　a：=x+y;

S_2：　　b：=a−5;

S_3：　　c：=b+1;

(2) 程序顺序执行时的特性

① 顺序性：程序的执行是按照程序结构所指定的次序进行的，可能的次序有分支、循环或跳转等。

② 封闭性：程序在执行过程中独占全部资源，计算机的状态完全由该程序的控制逻辑所决定。

③ 可再现性：只要程序执行的初始条件相同，执行结果就完全相同。例如，在程序中可利用空指令控制时间关系。

2. 程序的并发执行

如果每台计算机在任一时刻只处理一个具有独立功能的程序，那么操作系统的设计和功能都将变得非常简单，因为在这样的系统中不存在资源共享和程序的并发执行以及用户执行的随机性问题。但是，在很多情况下，需要计算机能够同时处理多个具有独立功能的程序，批处理系统、分时系统、实时系统以及网络与分布式系统等都是这样的系统。

程序的并发执行是指若干个程序(或程序段)同时在系统中运行，这些程序(或程序段)的执行在时间上是重叠的，一个程序(或程序段)的执行尚未结束时，另一个程序(或程序段)的执行可能已经开始。如图 3-2 所示为程序的并发执行示意图。

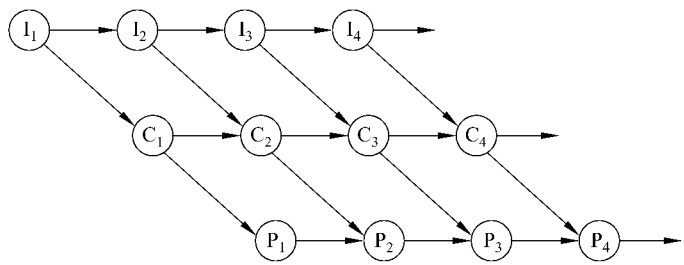


图 3-2 程序的并发执行示意图

程序的并发执行提高了系统的吞吐量，也产生了一些与顺序执行不同的新特点。

(1) 间断(异步)性：处理器交替执行多个程序，每个程序都是以“走走停停”的方式执行，可能走到中途停下来，而且程序无法预知每次执行和暂停的时间长度，从而失去了原有的时序关系。

(2) 失去封闭性：由于多个程序共享一个计算机系统的多种资源，因此每个程序的执行都会受其他程序的控制逻辑的影响，例如，一个程序写到存储器中的数据可能被另一个程序修改，失去原有的数据不变特征。

(3) 失去可再现性：由于程序执行环境的封闭性不再成立，因此程序每次执行的环境可能会不同，执行环境在程序的两次执行期间发生变化导致执行结果的不同，程序的执行结果失去原有的可重复特征。

【实例 3-2】 有两个循环程序 A 和 B，它们共享一个变量 N。程序 A 每执行一次时，都要做 N：=N+1 操作；程序 B 每执行一次时，都要执行 Print(N)操作，然后再将 N 置成“0”。程序 A 和 B 以不同的速度运行。

解：(1) N：=N+1 在 Print(N)和 N：=0 之前，此时得到的 N 值分别为 $n+1$、$n+1$、0。

(2) N：=N+1 在 Print(N)和 N：=0 之后，此时得到的 N 值分别为 n、0、1。

(3) N：=N+1 在 Print(N)和 N：=0 之间，此时得到的 N 值分别为 n、$n+1$、0。

3.2.2 进程的定义与特征

程序在顺序执行时，具有顺序性、封闭性和可再现性。但为了提高计算机资源的利用率

和增强系统的处理能力而引入了硬件并行操作，这可使某些程序并发执行，所谓并发是指在同一时间间隔内有若干事件发生。程序的并发执行不仅能提高系统的吞吐量，而且可显著地改善资源的利用率，所以并发已成为现代操作系统的一个基本特征。但是，程序的并发执行，使之失去了顺序程序的封闭性和可再现性，程序与计算不再一一对应，并产生了相互制约。为了描述程序的并发执行而引入了进程的概念。

1. 进程的定义

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，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。

进程是操作系统结构的基础，是一个正在执行的程序，计算机中正在运行的程序实例，可以分配给处理器并由处理器执行的一个实体，由单一顺序的执行显示、一个当前状态和一组相关的系统资源所描述的活动单元。

提示 进程为应用程序的运行实例，是应用程序的一次动态执行。可以简单地理解为：它是操作系统当前运行的执行程序。在系统当前运行的执行程序里包括：系统管理计算机个体和完成各种操作所必需的程序；用户开启、执行的额外程序，当然也包括用户不知道而自动运行的非法程序(它们就有可能是病毒程序)。

2. 进程的特征

(1) 动态性：进程的实质是程序的一次执行过程，进程是动态产生，动态消亡的。

(2) 并发性：任何进程都可以同其他进程一起并发执行。

(3) 独立性：进程是一个能独立运行的基本单位，同时也是系统分配资源和调度的独立单位。

(4) 异步性：由于进程间的相互制约，使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。

(5) 结构特征：进程由程序、数据和进程控制块三部分组成。

3. 进程的结构

进程由程序段、数据段和进程控制块(Process Control Block，PCB)这 3 个部分组成。

(1) 程序段是进程中能被进程调度程序在 CPU 上执行的程序代码段，通过程序段的执行可以实现程序的特定功能。

(2) 数据段可以是进程对应的程序加工处理的原始数据，也可以是程序执行后产生的中间或最终结果数据。

(3) 进程控制块是进程实体的一部分，是操作系统中重要的记录型数据结构，其中存放了操作系统所需的、用于描述进程情况和控制进程运行所需的全部信息。

提示 当系统创建一个新进程时，系统为某个程序(包含数据段)设置一个 PCB，用于进行控制和管理，进程执行完成时，系统收回其 PCB，进程随之消亡。系统根据 PCB 感知相应进程的存在，故 PCB 是进程存在的唯一标志。

进程控制块的组成：进程标识符、处理机状态信息、进程调度信息、进程控制信息。

4. 进程的状态及其转换

在操作系统中，进程至少要有三种基本状态：执行状态、就绪状态和阻塞状态，如图 3-3 所示。

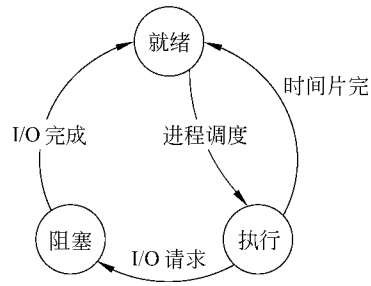


图 3-3 进程的基本状态及其转换

(1) 就绪状态。当进程已获得除处理机以外的所有资源(处理器被系统中的其他进程占用)，一旦分到了处理器即可立即执行时，则其处于就绪状态。

(2) 执行状态。当一个进程获得必要的资源，并占有处理器(在处理器上运行)时，则其处于执行状态。

(3) 阻塞状态。进程在执行过程中，由于发生某个事件而暂时无法执行下去时，就处于阻塞状态。

3.2.3 进程的互斥与同步

1. 临界资源

某段时间内只允许一个进程使用的资源称为临界资源。临界资源的访问遵循的准则。

(1) 当临界资源空闲时，允许一个(仅一个)请求进入临界区的进程立即进入其临界区，以有效地利用临界资源，且不至于使多个进程因相互阻塞而使彼此都不能进入临界区。

(2) 进程在临界区内停留有限的时间，对于要求访问临界资源的进程，应能在有限时间内进入自己的临界区，以免陷入死等状态。

(3) 当进程不能进入临界区时，应立即释放处理器，以免进程忙等。

提示 一次仅允许一个进程使用的共享资源。如打印机、内存单元、表格。

2. 临界区

在每个进程中访问临界资源的那段程序，称为临界区。

- 空闲让进：当无进程在互斥区时，任何有权使用互斥区的进程可进入。
- 忙则等待：不允许两个以上的进程同时进入互斥区。
- 有限等待：任何进入互斥区的要求应在有限的时间内得到满足。
- 让权等待：处于等待状态的进程应放弃占用 CPU，以使其他进程有机会得到 CPU 的使用权。

3. 互斥的定义

在计算机中有许多资源一次只能允许一个进程使用。如果有多个进程同时去使用这类资源就会引起激烈的竞争，即互斥。

4. 同步的定义

把这种进程间为了完成一个共同的目标，必须互相合作的协同工作关系、有前后次序的

直接制约关系称为进程的同步。

进程的同步与互斥的不同之处如下所示。

(1) 互斥的各个进程在各自单独执行时都可以得到正确的运行结果，但是当它们在临界区内交叉执行时就可能出现问题。而同步的各个进程，如果各自单独执行将不会完成作业的特定任务，只有当它们互相配合、共同协调推进时才能得到正确的运行结果。

(2) 互斥的进程只要求它们不能同时进入临界区，而不需规定进程进入临界区的顺序。但同步的进程的协调关系是建立在它们之间执行顺序的基础上，所以，各个进程必须按照严格的先后次序执行。

(3) 一般情况下，互斥的进程并不知道对方的存在。而同步的进程不仅知道其他进程的存在，还要通过与其他进程的通信来达到相互的协调。

5. 信号量

一般来说，信号量的值与相应资源的使用情况有关，信号量的值仅由 P、V 操作改变。P、V 操作都是原语，P 表示申请一个单位资源，V 表示释放一个单位资源。

P 操作 P(s)：取 s 值减 1，若 s<0，入等待队列；若 s≥0，继续。

V 操作 V(s)：取 s 值加 1，若 s≤0，唤醒一等待队列进程；若 s>0，继续。

提示 s 为信号量。

【实例 3-3】 用 P、V 原语实现互斥问题。进程 A 和进程 B 共享一台打印机，如果系统已经把打印机分配给了进程 A，则进程 B 因不能获得打印机的使用权而应处于等待状态；只有当进程 A 把打印机释放后，系统才能唤醒进程 B 使其获得打印机的使用权。如果这两个进程不这样做，那么也会造成极大的混乱。互斥信号量 mutex(初值为 1)，Pa 为分配进程，Pb 为释放进程。

解：

Pa：

```
…
P(mutex)
分配打印机
(读写分配表)
V(mutex)
   …
```

Pb：

```
…
P(mutex)
释放打印机
(读写分配表)
V(mutex)
   …
```

【实例 3-4】 有进程 A 和进程 B 两个进程，它们需要通过访问同一个数据缓冲区合作完成一项工作，进程 A 负责向缓冲区写入数据，进程 B 负责从缓冲区读取该数据。显然，当进程 A 还未向缓冲区写入数据时(缓冲区为空时)，进程 B 因不能从缓冲区得到有效数据而应处于等待状态；只有等进程 A 向缓冲区写入了数据后，才应通知进程 B 去取数据。相反，当缓冲区的数据还未被进程 B 读取时(缓冲区为满时)，进程 A 就不能向缓冲区写入新的数据而应处于等待状态；只有等进程 B 自缓冲区读取数据后，才应通知进程 A 去写入数据。显然，如果这两个进程不能如此协调工作，则势必造成严重的后果。信号量：s_1 表示缓冲区空否(初值为 1)，s_2 表示缓冲区满否(初值为 0)。

解：

供者进程：

```
L1: P(s1)
启动读卡机
    ...
收到输入结束中断
V(s2)
goto L1
```

用者进程：

```
L2: P(s2)
从缓冲区取出信息
    ...
V(s1)
goto L2
```

6. 经典的进程同步问题

【实例 3-5】 生产者—消费者问题。设生产者进程不断地生产产品，消费者进程不断地消费产品。生产者进程和消费者进程通过一个有界的缓冲区联系起来。

解：(1) 一个生产者和一个消费者公用一个缓冲区的问题分析。

解决办法，定义两个信号量：empty 表示缓冲区是否为空，初值为 1，生产者用它来判断缓冲区是否可写；full 表示缓冲区是否为满，初值为 0，消费者用它来判断缓冲区是否可读。

producer(生产者)的伪码：

```
while(1)
{
    P(empty);
    写缓冲区;
    V(full);
}
```

consumer(消费者)的伪码：

```
while(1)
{
    P(full);
    读缓冲区;
    V(empty);
}
```

(2) 一个生产者和一个消费者公用 m 个环形缓冲区的问题分析。分析过程与第一种情况类似，直接看伪码。

producer(生产者)的伪码：

```
while(1)
{
    P(empty);     /* empty 初值为 m */
    写第 in 个缓冲区;
    /* in 用来指示当前的第一个可写的
       缓冲区的下标,初值设为 0 */
    in=(in+1)%m;
    V(full);
}
```

consumer(消费者)的伪码：

```
while(1)
{
    P(full);     /* full 初值为 0 */
    读第 out 个缓冲区;
    /* out 用来指示当前的第一个可读的
       缓冲区的下标,初值设为 0 */
    out=(out+1)%m;
    V(empty);
}
```

(3) 一组生产者和一组消费者公用 m 个环形缓冲区的问题分析。

相比第 2 种情况，所要做的是用两个互斥变量 mutex_producer 和 mutex_consumer，来实现各生产者间、各消费者间互斥地访问某个缓冲区。

producer(生产者)的伪码：

```
while(1)
{
    P(empty);     /* empty 初值为 m */
    P(mutex_producer);
```

consumer(消费者)的伪码：

```
while(1)
{
    P(full);     /* full 初值为 0 */
    P(mutex_consumer);
```