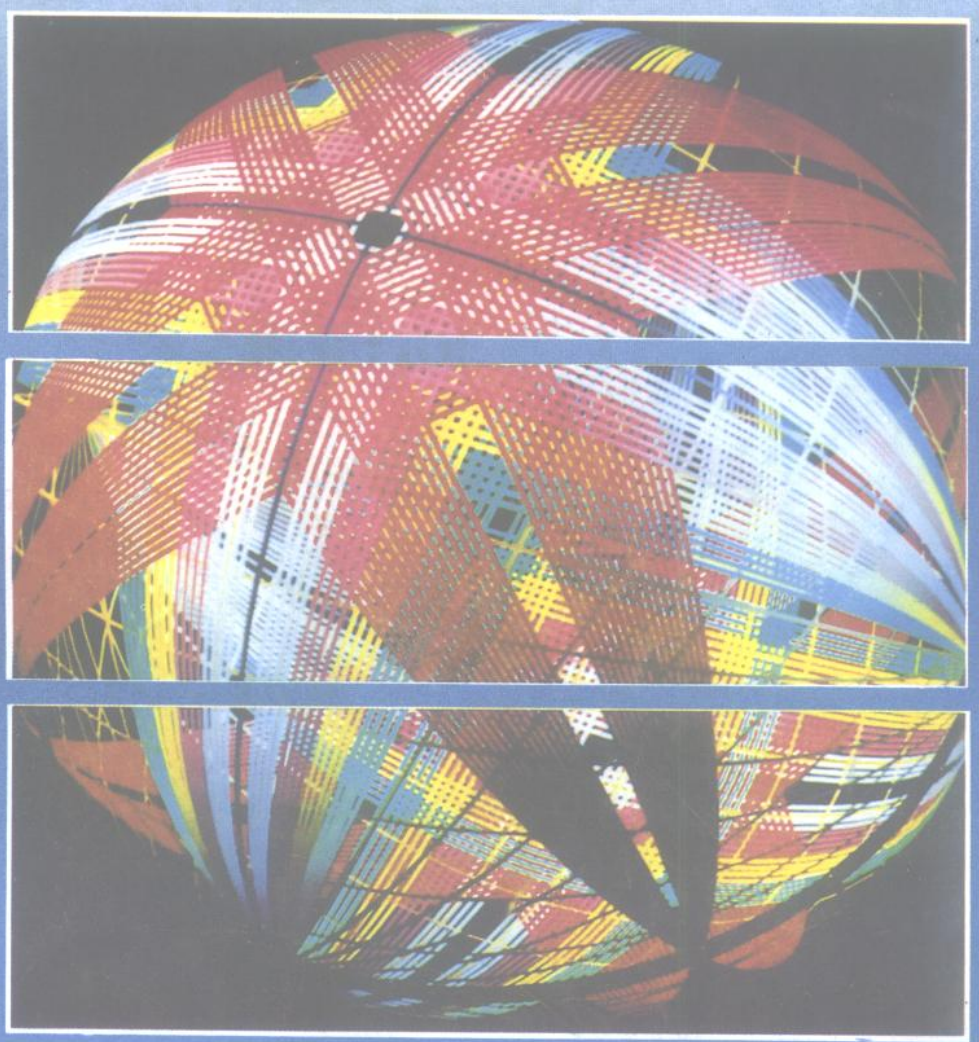
VAX/VMS系统调用方法及程序设计

夏 凌 董京春 胡凌美 编著



科学出版社

(京) 新登字 092 号

内 容 简 介

本书全面深入地介绍了 VAX／VMS 系统的调用方法和程序设计方法，是作者从事 VMS 系统的应用与开发的经验总结．全书共七章，内容包括：VMS 操作系统概述，本机方式下外部过程调用方法，进程控制，进程同步，进程通讯，利用 VAX／VMS 系统功能进行终端的输入输出，在 VAX／VMS 网络环境上设计多进程实时控制系统．书中介绍的大量程序设计实例，可被直接引用，对有关人员很有参考价值．

本书理论与实践相结合，注重实用，既可供科技人员深入了解、使用 VMS 系统和进行计算机应用开发时参考，也可作为教学参考书供高等院校有关专业师生阅读．

VAX／VMS 系统调用方法及程序设计

夏 凌 董京春 胡凌美 编著

责任编辑 孙桂荣

科学出版社 出版

北京东黄城根北街 16 号

邮政编码：100717

辽宁省科学技术情报研究所印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1994 年 3 月 第 一 版 开本：787×1092 1／16

1994 年 3 月 沈阳第一次印刷 印张：9

印数：0001—4 000 字数：219000

ISBN 7-03-003974-2／TP・218

定 价：8.70 元

前　言

VAX／VMS 是 Digital 公司在 VAX 系列计算机上配置的通用的多用户、多功能操作系统。它是当今世界上最具有代表性的操作系统之一。随着时间的推移，系统功能得到了不断扩充与完善。如何调用 VAX／VMS 的系统过程，利用系统提供的各种功能进行应用程序的开发，是广大 VAX 用户以及对 VMS 系统感兴趣的读者所关心的问题。因此，广大读者迫切需要一本综合论述这方面内容的参考书。

本书深入浅出，注重实用，力求将有关系统调用问题讲深讲透，以适合国内读者的阅读习惯和使用要求。每一部分都列举了一定实例，介绍应用的方法。书中的程序例子均已经过实际考验，可作为读者应用的借鉴。

本书共分七章。第一章是 VMS 操作系统概述。在后续章节中，出现了许多有关 VMS 的概念，因此对这些概念进行概貌性描述是必要的，但对熟悉 VMS 概念的读者则可跳过这一章。第二章介绍本机方式下外部过程的一般调用方法，主要介绍 VAX／VMS 过程调用标准及各种语言之间的相互调用。第三章介绍进程控制，即进程如何创建，如何控制、管理和改变进程的各种状态。第四章介绍进程同步问题。第五章介绍进程通讯问题。第三、第四、第五章的内容对从事实时应用的读者会有很大帮助。第六章介绍利用 VAX／VMS 系统功能组织终端的输入输出。其中屏幕管理技术如多窗口技术，对键盘随机输入的实时响应等等。在实际应用中显得非常重要。第七章结合前面各章内容介绍多进程实时控制模拟系统的设计方法。这一章结合作者的工作实践，对利用 VMS 特性来设计多进程实时控制模拟系统的方法进行了深入探讨。

本书可作为科技人员的参考书，也可供高等院校有关专业的师生使用。

本书由夏凌执笔，董京春、胡凌美参加了部分编写工作。最后由夏凌统稿。中国科学院沈阳计算技术研究所赵国泰研究员、大连理工大学郜荣春副教授审阅了本书稿。同时，叶天荣同志也审阅了全稿并提出了许多修改意见。在此一并表示衷心感谢。

由于时间仓促，作者学识浅薄，书中错误与不足在所难免，敬请批评指正。

作　者

目　录

第一章　VMS 操作系统概述

本书主要介绍 VMS 系统调用方法以及如何使用 VMS 系统功能进行程序设计。在介绍这些内容时，还将涉及到 VMS 操作系统的许多概念。这对初学者概貌了解 VMS 系统无疑是有帮助的。这一章主要介绍 VMS 操作系统中的几个重要问题：多道程序设计；进程；虚拟存贮；调度与交换；系统服务。

1.1　多道程序设计

1.1.1　程序和映象

程序是用计算机语言写成的指令的逻辑序列，它告诉计算机怎样解决一个问题。源程序必须转换成机器代码，然后转换成一个机器可执行的映象(见图 1.1)。

图 1.1　程序和映象

程序和映象是同一事物的不同阶段。在程序阶段着重点是逻辑结构，即算法。在映象阶段，着重点是机器的可执行形式。

1.1.2　程序状态

当我们在计算机上运行一个映象时，映象可处于几种可能的状态：

(1) 执行状态。如果映象现在处于处理机的控制下，那就是说处理机正在执行映象中的一条指令。除非有多个处理机，否则在同一时刻，只有一个映象可处于执行状态。

(2) 等待状态。由于等待一个事件，如等待与某些外围设备进行的 I／O 通讯的完成，致使映象在此时刻不能执行，则映象处于此状态。

(3) 就绪状态。映象处于可执行状态，但实际没有执行。

如图 1.2，既有状态(圆圈)也有转换(箭头)。这是因为某个事件出现，将引起映象从一种状态经过转换变成另一种状态。

转移	事件
a	I／O 请求
b	I／O 完成
c	为执行另一映象而交出 CPU

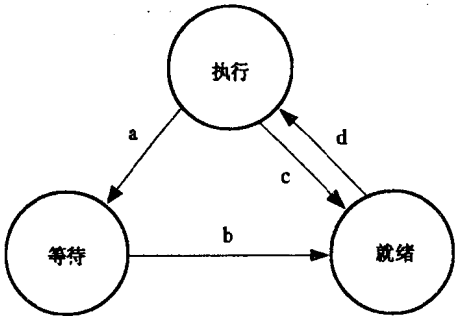


图 1.2 状态和转移

d 调度执行

执行状态是唯一的(除非有两个以上的处理机)，因为在任何时间只能有一个映象处于执行状态，在多道程序设计系统中，等待状态不是唯一的，有两个原因:

(1) 在任何给定的时间里，通常有两个以上的映象等待某一事件。

(2) 存在多个等待状态，决定于映象正在等待哪一个事件。

在不同的操作系统，各个等待状态是不同的。但是通常有许多状态决定于映象正在等待的事件的性质。同样，就绪状态不是唯一的(在 1.4 节中有进一步说明)。总之，在多道程序系统中，有三种状态，映象可能处于它们中的任何一种状态。映象可能正在执行，或者可能准备执行，或者可能正等待某一事件的完成，该事件可使映象处于可执行状态。

1.1.3 计算类和 I／O 类程序

如果考虑程序是做大量的计算工作，还是做大量的输入／输出处理，我们就能更进一步讨论程序状态。计算类和 I／O 类程序可由图 1.3 予以说明。

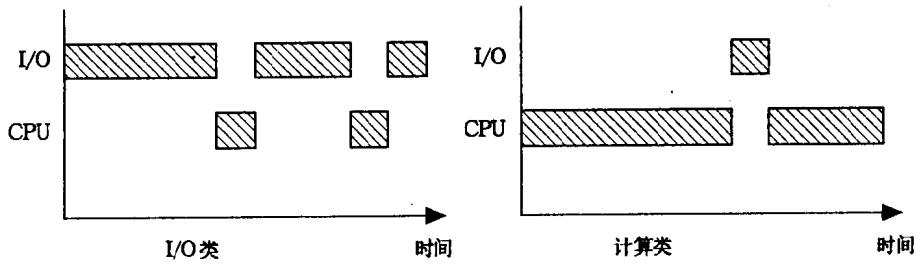


图 1.3 I／O 类和计算类程序

例如，一个计算 π 到小数点后一百万位的程序就是一个计算类型的程序，因为每计算一位都要花费一定时间，而最后只输出一串数字到外存。与此相应地，一个联机的飞机订票系统就是输入输出类型。订票系统花费了大量的时间等待，既等待来自一个职员在终端上的请求，也等待来自辅存上的文件。

在多道程序系统中，通常也有两类程序的混合，多道程序设计的一个目标是让系统中的所有程序尽可能地忙，使得这两种类型的程序都得到服务。

作为一个简单例子，让我们来看看两个程序。一个I/O类，一个计算类。如果不论何时，I/O类程序能执行我们就让它执行的话，会发生什么呢？I/O类程序将运行一会，然后进入等待状态。而计算类程序与I/O类程序相比很少受到小的延迟的影响。因为计算类程序通常是或者处于执行状态，或者处于就绪状态。如果让I/O类程序在"一般的"状态下运行，它不仅要等待I/O完成，而且必须要等待计算类的程序。I/O类程序的运行通常有很多小的时间间隔，所以它的性能会严重地受到影响。如果给它一个好处，即无论何时，只要准备好了就让它执行，它的性能会得到改进。对于计算类的程序则是无关紧要的。VAX/VMS虚拟存贮系统使用这种方法帮助I/O类程序，将在1.4节调度与交换中作进一步讨论。

1.1.4 多道程序设计

所谓多道程序设计，就是允许有两个以上的程序同时驻留内存。这种方法可使得正在执行的程序进入等待状态时另一个程序可投入执行。从而减少了处理机的空闲时间(见图1.4)。

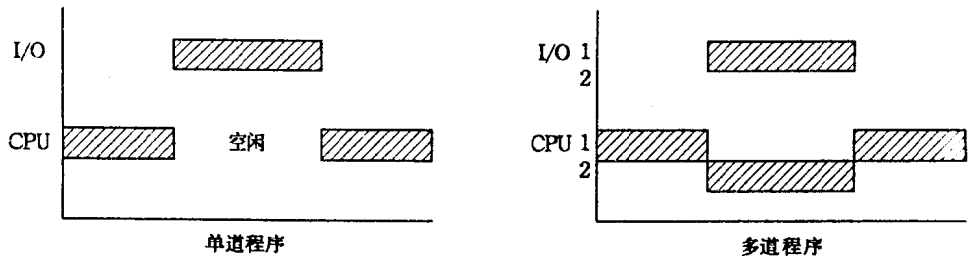


图1.4 单道程序与多道程序的比较

此外，多道程序设计通常能发挥较高的内存效率。在一个单道程序设计系统中，通常一个用户映象不能完全占据存贮器，而几个用户映象同时占据内存就会充分利用存贮资源，由于增加了内存利用率，使得多道程序操作系统必须做更复杂的事情。

尤其是：

(1) 哪些程序驻留内存?

(2) 紧接着执行哪一个驻留程序?

这两个问题在单道程序系统中都不会发生。一个程序驻留在内存，不论何时它都可以执行。由于设计一个较大的操作系统，其复杂性是一次性花费，而较大的处理机效率的改善是一个连续过程，因此大多数操作系统使用多道程序设计。须注意的是"单道程序"不同于"单用户"。因为有些操作系统只允许一个终端有一个用户，但它允许在同一时刻一个用户可执行两个以上的程序。例如，有些单用户系统允许用户运行一个高优先级的程序，而在同一时刻，只要高优先级的程序没有执行，则运行一个低优先级的程序。

1.1.5 交换

交换是指在映象的执行期间，在操作系统的控制下，内存与外存之间映象的换进换

出。当一个映象输到内存，我们就说它交换进来。同样，当映象离开内存，我们就说它交换出去。交换是在有限的内存资源与多个用户程序对较大的存贮空间的要求之间进行动态平衡。交换的优点在于它的灵活性。操作系统能动态地调整由各个用户程序提出的处于变化中的要求。交换程序部分地回答了这个问题，即"哪些程序将驻留内存?"交换的使用使得"所有的程序将占据内存一部分时间，某些程序可能驻留整个时间。"精确地说，哪些程序什么时间将驻留内存，取决于实现与设计的考虑。例如，在分时系统中，当程序正在等待终端用户输入时，它可能会被交换出去。或者在象 VAX／VMS 这样的有优先级的系统中，低优先级程序可能被交换出去以腾出空间给高优先级程序。

然而，不是所有的多道程序设计系统都使用交换。例如，在一个初级批处理多道程序设计系统中，它可能足以将内存划分为几个固定的区域，根据优先级或某一类似的考虑，这些区域将装入等待用户程序，一直到没有更多的区域空闲。一旦装进一个区域，程序就占据它直到结束。当然，只有当区域是空闲的时候，程序才能得到一个区域，这由操作系统来分配。这个过程是极其简单且容易实现的，操作系统简单地决定哪一个非驻留程序得到下一个空闲区域。但问题是这样简单地安排完全不能处理一个复杂快速的变化环境。一个服务于分时和实时应用的系统必须以一种灵活的方式响应。把内存资源无限期地交给专用程序，对于操作系统作出快速和灵活地响应没有帮助。

VAX／VMS 中使用了交换，这将在 1.4 节中详细介绍。

1.1.6 调度

在多道程序设计中通常使用以下两个处理机管理策略:

(1) 在优先级调度中，每个映象处于一定的优先级。

任何时刻，选择可执行的最高优先级映象投入执行，并且直到下列事件完成才停止执行:

① 执行完毕;

② 由于 I／O 请求交出处理机;

③ 一个更高的优先级程序进来投入运行。

(2) 时间片轮转法调度，给每一个映象固定的时间量执行，映象只要下列情况出现就停止执行。

① 执行完毕;

② 映象交出了处理机;

③ 用完了整个时间片。

时间片是轮转服务的一种重要方法。没有时间片也有轮转调度。例如，有一个等待队列，当运行中的程序交出 CPU 时，程序就放到队尾。用此方法，没有使用时间片，也得到了轮转的效果。但是这种轮转法不适合于既有计算类程序，又有 I／O 类程序的情形。因为计算类程序一般要运行较长的时间才交出 CPU。正由于这个原因，时间片才常用于保证有更多的相等的轮转服务。

上述两种调度方法既能用于批处理，又能用于交互处理。批处理包括在某一时刻提交一个完整的工作，通常叫做一个作业。一旦用户提交了作业，用户就失去了对它的控制，直到返回一个结果，一个例外就是用户通常能在执行中取消这个作业。有了交互处理，用

户就能处在一个连续的与计算机交互作用的环境中。响应时间(或周转时间)通常在批处理时需数小时甚至几天的时间，而在交互作业处理中只需几秒或几分钟时间。

在 VAX／VMS 系统中，使用轮转法和优先数调度的组合。尽管它们是完全不同的调度方法，组合起来后，每一种方法都面对着一个特殊类型的应用。有了轮转法调度，事件常常是一个来自时钟的时间片已到期的信号。有了优先级调度，较高优先级的映象就可准备执行。

(1) 轮转法调度用于分时，保证每个用户经常得到服务；

(2) 优先级调度用于实时应用，在这种应用中，如果系统不立刻响应某一事件就会出现较严重的情况。因此必须保证实时响应。

1.2 进程

1.2.1 VMS 中关于进程的描述

在 VAX／VMS 中，基本的程序实体叫做进程。这里，进程是指的整个环境，它包括：硬件上下文、软件上下文(合在一起称作上下文环境)和虚地址空间(见图 1.5)。

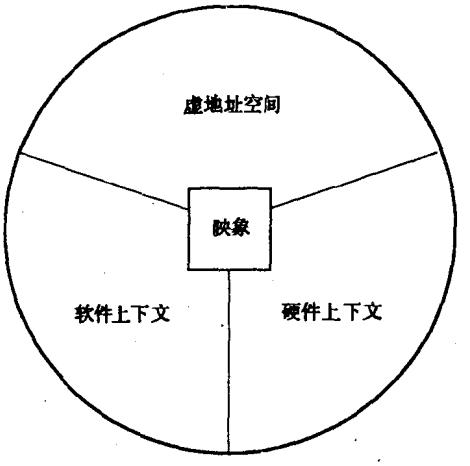


图 1.5 进程

映象正是在这样的环境中得到执行。这里需要特别说明一点，VAX／VMS 调度和交换的是进程，而不是映象。

1.2.2 硬件上下文与软件上下文

硬件上下文(hardware context)是进程的一部分。简单地说，就是处理机寄存器的内容。当一个进程正在执行时，它的硬件上下文存到处理机寄存器中。当该进程不再执行时，它的硬件环境移出处理机并存入内存或辅存中，这可出现在任何多道程序设计系统中。通过保存硬件寄存器的内容，系统能允许另一个进程执行，后面的进程保存原进程的硬件上下文环境。用此方法，原进程在它被中断的地方可以继续执行。这些都随着时间而

不断变化。硬件寄存器(有时称做处理机状态)保存进程中断时的状态。让我们看看保存在硬件上下文中的寄存器:

(1) PSL(处理机状态长字),包含各种各样的信息块,象条件码,自陷位,存取方式和中断处理信息。

(2) PC(程序计数器),实际是通用寄存器 R_{15},PC 指向了下一个要执行的指令。

(3) 通用寄存器 R_0—R_{14},这些寄存器中有的有特殊用途,如 R_{14} 用作堆栈指针(SP)。

(4) 其他特殊用途的寄存器。

总而言之,硬件上下文就是处理机寄存器的内容。在进程执行期间存贮在处理机中,当进程不执行时,存贮到主存或辅存中。决定一个进程停止执行而另一个进程开始执行的交换上下文的操作,对所有的多道程序设计系统是通用的。

软件上下文环境类似于硬件上下文环境。它描述的是进程的软件状态。

软件上下文包括进程的标识信息、计帐、优先数、特权以及份额等等。这些软件状态变量定义了一个进程能做什么,不能做什么。例如,软件上下文定义了多少内存空间可供使用,多少处理机时间以及一个进程可以访问哪些设备。如 PSL 有一个域,它定义了一个进程能执行哪些指令。同样,有些软件参数根据软件特权定义了一个映象能做什么。

1.2.3 虚地址空间

映象在某个环境中执行的一个重要方面是虚地址空间。虚地址空间包含有由映象访问的虚地址。VAX / VMS 给进程提供了 2^{32} 或稍小于 4 300 000 000 字节的虚存贮空间。地址空间有相当一部分由系统和进程控制信息所占有,所以并不是所有的 4 千 3 百兆字节都能使用。只有 1 千兆的空间可用于存贮用户的映象和数据。不过在实际应用中,多于 1 千兆字节的应用程序还是少见的。一部分虚地址空间(比一千兆通常要小得多)实际上是由映象定义的。这个虚地址范围通过映象机构与物理地址相关联。在 1.3 节中将更详细地讨论虚地址空间。

1.2.4 进程和作业

正如前面所讲的,在 VAX / VMS 上按用户的要求,进程是可调度的程序实体。从另一方面看,作业是一个进程簇,它们共享一个公共资源分配。换句话讲,在 VAX / VMS 里,作业是一个计算的实体。

进程可以产生子进程,以完成某些专门功能。同样一个子进程也可以产生它自己的子进程。这就有一个主进程和子进程的层次。进程在某时刻既可能是一个主进程也可能是一个子进程。也就是,一个进程可能是它的子进程的所有者,也可能同时是另一进程的子进程。

不管何时,一个主进程终止时,它的所有子进程也被迫终止。没有主进程的进程叫做独立进程。刚登录的用户就是一个独立进程。为了回答登录请求,系统产生一个独立进程去完成用户的工作。如果一个独立进程产生子进程,那么它就成为主进程。但不管它是否产生子进程,它仍为独立进程。

如图 1.6 所示,VAX / VMS 中的一个作业是独立进程加上它所产生的任何层次的子

进程。一个作业的所有进程当分别请求占有 CPU 时间时，共享一个通用的资源分配。将一个进程作为调度单元而一个作业作为计算单元。当一个进程产生一个子进程时，主进程必须给子进程一小部分资源份额和限额。依次地，如果子进程产生它自己的子进程，新的主进程必须转让它的一小部分资源给新的子进程。这样，一个作业的所有进程共享最初分配给独立进程的资源。

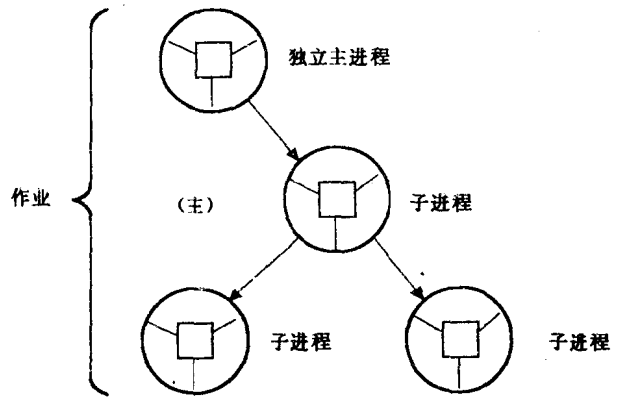


图 1.6 进程和作业

VMS 仅仅是一个能产生独立进程的实体；而对一个进程来说，它可以以合适的特权去产生一个独立的进程。在这种情况下，将自己最初的资源分配给独立进程。当第一个独立进程终止时，第二个独立进程并不被迫终止。事实上，创建一个独立进程就象用户第二次登录一样，除非第二个独立进程不与终端相连。然而，创建一个独立进程并不常用。一般一个进程没有足够的特权去这样做。这些将在以后的章节中给予详细说明。

1.3 虚拟存贮

1.3.1 虚地址

VMS 操作系统是虚拟存贮系统。VMS 是 Virtual Memory System 的缩写。现在我们就来谈谈"虚拟地址存贮"这种复杂而又功能很强的存贮管理技术。

虚拟存贮是一种将主存与辅存结合起来的存贮管理技术，它给程序员一个非常大的可用主存的假象。尽管虚拟存贮实现起来很复杂，但如果掌握了它的基本用途，是易于理解的。

对程序人员来讲，虚存的主要意思是让操作系统去关心许多存贮管理的细节。程序人员可以在一个极大的符号化的虚地址空间里编程序，系统负责对每一个符号地址进行硬件分配。

以上所讲关键在于"符号地址"。这里所讲的"符号"不象汇编语言程序设计中的符号地址那样精确。相反，我们在一种较不明确的意义上使用这个词。由映象访问的虚地址不是一个在内存上的实物理地址，或者一个辅存中的块。它是硬件定位中的一个符号的或逻辑的表示。

物理定位对程序人员来讲是不可见的。操作系统对由映象定义的虚地址空间的每一个单元安排一个内存或外存的单元。实存的字节定位在哪里，只有操作系统知道。不象物理地址，总是访问同一位置。虚拟地址是象征性的(符号的)，也就是说，它们首先转换成物理地址，然后才能取指令或操作数。

从以上我们可以看到，虚拟存贮有两种地址，虚拟地址和物理地址。虚拟地址可以看作是一个巨大的线性数组排列的字节序列，也可以想象成一个巨大的内存。物理地址是由处理机访问的主存中的实际硬件地址。由于实际物理地址远小于虚拟地址，操作系统通过使用辅存弥补了这种差距。

1.3.2 映射

虚存的秘密在于操作系统能保持对硬件位置的跟踪，这个硬件位置存贮着特殊虚拟地址的内容。这是由映射将虚地址空间转换到物理存贮介质(如主存或辅存)来完成的。任何时候，映射信息都使得操作系统能将正确的物理位置对应到给出的虚拟地址。

在 VAX／VMS 中，将地址组合成固定大小为 512 个字节的页。使用页的一个原因在于辅存设备，如磁盘是块结构的。在辅存(VAX 中的磁盘)与主存之间是整块的交换。第二个使用固定大小页的原因也是更强的原因，在于映射的实际考虑。

映射是通过使用页表来实现的。这就意味着对每一个虚页都有一个页表项。VMS 映射虚页到主存中的实页帧号，或辅存中的物理页块，见图 1.7。

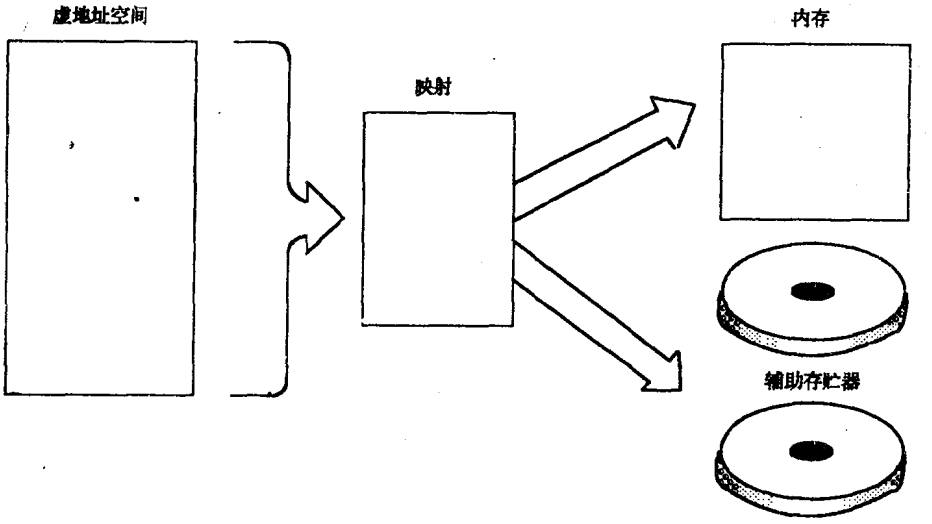


图 1.7 映射

在主存中进程可以使用的页的集合称做工作集(见图 1.8)。

请记住，工作集是由有效页组成的，进程正是在这些页上进行工作。如果在一个进程的环境中执行的映象访问一个存贮在工作集中页的地址，那么这个地址访问就称为有效。否则，称做无效，或发生页故障。一个页是否在工作集中，由该页的页表项中的某一位来表示。每一个虚地址空间的页在表中有一项，它描述了那个页的状态和位置。页表项的高位表示该页是否在工作集中。这一位叫做页有效位。因为它表明了访问这一项是不是有效。

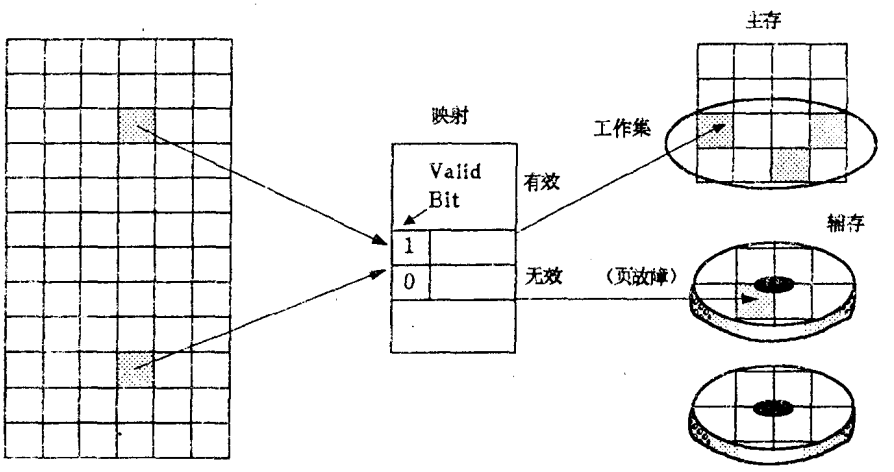


图 1.8 工作集

在页表中有两种使用映射信息的机构。

(1) 地址转换是一个硬件机构。如果所访问的页在工作集中，该机构将一个虚地址转换成一个物理地址。如果该页不在工作集中，转换无效，页故障产生。

(2) 软件调页机构用于响应页故障。调页使用映射以决定虚地址定位在工作集以外的什么地方，然后将该页调到工作集。

由此可见，地址转换和调页是互补机构。如果转换是有效的，硬件就到由虚地址转换的地方去取物理地址的内容。如果转换无效，由调页程序将该页传送到工作集。只有这时再进行地址转换才是有效的。

1.3.3 地址转换

地址转换是要找到正确的物理地址，如果映射指明了正确的页，硬件怎样去找正确的字节呢?

无论是对虚地址还是实地址，页内字节是一样的。当硬件查看一个虚地址时，并不是"从虚地址 φ 开始的字节 α"，而是看"从虚页 V 开始的字节 B"。映射将告诉硬件虚页 V 存在存贮器的什么地方。在内存的那个页上找到字节 B，实际上就是访问虚页 V 上的虚字节。问题是这个页在哪里？这个精确信息是由页表提供的(见图 1.9)。

虚地址的低 9 位说明了所要访问的是该页的哪个字节。虚地址的下一个位域指明了所要访问的虚页。这个域叫做虚页号，或 VPN。VPN 唯一地指出了虚页，所以它也用于索引页表(图 1.9)。页表是一个线性数组，由虚页号连续排序，第一个虚页对应第一个页表项。第二个虚页对应下一个页表项，对于由映象定义的虚页范围的项都是如此。对于虚页 N，索引页表就是去找页表项。这样，翻译一个虚地址利用了虚地址的两个域：即字节偏移量和虚页号。

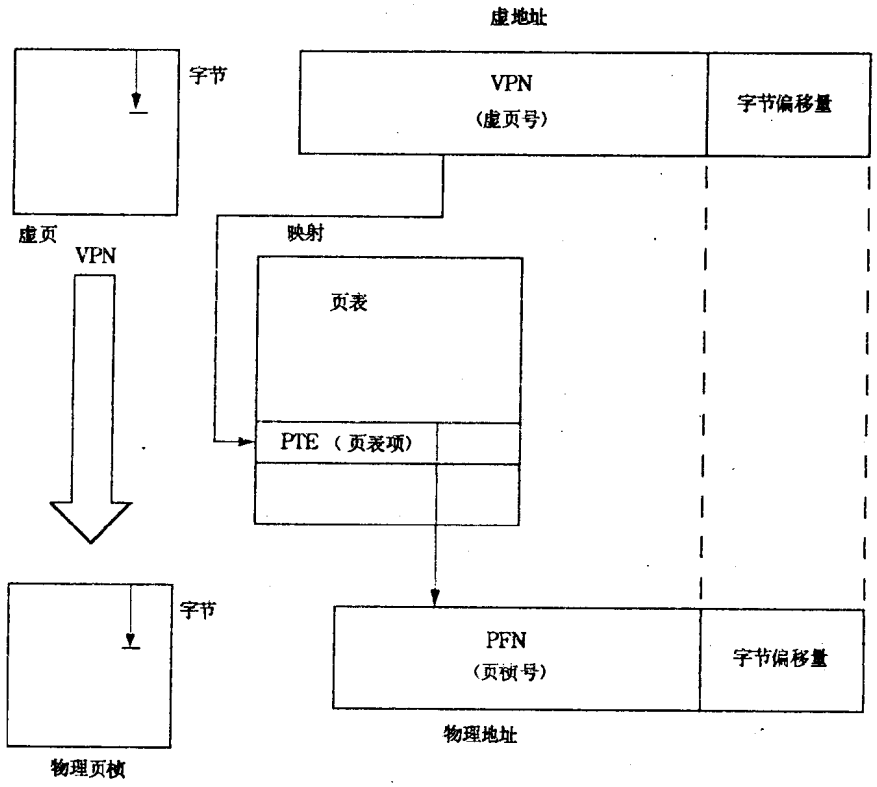


图 1.9 地址和页

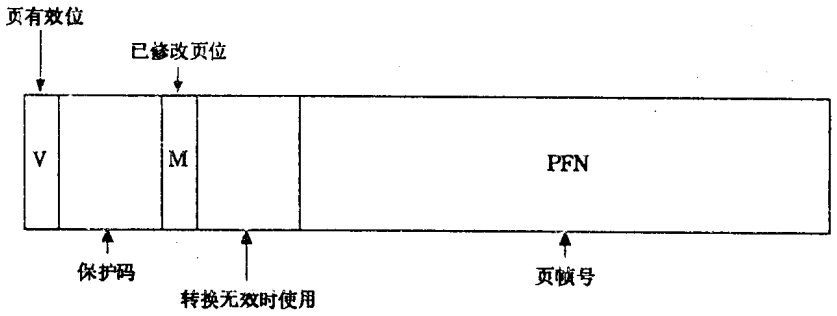


图 1.10 页表项

如果页有效位是 1，那么该页确实在工作集中．让我们再检查 PTE 的其他域．它们是：保护码(定义访问保护)，页修改位(指明是否已经改写过)，系统使用的域(页不在工作集时使用)以及页帧号(见图 1.10)。

地址转换可看作是一个黑盒子，它将虚地址在内存中变换成物理地址．在黑盒子里装的是映射机构，它将虚页号(VPN_s)转换成页帧号(PFN_s)．如果 VPN 能映射到 PFN，那么在页帧中的字节偏移量与虚页中的字节偏移量相同(见图 1.11)。

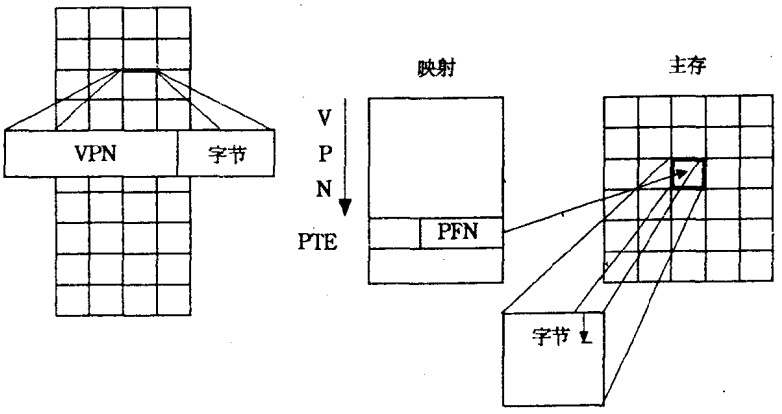


图 1.11　地址转换

1.3.4　调页

调页操作是由一个页故障请求的。而这个页故障是因为访问了不在工作集中的页引起的。如果所访问的页不在工作集中，那么它们可能在几个地方(见图 1.12):

(1) 在辅存上的原始映象文件里;

(2) 在内存里的自由页表上;

(3) 在内存里的已修改页表上;

(4) 在辅存的系统调页文件里。

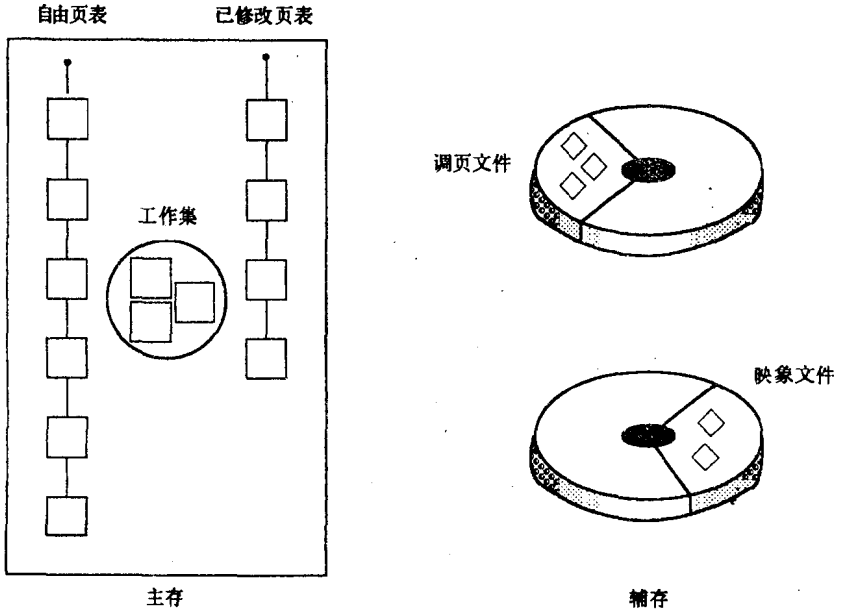


图 1.12　在内存中和辅存上的页