第3章　内存管理

3.1　408考纲要求

（一）内存管理基础

1. 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。

2. 连续分配管理方式

3. 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

1. 虚拟内存基本概念

2. 请求分页管理方式

3. 页面置换算法

最佳（OPT）置换算法；先进先出（FIFO）置换算法；最近最少使用（LRU）置换算法；时钟（CLOCK）置换算法。

4. 页面分配策略

5. 工作集

6. 抖动

3.2 知识结构梳理

3.2.1 内存管理基础

1. 内存管理概念

存储器是计算机系统的重要组成部分，所以存储器的管理是操作系统最主要的功能之一。在现代计算机体系结构中，由高速缓存、内存和外存构成了计算机系统的三级存储结构。操作系统原理的存储管理指的是内存管理。

存储管理的主要任务是管理内存资源，为多道程序运行提供有力的支撑。内存管理将根据用户需要给它分配存储器资源；尽可能地让内存中的多个用户进程实现存储资源的共享，以提高内存的利用率；能保护用户存放在内存中的信息不被破坏，要把诸用户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；由于物理内存容量有限，难于满足用户的需求，应该能从逻辑上来扩充内存，为用户提供一个比内存实际容量大得多的编程空间，方便用户的编程和使用。内存管理的四大功能：内存分配，内存保护，地址映射，虚拟内存。

按照经典的冯·诺依曼(von Neuman)计算机体系结构，CPU 执行程序、访问数据，都只能通过存储器进行。也就是说，进程用到的代码和数据都必须在内存中。

现代操作系统允许用户进程将程序放在内存空间的任意位置。因此，虽然计算机的地址空间从 0 开始，但用户程序的起始地址不必也是 0。这种组织方式为操作系统灵活使用地址空间创造了可能。

通常，程序员编写的是源程序，就是以某种高级语言或者汇编语言的语法、语义编制的文本文件，这些文本文件保存在磁盘的文件系统中。编译器将源文件转化成目标文件；链接工具将若干目标文件合并起来，转化成某种格式的可执行文件；装入工具将可执行文件装到内存，由操作系统指定其位置。在这些步骤中，地址有不同的表示形式。例如，源程序中的地址由符号表示(如 count)；编译器将符号地址转换成可重定位的地址(如从本模块开始的第 14 字节)；链接工具或装入工具将可重定位的地址转换成绝对地址(如 74014)。每次转换都是从一个地址空间到另一个地址空间的映射。

这样的地址转换，就是内存管理中的地址捆绑。将源程序的指令、数据捆绑到内存地址，可以在转换步骤的任何一步中执行。

(1) 编译时：如果在编译时就知道进程将在内存中的驻留地址，那么就可以生成之。

(2) 链接时：如果在链接时就知道进程将在内存中的驻留地址，那么也可以生成之。

(3) 装入时：如果在编译(链接)时并不知道进程将驻留在何处，那么编译器(链接工具)必须生成可重定位代码。这时，地址捆绑会在装入时进行。

(4) 执行时：如果进程在执行时可以从一个内存段移到另一个内存段，那么地址捆绑必须延迟到执行时才进行。绝大多数计算机及其上的操作系统，均采用这种方法。采用这种方法，需要特定硬件支持。

特别引起注意的是，无论源程序里面的地址，还是目标文件中的地址，还是可执行文件中的地址，都是逻辑地址；无论源程序里面的符号地址，还是后续的二进制码地址，都是逻辑地址。只有存在于地址总线，并且加载到物理内存条地址引脚上的地址，才是物理地址。众多形式的逻辑地址，只不过处于不同的逻辑地址空间。形式繁多的地址转换或地址捆绑，只不过是逻辑地址空间之间的相互映射。当然，最后当程序执行时，必须实现从逻辑地址到物理地址的映射。

为了多程序、多任务的要求，操作系统必须让多个进程同时驻留内存，并且保证各个进程有条不紊地运行。这是对操作系统内存管理方案的基本要求。每种方案的有效性取决于具体的应用需求。需考虑的因素主要包括以下内容。

(1) 硬件支持：每种方案的实施，往往都需要相应的硬件支持。也就是 CPU 中的存储管理部件(memory management unit，MMU)。从最简单的基址寄存器、界限寄存器，到复杂的联想寄存器、页地址映射、段地址映射等。

(2) 性能：内存管理方案的功能越强，实现起来就复杂，逻辑地址到物理地址的映射所需要的时间也越长，性能越差。如果映射不得不借助内存，那么性能就显著地降级。当然，转换后备缓冲区(translation look-aside buffers，TLB，也称联想存储器或快表)可以一定程度上改善性能。

(3) 碎片：有些内存管理方案会导致内存碎片问题，也就是内存浪费，分为内部碎片和外部碎片。

(4) 重定位：有些内存管理方案允许在进程执行时，其逻辑地址能动态地进行重定位。如果地址只能在装入时重定位，那么将影响方案的能力。

(5) 交换：任何内存管理方案都可加上内外存之间的交换操作。这种交换操作允许更多进程运行，但伴随着性能的损失。

(6) 共享：允许不同进程共享代码和数据。

(7) 保护：当允许不同进程共享代码和数据时，内存管理方案必须同时提供对不同用户进程，对用户进程不同区域的访问控制。例如，只可执行、只读或可读可写。

内存管理中的内存保护方案，很大程度上取决于采用的内存管理方案。连续区存储管理、页式存储管理、段式存储管理，它们各自都有内存保护的对策。

2. 连续分配管理方式

1) 单一连续分配

这是一种最简单的存储管理方式，用于单用户、单任务的操作系统。在 8 位和 16 位微

型计算机上，主要都使用单用户、单任务的操作系统（如 CP/M 和 MS-DOS），采用这种存储管理方式，这种存储分配方式未采取有效存储保护措施。它将内存分为两个区。

（1）系统区：仅供操作系统使用，通常设置在内存的低段。

（2）用户区：指除系统区以外的全部内存空间，提供给用户使用。

2）固定分区分配

固定分区是在进程装入之前，内存就被划分成若干分区。一旦划分完成，在系统运行期间不再重新划分，即分区的个数不可变，分区的大小不可变，所以，固定分区又称静态分区。

这种分区方式一般将内存的用户区域划分成大小不等的分区，以适应不同大小的进程的需要。系统用一个分区表数据结构来管理内存分配情况，每个表项说明一个分区的大小、起始地址和是否已分配的使用标志，如表 3.1 所示。

由于每个分区的大小是固定的，所以每个提出运行的进程必须说明所需的最大内存容量。在调度时，内存管理程序根据进程所需的内存容量，在分区表中找出一个足够大的空闲分区分配给它，然后用重定位装入程序将此进程装入。如果找不到，则调度程序选择另外一个进程。图 3.1 说明了某一时刻，进程 A、B、C 分别被分配到 1、2、3 这 3 个分区中，第 4 分区尚未分配，操作系统则占据内存低地址区 20KB 的空间。当一个进程结束时，系统又调用内存管理程序查到分区表，把所占分区的使用标志修改为未分配状态即可。

表 3.1　分区表

区　号	大小/KB	起始地址	标　志
1	16	20K	已分配
2	32	36K	已分配
3	64	68K	已分配
4	124	132K	未分配

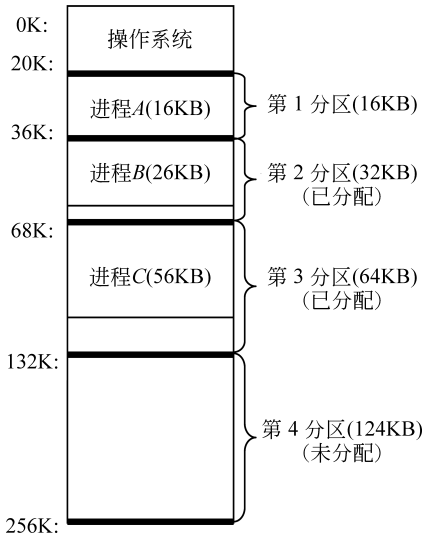

图 3.1　固定分区示例

采用这种技术，虽然可以使多个进程共驻内存，但是一个进程的大小不可能正好等于某个分区的大小，所以每个被分配的分区总有一部分被浪费，我们把这部分被浪费的存储区称为内零头（内部碎片）。

3）可变分区分配

可变分区是指在进程装入内存时，从可用的内存中划出一块连续的区域分配给它，且分

区大小正好等于该进程的大小。可变分区中分区的大小和分区的个数都是可变的，而且是根据进程的大小和多少动态地划分，因此又称动态分区分配。图 3.2 给出了可变分区的示例。

图 3.2 可变分区示例

系统初始化后，内存被划分成两块，一块用于常驻的操作系统，另一块则是完整的空闲区(用户区)。对于调入的若干进程，划分几个大小不等的分区给它们，随着进程的调入和撤销，相应的分区被划分和释放，原来整块的存储区形成空闲区和已分配区相间的局面。

可变分区的分配和释放的基本思想：在分配时，首先找到一个足够大的空闲分区，即这个空闲区的大小比进程要求的要大。系统则将这个空闲分区分成两部分，一部分成为已分配的分区，剩余的部分仍作为空闲区。在回收撤销进程所占领的分区时，要检查回收的分区是否与前后空闲的分区相邻接，若是，则加以合并，使之成为一个连续的大空间。

在这种内存管理中要解决的问题是分配算法，一般有 4 种。

(1) 首次适应(first fit)算法：从内存空闲分区表的第一个表目起查找该表，把最先能够满足要求的空闲区分配给进程，这种方法目的在于减少查找时间。为适应这种算法，空闲分区表(空闲区链表)中的空闲分区要按地址由低到高进行排序。该算法优先使用低地址部分空闲区，在低地址空间造成许多小的空闲区，在高地址空间保留大的空闲区。

(2) 最佳适应(best fit)算法：它从全部空闲区中找出能满足进程要求的且大小最小的空闲分区，这种方法能使碎片尽量小。为适应这种算法，空闲分区表(空闲区链表)中的空闲分区要按大小从小到大进行排序，自表头开始查找到第一个满足要求的自由分区分配。该算法保留大的空闲区，但造成许多小的空闲区。

(3) 最坏适应(worst fit)算法：从所有未分配的分区中挑选最大的且大于和等于进程大小的分区分给要求的进程，空闲分区按大小由大到小进行排序。该算法使小的空闲区减少，但造成大的空闲区不够大。

(4) 下次适应(next fit)算法：类似首次适应算法，每次分区时，总是从上次查找结束的地方开始，只要找到一个足够大的空白区，就把它划分后分配出去。

可变分区也有碎片问题。在系统不断地分配和回收中，必定会出现一些不连续的小的空闲区，称为外部碎片(也称外零头)。虽然可能所有碎片的总和超过某个进程的要求，但是由于不连续而无法分配。解决零头的方法是拼接(也称紧缩，compaction)，即向一个方向(例如，向低地址端)移动已分配的进程，使那些零散的小空闲区在另一方向连成一片。分区的拼接技术，一方面是要求能够对进程进行重定位，另一方面系统在拼接时要耗费较多的时间。

3. 非连续分配管理方式

各种连续分配管理方式的共同特点是要求分配给进程的物理空间必须为单一连续的。

这妨碍了存储管理方案的设计，也是引起大量碎片的主要原因。因此，有必要考虑非连续分配管理方式。其中，最有代表性的是分页管理方式、分段管理方式，以及二者的混合方式，如段页式管理方式。

1）分页管理方式

分页管理方式要求进程的物理内存空间分为固定大小的块，称为页帧(frame)，对页帧依次编号，即页帧号；进程的逻辑内存空间也分为同样大小的块，称为页(page)，对页依次编号，即页号；页的长度等于页帧的长度。为方便地址映射，最好要求页、页帧的长度正好为 2 的次幂。分页管理方案需要页表支持，每个进程独立拥有一张页表，页表的表项保存了页帧号，而页号则作为页表的索引地址。

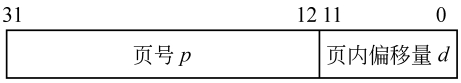

图 3.3 分页系统的地址结构

分页系统的地址结构如图 3.3 所示，它由两部分组成：前一部分为页号 p；后一部分为页内偏移量 d，即页内地址。图 3.3 中的地址长度为 32 位，其中 0～11 位为页内地址(每页的大小为 4KB)，12～31 位为页号，所以允许地址空间的大小最多为 1M 页。

在将进程的每页离散地分配到内存的多个页帧中后，系统应能保证进程的正确运行，即能在内存中找到每个页面所对应的页帧。为此，系统为每个进程建立了一张页面映射表，称为页表(见图 3.4)。每页在页表中占一个表项，记录该页在内存中对应的页帧号。进程在执行时，通过查找页表，就可以找到每页所对应的页帧号。可见，页表的作用是实现从页号到页帧号的地址映射。

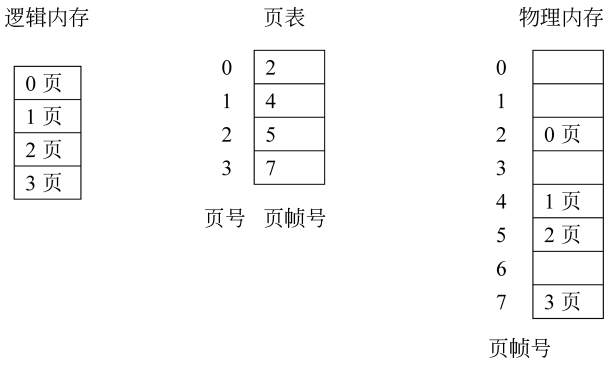

图 3.4 利用页表实现地址映射

分页管理方式下的地址映射，较连续分配管理方式复杂些，如图 3.5 所示。由 CPU 所生成的逻辑地址 L 分为两部分：页号(p)和页内偏移量(d)。以页号 p 作为索引查页表，得到页表项，页表项包含了页面对应的页帧号 f，即该页面在物理内存的基址。这个基址与页内偏移量 d 的组合，就形成了物理地址，就是逻辑地址 L 对应的物理地址。

分页技术不会产生外部碎片：每个页帧都可以分配给需要它的进程。不过，分页有内部碎片，由于进程的最后一页通常装不满一页帧，因此每个进程平均有半页的内部碎片。这意味着，页长度越小，碎片越少。不过，在逻辑空间总长度不变的情况下，页长度越小，意味

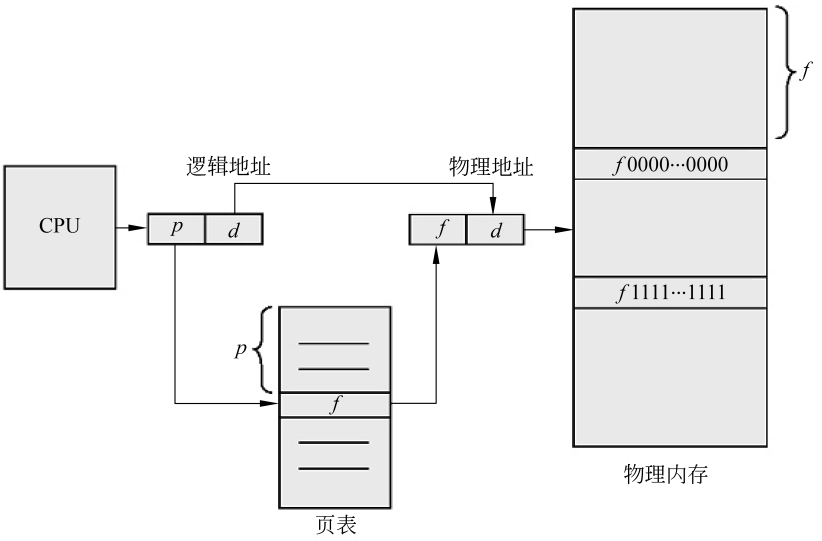

图 3.5 分页管理方式的地址映射

着页表中的表项数越多，页表占用的物理内存空间越多。

分页的一个重要特点是，虽然分配给用户进程的众多页帧并不一定连续，但是进程的逻辑空间依然是连续的；虽然物理空间驻留众多用户进程，但是用户程序在逻辑空间里“误以为”拥有整个内存。实现这种特点的关键是，逻辑地址转变成物理地址的地址映射。这种映射是用户所不知道的，但是受操作系统控制。因此，为操作系统带来许多“想象空间”。

如何保存页表也是分页管理方式必须解决的问题之一。

(1) 将页表作为一组专用寄存器来实现，地址映射效率很高，但是无法容纳大页表。

(2) 将页表放在内存中，并引入页表基址寄存器(page table base register，PTBR)指向那个拥有 CPU 的进程的页表。它可以容纳大页表，但是访问一个字节需要两次内存访问(一次用于页表项，一次用于字节)，使内存访问的速度减半。

(3) 采用称作 TLB 的硬件缓冲器。TLB 只包括页表的一小部分表项，TLB 表项由两部分组成：页号和对应的页帧号。如图 3.6 所示，映射一个逻辑地址时，其页号提交给 TLB。如果页号匹配(称为 TLB 命中)，那么就从 TLB 对应表项得到页帧号，进而与页内偏移量组合得到物理地址。如果页号不在 TLB 中(称为 TLB 失效)，那么需要访问内存中的页表，得到页帧号。此时，一方面用页帧号与页内偏移量组合得到物理地址访问内存；同时，将页号和页帧号添加到 TLB 中。如果 TLB 已满，那么须根据替换策略替换 TLB 的一个表项。

指定页号在 TLB 中被查找到的百分比称为命中率。这种由 TLB 参与下访问内存所耗用的时间称为有效访问时间。命中率越高，有效访问时间越短。

分页管理方式下，内存保护通过页表项中的相应保护位来实现。

当逻辑地址空间过大时，例如 64 位机的地址空间，页表也非常大，所以需要考虑组织页表的技术。常用的有多层页表、哈希页表、反向页表等。

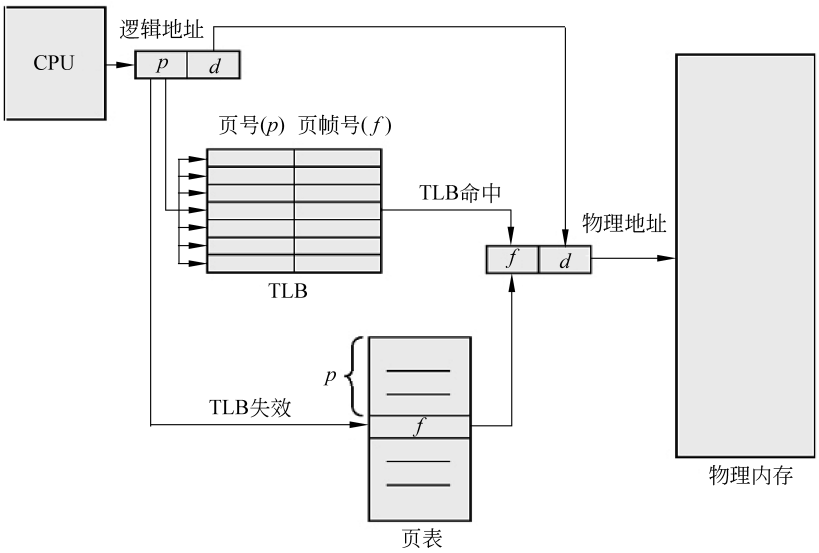

图 3.6 带 TLB 的地址映射

2) 分段管理方式

在非连续分配类型的存储管理方案中,与分页管理方式同样流行的,还有分段管理方式。其实,分段管理方式与分页管理方式极其相似,将二者对比着学习、掌握,不失为上策。

分页管理方式把进程的逻辑空间分割成大小一样的“页”,并且分别离散地对应到物理空间的页帧。分段管理方式与之唯一不同的是,将进程逻辑空间按照用户的视角(函数、数组、堆栈等)把逻辑空间分割成“段”,并且离散地对应到物理空间中。于是,逻辑地址不用<页号,页内偏移量>了,而代之以<段号 s,段内偏移量 d>。页表不用了,代之以段表。每个进程都拥有一张段表,段表项包含了该连续段在物理空间中的基址,以及段的有效长度。

分段管理方式的地址映射,就是将二维的逻辑地址映射到一维的物理地址,如图 3.7 所示。根据段号 s 从段表中找到对应段表项,取得基址和段长度。如果段内偏移量 d 超过了段长度,则产生段越界中断,报错。否则,将基址与 d 相加,即得到对应的物理地址了。

分段管理方式下的共享和保护,都通过段表进行,通过定义段表项的指定的位(bit)达到目的。这里,得到共享和保护的对象,在用户进程中具有明确的逻辑意义,相对于分页管理方式,此优点非常显著。

分页管理方式下不必关心物理空间的分配问题,因为页帧长度都是一样的。只要有空闲页帧,就可以拿来用。而分段管理方式却不同了,因为段长度是不同的。操作系统必须为进程的所有段分配空间。对照连续分配管理方案里的可变长多分区连续分配管理方法,所使用的动态存储空间分配算法与分段方式是相似的,只不过前者为整个进程寻找空间,后者为进程的一个段寻找空间。因此,分段管理方式下不会引起内部碎片问题,却有外部碎片。

保存段表的问题,也可以参照保存页表来解决:①将段表作为一组专用寄存器;②将段表放在内存中,并引入段表基址寄存器指向那个拥有 CPU 的进程的段表;③部分采用硬件缓冲器,结合内存中的完整段表。

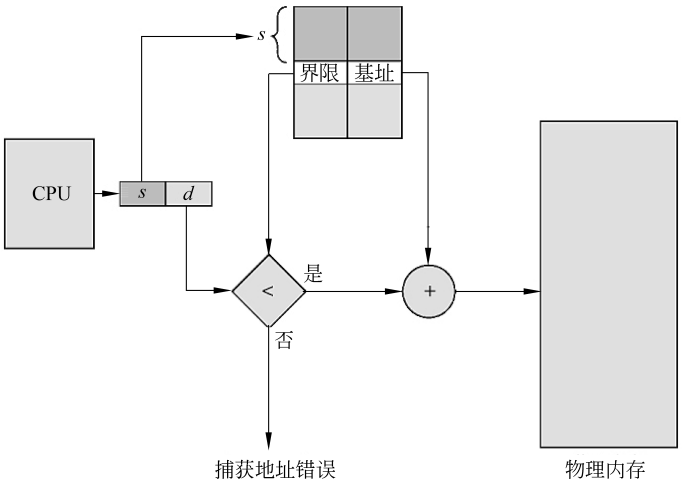

图 3.7 分段管理方式的地址映射

3) 段页式管理方式

分页管理方式和分段管理方式各有优缺点。采用段页式管理方式,即先将进程逻辑地址空间分段,建立段表,实现分段管理方式的地址映射,再将每段分页,建立页表,实现分页管理方式的地址映射,这种混合方式虽然会带来一定的复杂性(靠 MMU 这样的硬件装置解决),但是可以取长补短,体现了分页管理方式和分段管理方式的综合优点。在当今流行的 CPU 中,大多采用这种思想,例如 Intel CPU 的 Intel 80386 体系结构,如图 3.8 所示。

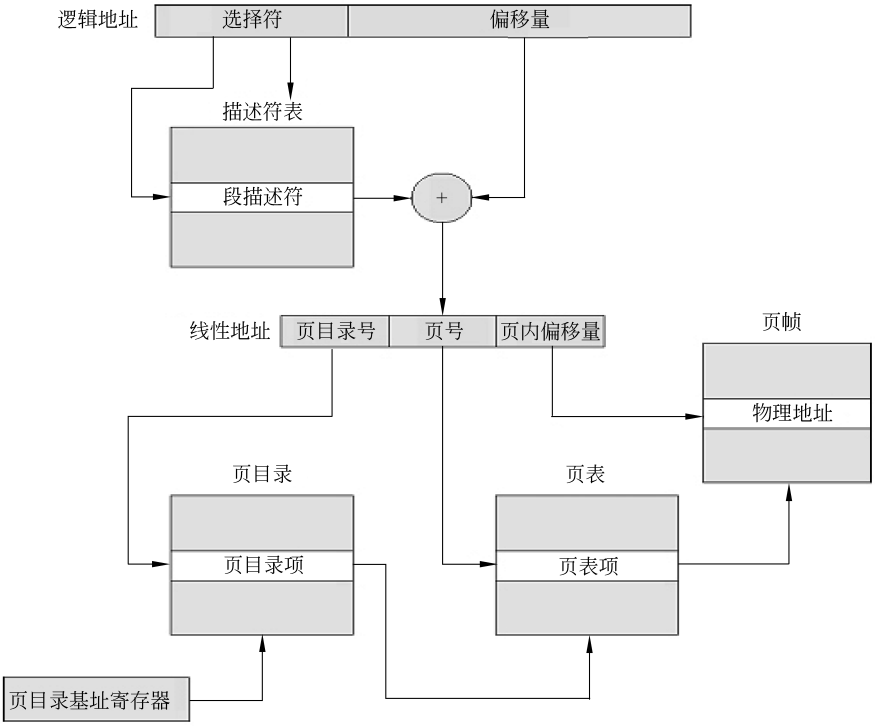

图 3.8 Intel 80386 的地址映射

3.2.2 虚拟内存管理

1. 虚拟内存基本概念

操作系统之所以能够采用分页管理方式、分段管理方式等实现各种存储空间管理的解决方案，并且同时保证用户进程拥有一个连续、完整的地址空间，其核心原因是操作系统接管了用户进程逻辑地址转换成计算机物理地址的映射。操作系统利用这个便利，应该发挥更大的作为。

另外，CPU 执行任何一条指令时，它只要求指令操作码及其操作数驻留内存，并没有要求当前运行进程的所有程序、代码等都驻留内存。也就是说，若不考虑系统性能因素，操作系统没有必要把整个进程的逻辑地址空间都装入内存。

综上两条，操作系统的存储管理方案，可以支持比物理空间更大的逻辑空间的进程运行；或者，可以支持更多的进程运行，这些进程的逻辑地址总和大于物理空间。这样的存储管理方案，就是虚拟存储管理。

虚拟存储管理的依据是程序局部性原理。程序在执行时将呈现出局部性规律，即在一段时间内，程序的执行仅局限于某个部分，相应地，它所访问的存储空间也局限于某个区域内。局限性表现在以下两方面。

(1) 时间局部性：如果程序中的某条指令一旦执行，则不久的将来该指令可能再次被执行；如果某个存储单元被访问，则不久以后该存储单元可能再次被访问。产生时间局部性的典型原因是在程序中存在大量的循环操作。

(2) 空间局部性：一旦程序访问了某个存储单元，则在不久的将来，其附近的存储单元也最有可能被访问。也就是说，程序在一段时间内所访问的地址，可能集中在一定的范围内，其典型原因是程序是顺序执行的。

虚拟存储器是指仅把程序的一部分装入内存便可运行程序的存储器系统。具体地说，虚拟存储器是指具有请求调入功能和置换功能，能从逻辑上对内存容量进行扩充的一种存储器系统。

2. 请求分页管理方式

请求分页管理方式则是一种典型的虚拟存储管理方案，是目前常用的一种虚拟存储管理的方式。请求分页管理方式是建立在纯分页基础上的，它是在分页系统的基础上，增加了请求调页功能、页面置换功能所形成的页式虚拟存储系统。它允许只装入若干页(而非全部程序)的用户程序和数据，便可启动运行。以后，再通过调页功能及页面置换功能，陆续地把即将要运行的页面调入内存，同时把暂不运行的页面换出到外存上，置换时以页面为单位。

为了能实现请求调页和置换功能，计算机系统必须提供缺页中断机制。

1) 请求分页的页表机制

它是在纯分页的页表机制上形成的，由于只将应用程序的一部分调入内存，还有一部分仍在磁盘上，故需要在页表中再增加若干项，供程序(数据)在换进、换出时参考。请求分页

系统中的每个页表项如图 3.9 所示。

页帧号	状态位 P	访问位 A	修改位 M	外存地址

图 3.9　请求分页系统中的每个页表项

各字段含义如下。

(1) 状态位(存在位 P)：用于指示该页是否已调入内存，供程序访问时参考。

(2) 访问位 A：用于记录本页在一段时间内被访问的次数，或最近已有多长时间未被访问，提供给置换算法选择换出页面时参考。

(3) 修改位 M：该页在调入内存后是否被修改过。由于内存中的每页都在外存上保留一份副本，因此，若未被修改，在置换该页时就不需将该页写回到外存上，以减少系统的开销和启动磁盘的次数；若已被修改，则必须将该页重写到外存上，以保证外存中所保留的始终是最新副本。

(4) 外存地址：用于指出该页在外存上的地址，通常是物理块号，供调入该页时使用。

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时，便要产生一缺页中断，请求操作系统将所缺页调入内存。缺页中断与普通中断的主要区别如下。

(1) 在指令执行期间产生和处理中断信号。

(2) 一条指令在执行期间，可能产生多次缺页中断。

缺页处理的过程如图 3.10 所示。

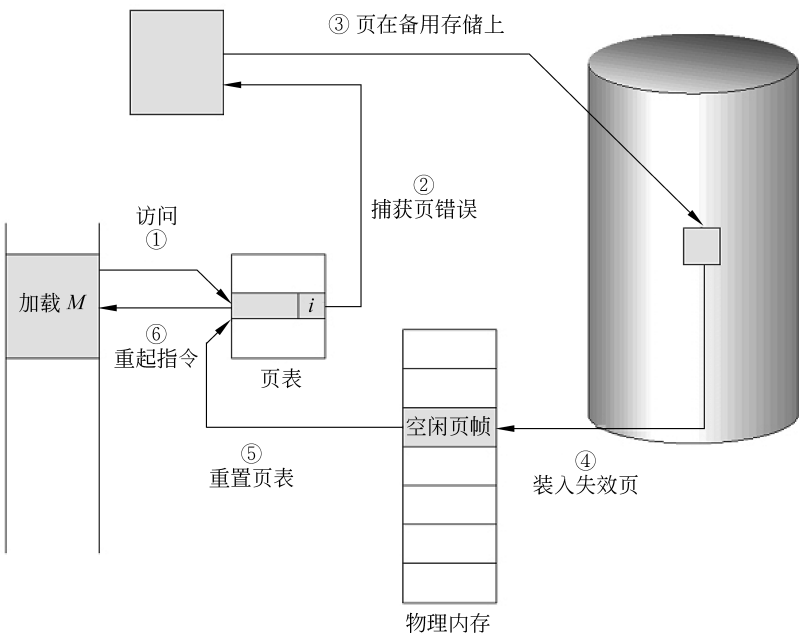

图 3.10　缺页处理的过程

进程运行时，当前指令或者操作数（图 3.10 中的“加载 M”指令）不在内存时，产生缺页，处理步骤如下。

（1）该逻辑地址对应的页表项中的“有效地址”标志位为 0，引起缺页中断。

（2）在缺页中断服务程序中判断，如果本次引用非法，那么终止进程。

（3）如果引用有效，则表示该逻辑地址对应的内容尚未调入内存页面，那么，试图找到一个空闲物理页帧。

（4）如果找到空闲物理页帧，操作系统请求一个磁盘操作，将需要的页调入空闲页帧。

（5）如果找不到一个空闲页帧（这是正常的，否则就不需要虚拟存储管理了），那么，必须启动页面置换算法，腾出一个空闲页帧。

（6）装入新页面后，修改进程的页表，以及其他数据结构。

（7）重新执行被中断的指令。

在整个过程中，用户进程并没有变更其控制流，就好像指令始终在内存中。要说差异，只是运行速度受到影响了。另外，请关注页面置换算法被调用的时机。

3）缺页率对有效访问时间的影响

在请求分页系统中，假设存储器的访问时间 m_a 为 100ns（一般为十纳秒到几百纳秒），缺页率为 p，缺页中断时间为 25ms，则有效访问时间就可以表示为

$$\begin{aligned}\text{有效访问时间} &= (1-\text{缺页率})\times\text{访问时间}+\text{缺页率}\times\text{缺页中断时间}\\ &= (1-\text{缺页率})\times 0.1+\text{缺页率}\times 25000\\ &= 0.1+24999.9\ \text{缺页率}\end{aligned}\tag{3.1}$$

由式（3.1）可以看出，有效访问时间与缺页率成正比。如果缺页率为 0.1%，则有效访问时间约为 25μs，与直接访问存储器的访问时间（0.1μs）相比，程序执行的性能将受到严重的影响，所以在虚拟存储系统中应该保持非常低的缺页率，同时选用高速磁盘和高速磁盘接口。

3. 页面置换算法

常用的页面置换算法有最佳（OPT）置换算法；先进先出（FIFO）置换算法；最近最少使用（LRU）置换算法；时钟（CLOCK）置换算法。

OPT 置换算法需要预知将来的页面使用情况，无法实现。FIFO 置换算法容易编程，但有 Belady 异常。LRU 置换算法能近似最优算法，但是它也很难实现。现实中大多数页面置换算法都是 LRU 置换算法的近似，如 CLOCK 置换算法。

1）最佳置换算法

最佳置换算法是一种理想化的算法，性能最好，但在实际上难于实现。也就是说，选择那些永不使用的，或者是在最长时间内不再被访问的页面置换出去。但是要确定哪个页面是未来最长时间内不再被访问的，目前来说是很难估计的，所以该算法通常用来评价其他算法。

例如，假定系统为某进程分配了 3 个物理块，并考虑有以下的页面引用串：7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1。如图 3.11 所示，进程运行时先后将 7,0,1 这 3 个页面装入内存。当进程访问页面 2 时，产生缺页中断，此时操作系统根据最佳置换算法，页面 7 将在第 18 次才被访问，是 3 个页面中将最久不被访问的页面，所以被淘汰。接着访问页面 0 时，发现已在内存中，而不会产生缺页中断，以此类推。

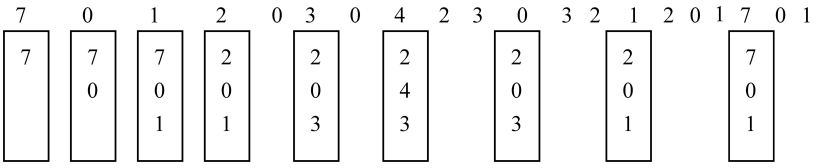

图 3.11　利用最佳置换算法的置换图

从图 3.11 可以看出，采用最佳置换算法，只发生了 6 次页面置换，产生了 9 次缺页中断。

2）先进先出置换算法

先进先出置换算法是一种最直观、性能最差的算法。该算法总是淘汰最先进入内存的页面，即选择在内存中驻留时间最久的页面予以淘汰。该算法实现简单，只需要把一个进程已调入内存的页面，按先后次序链接成一个队列，并设置一个指针即可。

仍用上面这个例子，如果采用先进先出置换算法进行页面置换，如图 3.12 所示。当进程访问页面 2 而产生缺页中断时，因为第 7 页是最先被调入内存的，所以被换出。可以看出，利用先进先出置换算法，产生了 15 次缺页中断，共发生了 12 次页面置换行为，比最佳置换算法多了一倍。

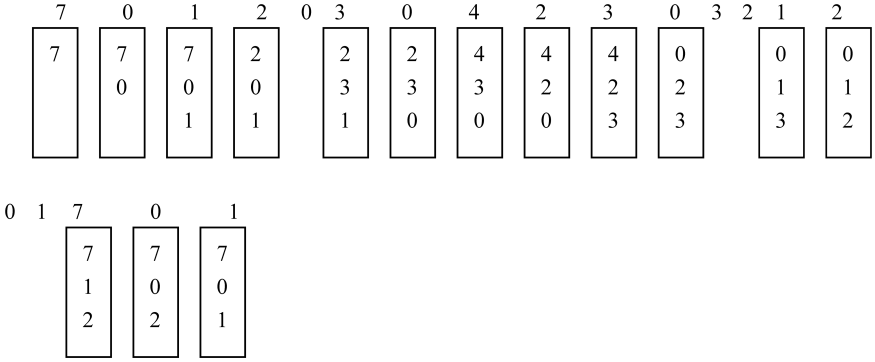

图 3.12　利用先进先出置换算法的置换图

3）最近最少使用置换算法

最近最少使用置换算法是选择最近最少使用的页面予以淘汰，系统在每个页面设置一个访问位，用以记录这个页面自上次被访问以来所经历的时间 T，当要淘汰一个页面时，选择 T 最大的页面。但在实现时需要硬件的支持（寄存器或栈）。利用最近最少使用置换算

法对上面的例子进行页面置换的结果如图 3.13 所示。

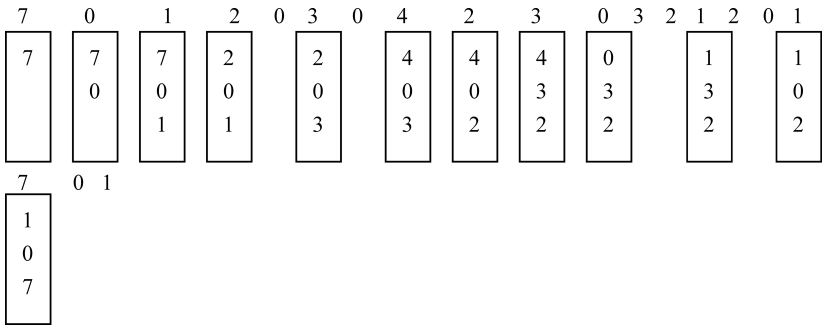

图 3.13 利用最近最少使用置换算法的置换图

4) 时钟置换算法

时钟置换算法是一种近似的最近最少使用置换算法。该算法为每个页面设置一位访问位,将内存中的所有页面都通过链接指针链成一个循环队列。当某页被访问时,其访问位置 1。在选择一页淘汰时,就检查其访问位,如果是 0,就选择该页换出;若为 1,则重新置为 0,暂不换出该页,在循环队列中检查下一个页面,直到访问位为 0 的页面为止。由于该算法只有一位访问位,只能用它表示该页是否已经使用过,而置换时是将未使用过的页面换出,所以把该算法称为最近未用(not recently used,NRU)置换算法。

4. 页面分配策略

前面所述的页面置换算法不为进程预先分配页帧;只有引用时,才通过缺页中断加以分配。为了提高效率,可以为进程预先分配若干页帧。页面分配策略有多种,可以是固定数量的,可以是按比例的,也可以是按进程优先级的,等等。

在请求分页系统中,可采取两种分配策略:固定和可变分配策略。在进行置换时,也可采取两种策略:全局置换和局部置换。于是可组合成以下 3 种策略。

(1) 固定分配局部置换策略:基于进程的类型(交互型或批处理型等),或根据程序员、系统管理员的建议,为每个进程分配一个固定页数的内存空间,在整个运行期间都不再改变。如果进程在运行中发现缺页,则只能从该进程在内存的固定页面中选出一页换出,然后再调入另一页,保证分配给该进程的内存空间不变。

(2) 可变分配全局置换策略:系统为每个进程分配一定数目的物理块,而操作系统本身也保持一个空闲物理块队列。当某进程发现缺页时,由系统从空闲物理块队列中,取出一物理块分配给该进程,并将欲调入的缺页装入其中。当空闲物理块队列中的物理块用完时,操作系统才能从内存中选择一页调出,该页可能是系统中任一进程的页。

(3) 可变分配局部置换策略:根据进程的类型或程序员的要求,为每个进程分配一定数目的内存空间;但当某进程发生缺页时,只允许从该进程在内存的页面中选出一页换出,而不影响其他进程的运行。

在采用固定分配策略时，可采用以下几种物理块分配方法。

(1) 平均分配算法：将系统中所有可供分配的物理块，平均分配给各个进程。

(2) 按比例分配算法：这是根据进程的大小按比例分配物理块。

(3) 考虑优先权的分配算法：该方法是把内存中可供分配的所有物理块分成两部分。一部分按比例分配给各进程；另一部分则根据各进程的优先权，适当地增加其相应份额后，分配给各进程。

5. 抖动和工作集

如果进程没有足够多的页帧，将会导致频繁的缺页中断，大量的页面换入、换出。当计算机系统用于页面交换的时间明显多时，它就会颠簸(thrashing)，或者叫抖动。

工作集的理论是在1968年由Denning提出来的。他认为，程序在运行时对页面的访问是不均匀的，即往往在某段时间内的访问仅局限于较少的若干页面，如果能够预知程序在某段时间间隔内要访问哪些页面，并能将它们提前调入内存，将会大大地降低缺页率，从而减少置换工作，提高CPU的利用率。

工作集是指在某段时间间隔(Δ)里，进程实际要访问的页面的集合。Denning认为，虽然程序只需有少量几页在内存就可以运行，但为了使程序能够有效地运行，较少地产生缺页，就必须使程序的工作集全部在内存中。把某进程在时间 t 的工作集记为 $w(t,\Delta)$，把变量 Δ 称为工作集窗口尺寸(windows size)。正确选择工作集窗口尺寸，对存储器的有效利用和系统吞吐量的提高，都将产生重要的影响。

在有些操作系统中，如Windows，虚拟存储管理程序为每个进程分配固定数量的页面，并且这个数目可以进行动态地调整。那么这个数量如何确定？又如何进行动态地调整呢？这个数目就是由每个进程的工作集来确定，并且根据内存的负荷和进程的缺页情况动态地调整其工作集。

其具体的做法：一个进程在创建时就指定了一个最小工作集，该工作集的大小是保证进程运行在内存中应有的最小页面数。但在内存负荷不太大(页面不太满)时，虚拟存储管理程序还允许进程拥有尽可能多的页面作为其最大工作集。当内存负荷发生变化时，如空闲页架(块)不多了，虚拟存储管理程序就使用“自动调整工作集”的技术来增加内存中可用的自由页架。方法是检查内存中的每个进程，将它当前工作集大小与其最小工作集进行比较。如果大于最小值，则从它们的工作集中移去一些页面作为内存自由页面，可为其他进程所使用。若内存自由页面仍然太小，则不断进行检查，直到每个进程的工作集都达到最小值为止。

当每个工作集都已达到最小值时，虚拟存储管理程序跟踪进程的缺页数量，根据内存中自由页面数量可以适当增加其工作集的大小。

3.3 典型真题、同步测试练习及分析

3.3.1 单项选择题

在下列存储管理方案中，不适用于多道程序设计的是(　　)。

A. 单一连续分配　　B. 固定分区分配

C. 可变分区分配　　D. 段页式存储管理

答案　A。

分析　单一连续分配早期用于单道批处理系统，现只用于单用户、单任务的微型计算机操作系统，它不适用于多道程序设计。

(2009)　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

答案　A。

分析　B、C、D 都不是存储保护的措施。使用界限寄存器(limit register)可以实现分区分配内存管理方法的存储保护。

存储管理中，下列说法中正确的是(　　)。

A. 无论采用哪种存储管理方式，用户程序的逻辑地址均是连续的

B. 地址映射需要有硬件地址转换机制作支持

C. 段表和页表都是由用户根据进程情况而建立的

D. 采用静态重定位可实现程序浮动

答案　B。

分析　段式管理时用户使用的逻辑地址是不连续的，页表由系统确定，地址映射一定要有硬件地址转换机制作支持，采用静态重定位不能实现程序浮动。

题 4 在可变分区存储管理方案中需要一对界限地址寄存器，其中(　　)作为地址映射(重定位)使用。

A. 逻辑地址寄存器　B. 长度寄存器　　C. 物理地址寄存器　D. 基址寄存器

答案 D。

分析 在可变分区存储管理系统设置一对基址、限长寄存器，此时基址寄存器起着重定位寄存器的作用，运行进程所在分区的起始地址和大小分别装入基址寄存器和限长寄存器。

题 5 分段存储管理系统中信息的逻辑地址到物理地址的变换是通过(　　)来实现的。

A. 段表　　B. 页表　　C. 物理结构　　D. 重定位寄存器

答案 A。

分析 在分段存储管理系统中为每个进程建立一张段表，每个段在表中占有一个表项，在其中记录了该段在内存中的起始地址(又称基址)和段的长度。进程在执行中，通过查段表来找到每个段所对应的内存区，段表实现了从逻辑段到物理内存区的映射。

题 6 (2010)　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} 字节，页表项大小为 2 字节，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

答案 B。

分析 多级分页机制是使页表离散存储，以页帧为单位存储。根据题目，每个页帧的大小为 2^{10}B 即 1KB。每个页帧能存放 1KB/2＝512 个页表项，暂且称为一个页表，即一个页表有 512 个页表项，而逻辑地址空间大小为 2^{16} 页，则需要 2^{16} 个页表项，可以计算出页表的个数为 $2^{16}/512=2^7=128$，即在页目录表中需要有 128 个地址来指向 128 个页表，所以页目录表中包含表项的个数至少是 128。

题 7 有利于进程动态链接的内存管理方法是(　　)。

A. 分段虚拟存储管理　　B. 分页虚拟存储管理

C. 动态(可变)分区管理　　D. 固定式分区管理

答案 A。

分析 动态链接机制是为了各个进程共享动态链接库中的函数，提高了内存利用率。分段虚拟存储管理中的每个段是按照程序逻辑意义上划分的，如一个函数为一个段，动态链接库中的函数可以作为一个段来管理。相比分页虚拟存储管理来讲，分段机制可节省段表占用的空间。目前流行的通用操作系统（如 Windows、Linux 操作系统）都是使用分页虚拟存储管理方法，当然也使用动态链接技术。选项 A 是最佳答案。

题 8 在动态分区系统中，空闲块的块大小和块基址如表 3.2 所示。

表 3.2 空闲块的块大小和块基址

空 闲 块	块大小/KB	块 基 址
1	80	60K
2	75	150K
3	55	250K
4	90	350K

此时，某进程 *P* 请求 50KB 内存，系统从第 1 个空闲块开始查找，结果把第 4 个空闲块分配给了 *P* 进程，请问是用哪种分区分配算法实现这一方案？（ ）

A. 首次适应　　B. 最佳适应　　C. 最坏适应　　D. 下次适应

答案 C。

分析 系统从第 1 空闲块开始查找，到第 4 空闲块查到，有 4 块空闲空间分别为 60K 起始地址的 80KB、150K 起始地址的 75KB、250K 起始地址的 55KB、350K 起始地址的 90KB，满足申请 50KB 内存空间要求，现最终分配第 4 块是最大的，所以是采用最坏适应算法。

题 9 在分区存储管理中的拼接（compaction）技术可以（ ）。

A. 缩短访问周期　　B. 增加主存容量

C. 集中空闲区　　D. 加速地址转换

答案 C。

分析 在分区存储管理中解决零头的方法是拼接或紧缩（compaction），即向一个地址方向（例如向低地址端）移动已分配的作业，使那些零散的小空闲区在另一方向连成一片。

题 10 在页式存储管理中，可以用位示图表示内存空闲块状况。假设字长为 32 位，每位（编号为 0～31）与一个内存块对应，取值可为 0 或 1。当取值为 1 时表示对应块已被占用，当取值为 0 时表示对应块为空闲。如果内存可分配区被划分为

1024 块，则位示图共需要(　　)个字来表示。

A. 15　　B. 16　　C. 31　　D. 32

答案 D。

分析 字长为 32 位，1024 块需要 1024/32＝32 个字。

题 11 在页式存储管理中，可以用位示图表示内存空闲块状况。假设字长为 32 位，每位(编号为 0～31)与一个内存块对应，取值可为 0 或 1。当取值为 1 时表示对应块已被占用，当取值为 0 时表示对应块为空闲。已知某位的字号是 5，位号为 14，假设字号也从 0 开始编号。则对应的内存块号是(　　)。(假设内存块从 0 开始编号)

A. 70　　B. 105　　C. 174　　D. 224

答案 C。

分析 内存块号＝字号×字长＋位号＝5×32＋14＝174。

题 12 假设页的大小为 4KB，页表的每个表项占用 4B。对于一个 64 位地址空间系统，采用多级页表机制，至少需要(　　)级页表。

A. 2　　B. 3　　C. 6　　D. 7

答案 C。

分析 内存每帧的大小是 4KB，每个页表项占用 4B，则每帧可以存放 1K 个页表项地址，采用 n 级页表可以寻址的地址空间的容量为$(1K)^n \times 4KB = 2^{10n+12}B$。64 位地址空间的大小为 $2^{64}B$。为使 n 级页表可以寻址 64 位的地址空间，应使下式成立：$2^{10n+12} = 2^{64}$。可解得 $n=6$。因此用 6 级页表就可以解决 64 位地址空间的寻址问题。

题 13 (2010)　某基于动态分区存储管理的计算机，其主存容量为 55MB(初始为空闲)，采用最佳适应(best fit)算法，分配和释放的顺序：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

答案 B。

分析 在主存容量为 55MB 中，采用最佳适应算法，对于题中的分配和释放顺序的过程可以用图 3.14 来表示，图中有灰色底纹的表示已经分配，空白部分表示空闲。经过分配 15MB、分配 30MB、释放 15MB、分配 8MB、分配 6MB 后，最后一个方框有黑色底纹的为最大空闲分区，即 9MB 空间。

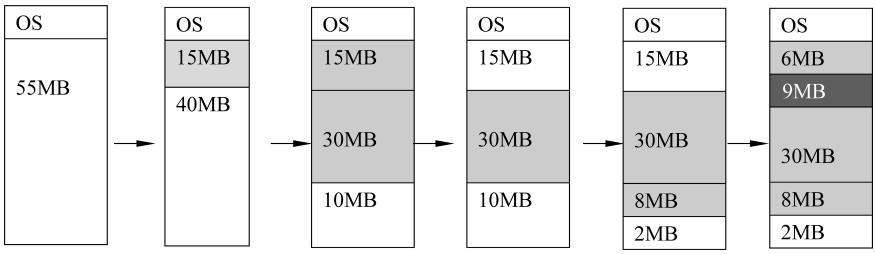

图 3.14 题 13 的分配和释放顺序

题 14 在有联想存储器(TLB)的存储管理系统中,假设工作集的大小为 400KB(4KB/页),要得到一个较好的命中率,最小需要(　　)个表项的联想存储器。

A. 50　　B. 100　　C. 200　　D. 400

答案 B。

分析 工作集的大小为 400KB,每页 4KB,工作集的大小为 100 页,要得到一个较好的命中率,最小需要 100 个表项的联想存储器。

题 15 在一分页存储管理系统中,页表内容如表 3.3 所示。

表 3.3 页表(题 15)

页　　号	页 帧 号
0	2
1	1
2	8

若页大小为 1K,逻辑地址的页号为 2,页内地址为 451,转换成的物理地址为(　　)。

A. 8643　　B. 8192　　C. 2048　　D. 2499

答案 A。

分析 逻辑地址的页号为 2,页内偏移量为 451。查找页表,进程的第 2 页放在内存的第 8 帧,因此该逻辑地址所对应的物理地址为 8×1K+451=8643。

题 16 如果一个程序为多个进程所共享,那么该程序的代码在执行的过程中不能被修改,即程序应该是(　　)。

A. 可执行码　　B. 可重入码　　C. 可改变码　　D. 可再现码

答案 B。

分析 在实现段共享时,共享程序段必须是可重入代码或纯代码。它是一种允许多个