择一些块丢掉。除了花费时间较短外（有关操作必须在若干纳秒中完成，而不是像页面置换那样在微秒级上完成），这个问题同页面置换问题完全一样。之所以花费时间较短，其原因是丢掉的高速缓存块可以从内存中获得，而内存既没有寻道时间也不存在旋转延迟。

第二个例子是Web服务器。服务器可以把一定数量的经常访问的Web页面存放在存储器的高速缓存中。但是，当存储器高速缓存已满并且要访问一个不在高速缓存中的页面时，就必须要置换高速缓存中的某个Web页面。由于在高速缓存中的Web页面不会被修改，因此在磁盘中的Web页面的副本总是最新的。而在虚拟存储系统中，内存中的页面既可能是干净页面也可能是脏页面。除此之外，置换Web页面和置换虚拟内存中的页面需要考虑的问题是类似的。

在接下来讨论的所有页面置换算法中都存在一个问题：当需要从内存中换出某个页面时，它是否只能是缺页进程本身的页面？这个要换出的页面是否可以属于另外一个进程？在前一种情况下，可以有效地将每一个进程限定在固定的页面数目内；后一种情况则不能。这两种情况都是可能的。在3.5.1节我们会继续讨论这一点。

3.4.1 最优页面置换算法

很容易就可以描述出最好的页面置换算法，虽然此算法不可能实现。该算法是这样工作的：在缺页中断发生时，有些页面在内存中，其中有一个页面（包含紧接着的下一条指令的那个页面）将很快被访问，其他页面则可能要到10、100或1000条指令后才会被访问，每个页面都可以用在该页面首次被访问前所要执行的指令数作为标记。

最优页面置换算法规定应该置换标记最大的页面。如果一个页面在800万条指令内不会被使用，另外一个页面在600万条指令内不会被使用，则置换前一个页面，从而把因需要调入这个页面而发生的缺页中断推迟到将来，越久越好。计算机也像人一样，希望把不愉快的事情尽可能地往后拖延。

这个算法惟一的问题就是它是无法实现的。当缺页中断发生时，操作系统无法知道各个页面下一次将在什么时候被访问。（在最短作业优先调度算法中，我们曾遇到同样的情况，即系统如何知道哪个作业是最短的呢？）当然，通过首先在仿真程序上运行程序，跟踪所有页面的访问情况，在第二次运行时利用第一次运行时收集的信息是可以实现最优页面置换算法的。

用这种方式，我们可以通过最优页面置换算法对其他可实现算法的性能进行比较。如果操作系统达到的页面置换性能只比最优算法差1%，那么即使花费大量的精力来寻找更好的算法最多也只能换来1%的性能提高。

为了避免混淆，读者必须清楚以上页面访问情况的记录只针对刚刚被测试过的程序和它的一个特定的输入，因此从中导出的性能最好的页面置换算法也只是针对这个特定的程序和输入数据的。虽然这个方法对评价页面置换算法很有用，但它在实际系统中却不能使用。下面我们将研究可以在实际系统中使用的算法。

3.4.2 最近未使用页面置换算法

为使操作系统能够收集有用的统计信息，在大部分具有虚拟内存的计算机中，系统为每一页面设置了两个状态位。当页面被访问（读或写）时设置R位；当页面（即修改页面）被写入时设置M位。这些位包含在页表项中，如图3-11所示。每次访问内存时更新这些位，因此由硬件来设置它们是必要的。一旦设置某位为1，它就一直保持1直到操作系统将它复位。

如果硬件没有这些位，则可以进行以下的软件模拟：当启动一个进程时，将其所有的页面都标记为不在内存；一旦访问任何一个页面就会引发一次缺页中断，此时操作系统就可以设置R位（在它的内部表格中），修改页表项使其指向正确的页面，并设为READ ONLY模式，然后重新启动引起缺页中断的指令；如果随后对该页面的修改又引发一次缺页中断，则操作系统设置这个页面的M位并将其改为READ/WRITE模式。

可以用R位和M位来构造一个简单的页面置换算法：当启动一个进程时，它的所有页面的两个位都由操作系统设置成0，R位被定期地（比如在每次时钟中断时）清零，以区别最近没有被访问的页面和被访问的页面。

当发生缺页中断时，操作系统检查所有的页面并根据它们当前的R位和M位的值，把它们分为4类：