正如你可以想象的，MFU 和 LFU 置换都不常用。这些算法的实现是昂贵的，并且它们不能很好地近似 OPT 置换。

9.4.7　页面缓冲算法

除了特定页面置换算法之外，还经常采用其他措施。例如，系统通常保留一个空闲帧缓冲池。当出现缺页错误时，会像以前一样选择一个牺牲帧。然而，在写出牺牲帧之前，所需页面就读到来自缓冲池的空闲帧。这种措施允许进程尽快重新启动，而无需等待写出牺牲帧。当牺牲帧以后被写出后，它被添加到空闲帧池。

这种方法的扩展之一是，维护一个修改页面的列表。每当调页设备空闲时，就选择一个修改页面以写到磁盘上，然后重置它的修改位。这种方案增加了在需要选择置换时干净的且
412 无需写出的页面的概率。

另一种修改是，保留一个空闲帧池，并且记住哪些页面在哪些帧内。因为在帧被写到磁盘后帧内容并未被修改，所以当该帧被重用之前，如果再次需要，那么旧的页面可以从空闲帧池中直接取出并被使用。这种情况不需要 I/O。当发生缺页错误时，首先检查所需页面是否在空闲帧池中。如果不在，我们应选择一个自由帧并读入页面。

这种技术与 FIFO 置换算法一起用于 VAX/VMS 系统中。当 FIFO 置换算法错误地置换了一个常用页面时，该页面可从空闲帧池中很快取出，而且不需要 I/O。这种空闲帧缓冲弥补了相对差但却简单的 FIFO 置换算法。这种方法是必要的，因为早期版本的 VAX 没有正确实现引用位。

有些版本的 UNIX 系统将此方法与第二次机会算法一起使用。这种方法可用来改进任何页面置换算法，以降低因错误选择牺牲页面而引起的开销。

9.4.8　应用程序与页面置换

在某些情况下，通过操作系统的虚拟内存访问数据的应用程序比操作系统根本没有提供缓冲区更差。一个典型的例子是数据库，它提供自己的内存管理和 I/O 缓冲。类似这样的程序比提供通用目的算法的操作系统，更能理解自己的内存使用与磁盘使用。如果操作系统提供 I/O 缓冲而应用程序也提供 I/O 缓冲，那么用于这些 I/O 的内存自然就成倍了。

另一个例子是数据仓库，它频繁地执行大量的、顺序的磁盘读取，随后计算并写入。LRU 算法会删除旧的页面并保留新的页面，而应用程序将更可能读取较旧的页面而不是较新的页面（因为它再次开始顺序读取）。这里，MFU 可能比 LRU 更为高效。

由于这些问题，有的操作系统允许特殊程序能够将磁盘分区作为逻辑块的大的数组来使用，而不需要通过文件系统的数据结构。这种数组有时称为**原始磁盘**（raw disk），而这种数组的 I/O 称为原始 I/O。原始 I/O 绕过所有文件系统服务，例如文件 I/O 的请求调页、文件锁定、预取、空间分配、文件名和目录等。请注意，尽管有些应用程序在原始分区上实现自己的专用存储服务更加高效，但是大多数应用程序采用通用文件系统服务更好。

9.5　帧分配

接下来讨论分配问题。在各个进程之间，如何分配固定数量的可用内存？如果有 93 个空闲帧和 2 个进程，那么每个进程各有多少帧？