况下，可能会从该进程的资源中剥夺部分页框。这样，PFF尽力让每个进程的缺页中断率控制在可接受的范围内。

值得注意的是，一些页面置换算法既适用于局部置换算法，又适用于全局置换算法。例如，FIFO能够将所有内存中最老的页面置换掉（全局算法），也能将当前进程的页面中最老的替换掉（局部算法）。相似地，LRU或是一些类似算法能够将所有内存中最近最少访问的页框替换掉（全局算法），或是将当前进程中最近最少使用的页框替换掉（局部算法）。在某些情况下，选择局部策略还是全局策略是与页面置换算法无关的。

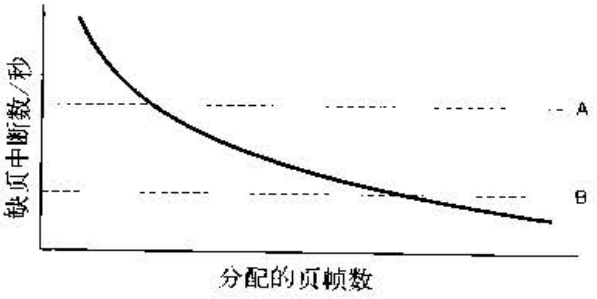

图3-24 缺页中断率是分配的页框数的函数

另一方面，对于其他的页面置换算法，只有采用局部策略才有意义。特别是工作集和WSClock算法是针对某些特定进程的而且必须应用在这些进程的上下文中。实际上没有针对整个机器的工作集，并且试图使用所有工作集的并集作为机器的工作集可能会丢失一些局部特性，这样算法就不能得到好的性能。

3.5.2 负载控制

即使是使用最优页面置换算法并对进程采用理想的全局页框分配，系统也可能会发生颠簸。事实上，一旦所有进程的组合工作集超出了内存容量，就可能发生颠簸。该现象的症状之一就是如PFF算法所指出的，一些进程需要更多的内存，但是没有进程需要更少的内存。在这种情况下，没有方法能够在不影响其他进程的情况下满足那些需要更多内存的进程的需要。惟一现实的解决方案就是暂时从内存中去掉一些进程。

减少竞争内存的进程数的一个好方法是将一部分进程交换到磁盘，并释放他们所占有的所有页面。例如，一个进程可以被交换到磁盘，而它的页框可以被其他处于颠簸状态的进程分享。如果颠簸停止，系统就能够这样运行一段时间。如果颠簸没有结束，需要继续将其他进程交换出去，直到颠簸结束。因此，即使是使用分页，交换也是需要的，只是现在交换是用来减少对内存潜在的需求，而不是收回它的页面。

将进程交换出去以减轻内存需求的压力是借用了两级调度的思想，在此过程中一些进程被放到磁盘，此时用一个短期的调度程序来调度剩余的进程。很明显，这两种思路可以被组合起来，将恰好足够的进程交换出去以获取可接受的缺页中断率。一些进程被周期性地从磁盘调入，而其他一些则被周期性地交换到磁盘。

不过，另一个需要考虑的因素是多道程序设计的道数。当内存中的进程数过低的时候，CPU可能在很长的时间内处于空闲状态。考虑到该因素，在决定交换出哪个进程时不光要考虑进程大小和分页率，还要考虑它的特性（如它究竟是CPU密集型还是I/O密集型）以及其他进程的特性。

3.5.3 页面大小

页面大小是操作系统可以选择的一个参数。例如，即使硬件设计只支持512字节的页面，操作系统也可以很容易通过总是为页面对0和1、2和3、4和5等分配两个连续的512字节的页框，而将其作为1KB的页面。

要确定最佳的页面大小需要在几个互相矛盾的因素之间进行权衡。从结果看，不存在全局最优。首先，有两个因素可以作为选择小页面的理由。随便选择一个正文段、数据段或堆栈段很可能不会恰好装满整数个页面，平均的情况下，最后一个页面中有一半是空的。多余的空间就被浪费掉了，这种浪费称为内部碎片（internal fragmentation）。在内存中有n个段、页面大小为p字节时，会有$np/2$字节被内部碎片浪费。从这方面考虑，使用小页面更好。

选择小页面还有一个明显的好处，如果考虑一个程序，它分成8个阶段顺序执行，每阶段需要4KB内存。如果页面大小是32KB，那就必须始终给该进程分配32KB内存。如果页面大小是16KB，它就只需要16KB。如果页面大小是4KB或更小，在任何时刻它只需要4KB内存。总的来说，与小页面相比，大页面使更多没有用的程序保留在内存中。

在另一方面，页面小意味着程序需要更多的页面，这又意味着需要更大的页表。一个32KB的程序只需要4个8KB的页面，却需要64个512字节的页面。内存与磁盘之间的传输一般是一次一页，传输中的大部分时间都花在了寻道和旋转延迟上，所以传输一个小的页面所用的时间和传输一个大的页面基本上