却无法看到这些。在实践中，如果时钟滴答是20ms，8位一般是够用的。假如一个页面已经有160ms没有被访问过，那么它很可能并不重要。

3.4.8 工作集页面置换算法

在单纯的分页系统里，刚启动进程时，在内存中并没有页面。在CPU试图取第一条指令时就会产生一次缺页中断，使操作系统装入含有第一条指令的页面。其他由访问全局数据和堆栈引起的缺页中断通常会紧接着发生。一段时间以后，进程需要的大部分页面都已经在内存了，进程开始在较少缺页中断的情况下运行。这个策略称为请求调页（demand paging），因为页面是在需要时被调入的，而不是预先装入。

编写一个测试程序很容易，在一个大的地址空间中系统地读所有的页面，将出现大量的缺页中断，因此会导致没有足够的内存来容纳这些页面。不过幸运的是，大部分进程不是这样工作的，它们都表现出了一种局部性访问行为，即在进程运行的任何阶段，它都只访问较少的一部分页面。例如，在一个多次扫描编译器中，各次扫描时只访问所有页面中的一小部分，并且是不同的部分。

一个进程当前正在使用的页面的集合称为它的工作集（working set）（Denning，1968a；Denning，1980）。如果整个工作集都被装入到了内存中，那么进程在运行到下一运行阶段（例如，编译器的下一遍扫描）之前，不会产生很多缺页中断。若内存太小而无法容纳下整个工作集，那么进程的运行过程中会产生大量的缺页中断，导致运行速度也会变得很缓慢，因为通常只需要几个纳秒就能执行完一条指令，而通常需要十毫秒才能从磁盘上读入一个页面。如果一个程序每10ms只能执行一到两条指令，那么它将会需要很长时间才能运行完。若每执行几条指令程序就发生一次缺页中断，那么就称这个程序发生了颠簸（thrashing）（Denning，1968b）。

在多道程序设计系统中，经常会把进程转移到磁盘上（即从内存中移走所有的页面），这样可以让其他的进程有机会占有CPU。有一个问题是，当该进程再次调回来以后应该怎样办？从技术的角度上讲，并不需要做什么。该进程会一直产生缺页中断直到它的工作集全部被装入内存。然而，每次装入一个进程时都要产生20、100甚至1000次缺页中断，速度显然太慢了，并且由于CPU需要几毫秒时间处理一个缺页中断，因此有相当多的CPU时间也被浪费了。

所以不少分页系统都会设法跟踪进程的工作集，以确保在让进程运行以前，它的工作集就已在内存中了。该方法称为工作集模型（working set model）（Denning，1970），其目的在于大大减少缺页中断率。在让进程运行前预先装入其工作集页面也称为预先调页（prepaging）。请注意工作集是随着时间变化的。

人们很早就发现大多数程序都不是均匀地访问它们的地址空间的，而访问往往是集中于一小部分页面。一次内存访问可能会取出一条指令，也可能会取数据，或者是存储数据。在任一时刻t，都存在一个集合，它包含所有最近k次内存访问所访问过的页面。这个集合$w(k,t)$就是工作集。因为最近$k=1$次访问肯定会访问最近$k>1$次访问所访问过的页面，所以$w(k,t)$是k的单调非递减函数。随着k的变大，$w(k,t)$是不会无限变大的，因为程序不可能访问比它的地址空间所能容纳的页面数目上限还多的页面，并且几乎没有程序会使用每个页面。图3-19描述了作为k的函数的工作集的大小。

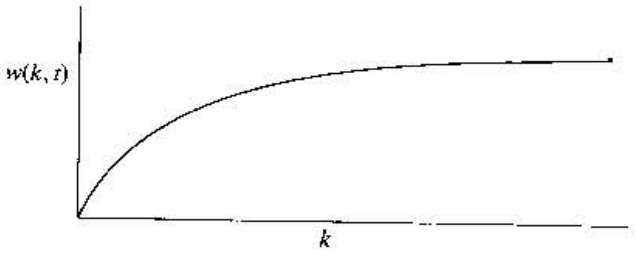

图3-19 工作集是最近k次内存访问所访问过的页面的集合，函数$w(k,t)$是在时刻t时工作集的大小

事实上大多数程序会任意访问一小部分页面，但是这个集合会随着时间而缓慢变化，这个事实也解释了为什么一开始曲线快速地上升而k较大时上升会变慢。举例来说，某个程序执行占用了两个页面的循环，并使用四个页面上的数据，那么可能每执行1000条指令，它就会访问这六个页面一次，但是最近的对其他页面的访问可能是在100万条指令以前的初始化阶段。因为这是个渐进的过程，k值的选择对工作集的内容影响不大。换句话说，k的值有一个很大的范围，它处在这个范围中时工作集不会变。因为工作集随时间变化很慢，那么当程序重新开始时，就有可能根据它上次结束时的工作集对要用到的页面做一个合理的推测，预先调页就是在程序继续运行之前预先装入推测出的工作集的页面。

为了实现工作集模型，操作系统必须跟踪哪些页面在工作集中。通过这些信息可以直接推导出一个