当必须查找一个路径时，名字解析器首先查阅高速缓存并搜索它以找到高速缓存中存在的最长的子字符串。例如，如果存在路径/usr/ast/grants/stw，高速缓存会返回/usr/ast/是i节点26这样的事实，这样搜索就可以从这里开始，消除了四次磁盘访问。

对路径进行高速缓存存在的一个问题是，文件名与i节点号之间的映射并不总是固定的。假设文件/usr/ast/mbox从系统中被删除，并且其i节点重用于不同用户所拥有的不同的文件。随后，文件/usr/ast/mbox再次被创建，并且这一次它得到i节点106。如果不对这件事情进行预防，高速缓存项现在将是错误的，并且后来的查找将返回错误的i节点号。为此，当一个文件或目录被删除时，它的高速缓存项以及（如果它是一个目录的话）它下面所有的项都必须从高速缓存中清除。

磁盘块与路径名并不是能够高速缓存的惟一项目，i节点也可以被高速缓存。如果弹出的线程用来处理中断，每个这样的线程需要一个栈和某些附加的机构。这些以前用过的线程也可以被高速缓存，因为刷新一个用过的线程比从头创建一个新的线程更加容易（为了避免必须分配内存）。难于生产的任何事物几乎都能够被高速缓存。

### 13.4.5 线索

高速缓存项总是正确的。高速缓存搜索可能失败，但是如果找到了一项，那么这一项保证是正确的并且无需再费周折就可以使用。在某些系统中，包含线索（hint）的表是十分便利的。这些线索是关于答案的暗示，但是它们并不保证是正确的。调用者必须自行对结果进行验证。

众所周知的关于线索的例子是嵌在Web页上的URL。点击一个链接并不能保证被指向的Web页就在那里。事实上，被指向的网页可能10年前就被删除了。因此包含URL的网页上面的信息只是一个线索。

线索还用于连接远程文件。信息是提示有关远程文件某些事项的线索，例如文件存放的位置。然而，自该线索被记录以来，文件可能已经被移动或者被删除了，所以为了明确线索是否正确，总是需要进行检查。

### 13.4.6 利用局部性

进程和程序的行为并不是随机的，它们在时间上和空间上展现出相当程度的局部性，并且可以以各种方式利用该信息来改进性能。空间局部性的一个常见例子是这样的事实：进程并不是在其地址空间内部随机地到处跳转的。在一个给定的时间间隔内，它们倾向于使用数目比较少的页面。进程正在有效地使用的页面可以被标记为它的工作集，并且操作系统能够确保当进程被允许运行时，它的工作集在内存中，这样就减少了缺页的次数。

局部化原理对于文件也是成立的。当一个进程选择了一个特定的工作目录时，很可能将来许多文件引用将指向该目录中的文件。通过在磁盘上将每个目录的所有i节点和文件就近放在一起，可能会获得性能的改善。这一原理正是Berkeley快速文件系统的基础（McKusick等人，1984）。

局部性起作用的另一个领域是多处理器系统中的线程调度。正如我们在第8章中看到的，在多处理器上一种调度线程的方法是试图在最后一次用过的CPU上运行每个线程，期望它的某些内存块依然还在内存的高速缓存中。

### 13.4.7 优化常见的情况

区分最常见的情况和最坏可能的情况并且分别处理它们，这通常是一个好主意。针对这两者的代码常常是相当不同的。重要的是要使常见的情况速度快。对于最坏的情况，如果它很少发生，使其正确就足够了。

第一个例子，考虑进入一个临界区。在大多数时间中，进入将是成功的，特别是如果进程在临界区内部不花费很多时间的话。Windows Vista提供的一个Win32 API调用EnterCriticalSection就利用了这一期望，它自动地在用户态测试一个标志（使用TSL或等价物）。如果测试成功，进程只是进入临界区并且不需要内核调用。如果测试失败，库过程将调用一个信号量上的down操作以阻塞进程。因此，在通常情况下是不需要内核调用的。

第二个例子，考虑设置一个警报（在UNIX中使用信号）。如果当前没有警报待完成，那么构造一个警报并且将其放在定时器队列上是很简单的。然而，如果已经有一个警报待完成，那么就必须找到它并且从定时器队列中删除。由于alarm调用并未指明是否已经设置了一个警报，所以系统必须假设最坏的