点是文件名不再需要从字的边界开始，这样，原先在图4-15a中需要的填充字符，在图4-15b中的文件名之后就不再需要了。

到目前为止，在需要查找文件名时，所有的方案都是线性地从头到尾对目录进行搜索。对于非常长的目录，线性查找就太慢了。加快查找速度的一个方法是在每个目录中使用散列表。设表的大小为n。在输入文件名时，文件名被散列到1和$n-1$之间的一个值，例如，它被n除，并取余数。其他可以采用的方法有，对构成文件名的字求和，其结果被n除，或某些类似的方法。

不论哪种方法都要对与散列码相对应的散列表表项进行检查。如果该表项没有被使用，就将一个指向文件目录项的指针放入，文件目录项紧连在散列表后面。如果该表项被使用了，就构造一个链表，该链表的表头指针存放在该表项中，并链接所有具有相同散列值的文件目录项。

查找文件按照相同的过程进行。散列处理文件名，以便选择一个散列表项。检查链表头在该位置上的所有表项，查看要找的文件名是否存在。如果名字不在该链上，该文件就不在这个目录中。

使用散列表的优点是查找非常迅速。其缺点是需要复杂的管理。只有在预计系统中的目录经常会有成百上千个文件时，才把散列方案真正作为备用方案考虑。

一种完全不同的加快大型目录查找速度的方法是，将查找结果存入高速缓存。在开始查找之前，先查看文件名是否在高速缓存中。如果是，该文件可以立即定位。当然，只有在构成查找主体的文件非常少的时候，高速缓存的方案才有效果。

4.3.4 共享文件

当几个用户同在一个项目里工作时，他们常常需要共享文件。其结果是，如果一个共享文件同时出现在属于不同用户的不同目录下，工作起来就很方便。图4-16再次给出图4-7所示的文件系统，只是C的一个文件现在也出现在B的目录下。B的目录与该共享文件的联系称为一个连接（link）。这样，文件系统本身是一个有向无环图（Directed Acyclic Graph，DAG）而不是一棵树。

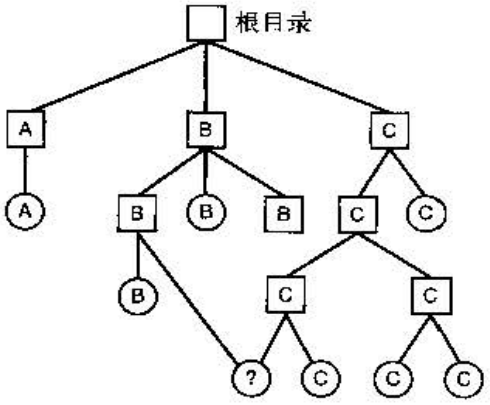

图4-16 有共享文件的文件系统

共享文件是方便的，但也带来一些问题。如果目录中包含磁盘地址，则当连接文件时，必须把C目录中的磁盘地址复制到B目录中。如果B或C随后又往该文件中添加内容，则新的数据块将只列入进行添加工作的用户的目录中。其他的用户对此改变是不知道的。所以违背了共享的目的。

有两种方法可以解决这一问题。在第一种解决方案中，磁盘块不列入目录，而是列入一个与文件本身关联的小型数据结构中。目录将指向这个小型数据结构。这是UNIX系统中所采用的方法（小型数据结构即是i节点）。

在第二种解决方案中，通过让系统建立一个类型为LINK的新文件，并把该文件放在B的目录下，使得B与C的一个文件存在连接。新的文件中只包含了它所连接的文件的路径名。当B读该连接文件时，操作系统查看到要读的文件是LINK类型，则找到该文件所连接的文件的名字，并且去读那个文件。与传统（硬）连接相对比起来，这一方法称为符号连接（symbolic linking）。

以上每一种方法都有其缺点。第一种方法中，当B连接到共享文件时，i节点记录文件的所有者是C。建立一个连接并不改变所有关系（见图4-17），但它将i节点的连接计数加1，所以系统知道目前有多少目录项指向这个文件。

如果以后C试图删除这个文件，系统将面临问题。如果系统删除文件并清除i节点，B则有一个目录项指向一个无效的i节点。如果该i节点以后分配给另一个文件，则B的连接指向一个错误的文件。系统通过i节点中的计数可知该文件仍然被引用，但是没有办法找到指向该文件的全部目录项以删除它们。指向目录的指针不能存储在i节点中，原因是有可能有无数个目录。

惟一能做的就是只删除C的目录项，但是将i节点保留下来，并将计数置为1，如图4-17c所示。而现在的状况是，只有B有指向该文件的目录项，而该文件的所有者是C。如果系统进行记账或有配额，那么C将继续为该文件付账直到B决定删除它，如果真是这样，只有到计数变为0的时刻，才会删除该文件。