VFS也有一个对于实际文件系统的“更低层”接口，就是在图4-18中被标记为VFS接口的部分。这个接口包含许多功能调用，这样VFS可以使每一个文件系统完成任务。因此，当创造一个新的文件系统和VFS一起工作时，新文件系统的设计者就必须确定它提供VFS所需要的功能调用。关于这个功能的一个明显的例子就是从磁盘中读某个特定的块，把它放在文件系统的高速缓冲中，并且返回指向它的指针。因此，VFS有两个不同的接口：上层给用户进程的接口和下层给实际文件系统的接口。

尽管VFS下大多数的文件系统体现了本地磁盘的划分，但并不总是这样。事实上，Sun建立虚拟文件系统最原始的动机是支持使用NFS（Network File System，网络文件系统）协议的远程文件系统。VFS设计是只要实际的文件系统提供VFS需要的功能，VFS就不需知道或者关心数据具体存储在什么地方或者底层的文件系统是什么样的。

大多数VFS应用本质上都是面向对象的，即便它们用C语言而不是C++编写。有几种通常支持的主要的对象类型，包括超块（描述文件系统）、v节点（描述文件）和目录（描述文件系统目录）。这些中的每一个都有实际文件系统必须支持的相关操作。另外，VFS有一些供它自己使用的内部数据结构，包括用于跟踪用户进程中所有打开文件的装载表和文件描述符的数组。

为了理解VFS是如何工作的，让我们按时间的先后举一个例子。当系统启动时，根文件系统在VFS中注册。另外，当装载其他文件系统时，不管在启动时还是在操作过程中，它们也必须在VFS中注册。当一个文件系统注册时，它做的最基本的工作就是提供一个包含VFS所需要的函数地址的列表，可以是一个长的调用矢量（表），或者是许多这样的矢量（如果VFS需要），每个VFS对象一个。因此，只要一个文件系统在VFS注册，VFS就知道如何从它那里读一个块——它从文件系统提供的矢量中直接调用第4个（或者任何一个）功能。同样地，VFS也知道如何执行实际文件系统提供的每一个其他的功能：它只需调用某个功能，该功能所在的地址在文件系统注册时就提供了。

装载文件系统后就可以使用它了。比如，如果一个文件系统装载在/usr并且一个进程调用它：

```
open("/usr/include/unistd.h", O_RDONLY)
```

当解析路径时，VFS看到新的文件系统被装载在/usr，并且通过搜索已经装载文件的超块表来确定它的超块。做完这些，它可以找到它所装载的文件的根目录，在那里查找路径include/unistd.h。然后VFS创建一个v节点并调用实际文件系统，以返回所有的在文件i节点中的信息。这个信息被和其他信息一起复制到v节点中（在RAM中），而这些信息中最重要的是指向包含调用v节点操作的功能表的指针，比如read、write和close等。

当v节点被创建以后，VFS在文件描述符表中为调用进程创建一个入口，并且将它指向一个新的v节点（为了简单，文件描述符实际上指向另一个包含当前文件位置和指向v节点的指针的数据结构，但是这个细节对于我们这里的陈述并不重要）。最后，VFS向调用者返回文件描述符，所以调用者可以用它去读、写或者关闭文件。

随后，当进程用文件描述符进行一个读操作，VFS通过进程表和文件描述符表确定v节点的位置，并跟随指针指向功能表（所有这些都是被请求文件所在的实际文件系统中的地址）。这样就调用了处理read的功能，在实际文件系统中的代码运行并得到所请求的块。VFS并不知道数据是来源于本地硬盘，还是来源于网络中的远程文件系统、CD-ROM、USB存储棒或者其他介质。所有有关的数据结构在图4-19中展示。从调用者进程号和文件描述符开始，进而是v节点，读功能指针，然后是对实际文件系统的入口函数定位。

通过这种方法，加入新的文件系统变得相当直接。为了加入一个文件系统，设计者首先获得一个VFS期待的功能调用的列表，然后编写文件系统实现这些功能。或者，如果文件系统已经存在，它们必须提供VFS需要的包装功能，通常通过建造一个或者多个内在的指向实际文件系统的调用来实现。