参数传递。在Linux系统中添加一个新的设备类型，意味着要向这些表添加一个新的表项，并提供相应的函数来处理此设备上的各种操作。

图10-21展示了一部分可以跟不同的字符设备关联的操作。每一行指向一个单独的I/O设备（即一个单独的驱动程序）。列表示所有的字符驱动程序必须支持的功能。还有几个其他的功能。当一个操作要在一个字符特殊文件上执行时，系统通过检索字符设备的散列表来选择合适的数据结构，然后调用相应的功能来执行此操作。因此，每个文件操作都包含指向相应驱动程序的一个函数指针。

设备	Open	Close	Read	Write	Ioctl	其他
Null	null	null	null	null	null	…
内存	null	null	mem_read	mem_write	null	…
键盘	k_open	k_close	k_read	error	k_ioctl	…
Tty	tty_open	tty_close	tty_read	tty_write	tty_ioctl	…
打印机	lp_open	lp_close	error	lp_write	lp_ioctl	…

图10-21 典型字符设备支持的部分文件操作

每个驱动程序都分为两部分。这两部分都是Linux内核的一部分，并且都运行在内核态。上半部分运行在调用者的上下文并且与Linux其他部分交互。下半部分运行在内核上下文并且与设备进行交互。驱动程序可以调用内存分配、定时器管理、DMA控制等内核过程。所有可以被调用的内核功能都定义在一个叫做*驱动程序－内核接口*（Driver-Kernel Interface）的文档中。编写Linux设备驱动的细节请参见文献（Egan和Teixeira，1992；Rubini等人，2005）。

I/O系统被划分为两大部分：处理块特殊文件的部分和处理字符特殊文件的部分。下面将依次讨论这两部分。

系统中处理块特殊文件（比如，磁盘）I/O的部分的目标是使必须要完成的传输次数最小。为了实现这个目标，Linux系统在磁盘驱动程序和文件系统之间放置了一个*高速缓存*（cache），如图10-22。在2.2版本内核之前，Linux系统完整地维护着两个单独的缓存：*页面缓存*（page cache）和*缓冲器缓存*（buffer cache），因此，存储在一个磁盘块中的文件可能会被缓存在两个缓存中。2.2版本以后的Linux内核版本只有一个统一的缓存。一个*通用数据块层*（generic block layer）把这些组件整合在了一起，执行磁盘扇区、数据块、缓冲区和数据页面之间必要的转换，并且激活作用于这些结构上的操作。

cache是内核里面用来保存数以千计的最近使用的数据块的表。不管本着什么样的目的（i节点，目录或数据）而需要一个磁盘块，系统首先检查这个块是否在cache里面。如果在cache中，就可以从cache里直接得到这个块，从而避免了一次磁盘访问，这可以在很大程度上提高系统性能。

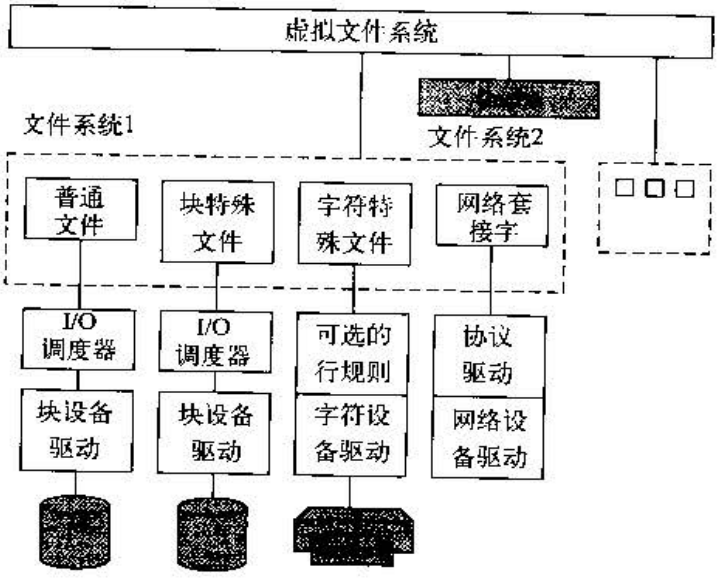

图10-22 Linux I/O系统中一个文件系统的细节