如果页面cache中没有这个块，系统就会从磁盘中把这个块读入到cache中，然后再从cache中复制到请求它的地方。由于页面cache的大小是固定的，因此，前面章节介绍的页面置换算法在这里也是需要的。

页面cache也支持写数据块，就像读数据一样。一个程序要回写一个块时，它被写到cache里，而不是直接写到磁盘上。当cache增长到超过一个指定值时，pdflush守护进程会把这个块写回到磁盘上。另外，为了防止数据块被写回到磁盘之前在cache里存留太长时间，每隔30秒系统会把所有的“脏块”都写回到磁盘上。

Linux依靠一个I/O调度器来保证磁头反复移动的延迟最小。I/O调度器的作用是对块设备的读写请求重新排序或对这些读写请求进行合并。有很多调度器变种，它们是根据不同类型的工作负载进行优化的结果。基本的Linux I/O调度器基于最初的Linus电梯调度器（Linus Elevator scheduler）。电梯调度器的操作可以这样总结：按磁盘请求的扇区地址的顺序将磁盘操作在一个双向链表中排序。新的请求以排序的方式插入到双向链表中。这种方法可以有效地防止磁头重复移动。请求列表经过合并后，相邻的操作会被整合为一条单独的磁盘请求。基本电梯调度器有一个问题是会导致饥饿的情况发生。因此，Linux磁盘调度器的修改版本包括两个附加的列表，维护按时限（deadline）排序的读写操作。读请求的缺省时限是0.5s，写请求的缺省时限是5s。如果最早的写操作的系统定义的时限要过期了，那么相对于任何在主双向链表中的请求来说，这个写请求会被优先服务。

除了正常的磁盘文件，还有其他的块特殊文件，也被称为原始块文件（raw block file）。这些文件允许程序通过绝对块号来访问磁盘，而不考虑文件系统。它们通常被用于分页和系统维护。

与字符设备的交互是很简单的。因为字符设备产生和接收的是字符流或字节数据，所以让字符设备支持随机访问是几乎没有意义的。不过行规则（line disciplines）的使用是个例外。一个行规则可以和一个终端设备联合在一起，通过tty_struct结构来表示，一般作为和终端交换的数据的解释器。例如，利用行规则可以完成本地行编辑（即擦除的字符和行可以被删除），回车可以映射为换行，以及其他的特殊处理能够被完成。然而，如果一个进程要跟每个字符交互，那么它可以把行设置为原始模式，此时行规则将被忽略。另外，并不是所有的设备都有行规则。

输出采用与输入类似的工作方式，如把tab扩展为空格，把换行转变为回车＋换行，在慢的机械式终端的回车后面加填充字符等。像输入一样，输出可以通过（加工模式）行规则，或者忽略（原始模式）行规则。原始模式对于GUI和通过一个串行数据线发送二进制数据到其他的计算机的情况尤其有用，因为这些情况都不需要进行转换。

和网络设备的交互与前面的讨论有些不同。虽然网络设备也是产生或者接收字符流，但是它们的异步特性使得它们并不适合与其他的字符设备统一使用相同的接口。网络设备驱动程序产生具有多个字节的数据包和网络头。接着，这些包会经过一连串的网络协议驱动程序传送，最后被发送到用户空间应用程序。套接字缓冲区，skbuff，是一个关键的数据结构，它用来表示填有包数据的部分内存。skbuff缓冲区里面的数据并不总是始于缓冲区的开始位置，因为它们被网络栈中的不同协议处理过，可能会添加或删除协议头。用户进程通过套接字与网络设备进行交互，在Linux中支持原始的BSD的套接字API。通过raw_sockets，协议驱动程序可以被忽略，从而可以实现对底层网络设备的直接访问。只有超级用户才可以创建原始套接字（raw socket）。

10.5.5 Linux中的模块

几十年来，UNIX设备驱动程序是被静态链接到内核中的。因此，只要系统启动，设备驱动程序都会被加载到内存中。在UNIX比较成熟的环境中，如大部分的部门小型计算机以及高端的工作站，其共同的特点是I/O设备集都较小并且稳定不变，这种模式工作得很好。基本上，一个计算机中心会构造一个包含I/O设备驱动程序的内核，并且一直使用它。如果第二年，这个中心买了一个新的磁盘，那么重新链接内核就可以了。一点问题也没有。

随着个人电脑平台Linux系统的到来，所有这些都改变了。相对于任何一台小型机上的I/O设备，PC机上可用I/O设备的数量都有了数量级上的增长。另外，虽然所有的Linux用户都有（或者很容易得到）Linux源代码，但是绝大部分用户都没有能力去添加一个新的驱动程序、更新所有的设备驱动程序数据结构、重链接内核，然后把它作为可启动的系统进行安装（更不用提要处理构造完成后内核不能启动的问题）。