个未读字节。如果缓冲区为空，那么会通过调用 read 再填满它。这个 read 调用收到一个不足值并不是错误，只不过读缓冲区是填充了一部分。一旦缓冲区非空，rio_read 就从读缓冲区复制 n 和 rp->rio_cnt 中较小值个字节到用户缓冲区，并返回复制的字节数。

code/src/csapp.c

```
1   static ssize_t rio_read(rio_t *rp, char *usrbuf, size_t n)
2   {
3       int cnt;
4
5       while (rp->rio_cnt <= 0) {  /* Refill if buf is empty */
6           rp->rio_cnt = read(rp->rio_fd, rp->rio_buf,
7                              sizeof(rp->rio_buf));
8           if (rp->rio_cnt < 0) {
9               if (errno != EINTR) /* Interrupted by sig handler return */
10                  return -1;
11          }
12          else if (rp->rio_cnt == 0)  /* EOF */
13              return 0;
14          else
15              rp->rio_bufptr = rp->rio_buf; /* Reset buffer ptr */
16      }
17
18      /* Copy min(n, rp->rio_cnt) bytes from internal buf to user buf */
19      cnt = n;
20      if (rp->rio_cnt < n)
21          cnt = rp->rio_cnt;
22      memcpy(usrbuf, rp->rio_bufptr, cnt);
23      rp->rio_bufptr += cnt;
24      rp->rio_cnt -= cnt;
25      return cnt;
26  }
```

code/src/csapp.c

图 10-7　内部的 rio_read 函数

对于一个应用程序，rio_read 函数和 Linux read 函数有同样的语义。在出错时，它返回值−1，并且适当地设置 errno。在 EOF 时，它返回值 0。如果要求的字节数超过了读缓冲区内未读的字节的数量，它会返回一个不足值。两个函数的相似性使得很容易通过用 rio_read 代替 read 来创建不同类型的带缓冲的读函数。例如，用 rio_read 代替 read，图 10-8 中的 rio_readnb 函数和 rio_readn 有相同的结构。相似地，图 10-8 中的 rio_readlineb 程序最多调用 maxlen-1 次 rio_read。每次调用都从读缓冲区返回一个字节，然后检查这个字节是否是结尾的换行符。

旁注　RIO 包的起源

RIO 函数的灵感来自于 W. Richard Stevens 在他的经典网络编程作品[110]中描述的 readline、readn 和 writen 函数。rio_readn 和 rio_writen 函数与 Stevens 的 readn 和 writen 函数是一样的。然而，Stevens 的 readline 函数有一些局限性在 RIO 中得到了纠正。第一，因为 readline 是带缓冲的，而 readn 不带，所以这两个函数不能在同一描述符上一起使用。第二，因为它使用一个 static 缓冲区，Stevens 的 readline