read 函数从描述符为 fd 的当前文件位置复制最多 n 个字节到内存位置 buf。返回值−1 表示一个错误，而返回值 0 表示 EOF。否则，返回值表示的是实际传送的字节数量。

write 函数从内存位置 buf 复制至多 n 个字节到描述符 fd 的当前文件位置。图 10-3 展示了一个程序使用 read 和 write 调用一次一个字节地从标准输入复制到标准输出。

code/io/cpstdin.c

```
1   #include "csapp.h"
2
3   int main(void)
4   {
5       char c;
6
7       while(Read(STDIN_FILENO, &c, 1) != 0)
8           Write(STDOUT_FILENO, &c, 1);
9       exit(0);
10  }
```

code/io/cpstdin.c

图 10-3　一次一个字节地从标准输入复制到标准输出

通过调用 lseek 函数，应用程序能够显示地修改当前文件的位置，这部分内容不在我们的讲述范围之内。

旁注　ssize_t 和 size_t 有些什么区别？

你可能已经注意到了，read 函数有一个 size_t 的输入参数和一个 ssize_t 的返回值。那么这两种类型之间有什么区别呢？在 x86-64 系统中，size_t 被定义为 unsigned long，而 ssize_t(有符号的大小)被定义为 long。read 函数返回一个有符号的大小，而不是一个无符号大小，这是因为出错时它必须返回−1。有趣的是，返回一个−1 的可能性使得 read 的最大值减小了一半。

在某些情况下，read 和 write 传送的字节比应用程序要求的要少。这些不足值(short count)不表示有错误。出现这样情况的原因有：

- 读时遇到 EOF。假设我们准备读一个文件，该文件从当前文件位置开始只含有 20 多个字节，而我们以 50 个字节的片进行读取。这样一来，下一个 read 返回的不足值为 20，此后的 read 将通过返回不足值 0 来发出 EOF 信号。
- 从终端读文本行。如果打开文件是与终端相关联的(如键盘和显示器)，那么每个 read 函数将一次传送一个文本行，返回的不足值等于文本行的大小。
- 读和写网络套接字(socket)。如果打开的文件对应于网络套接字(11.4 节)，那么内部缓冲约束和较长的网络延迟会引起 read 和 write 返回不足值。对 Linux 管道(pipe)调用 read 和 write 时，也有可能出现不足值，这种进程间通信机制不在我们讨论的范围之内。

实际上，除了 EOF，当你在读磁盘文件时，将不会遇到不足值，而且在写磁盘文件时，也不会遇到不足值。然而，如果你想创建健壮的(可靠的)诸如 Web 服务器这样的网络应用，就必须通过反复调用 read 和 write 处理不足值，直到所有需要的字节都传送完毕。

10.5　用 RIO 包健壮地读写

在这一小节里，我们会讲述一个 I/O 包，称为 RIO(Robust I/O，健壮的 I/O)包，它