# 总结open与fopen的区别

`LINUX` `C语言基础`

---

参考链接1
参考链接2

对于这两个名字很类似的函数，对于很多初学者来说，不容易搞清楚它们有什么不同，只知道按照函数用法使用。如果能很好的区分两者，相信大家对于C语言和UNIX系统（包括LINUX）有更深入的了解。

在网上查找了一些资料，但是感觉不够全面，一些答案只是从某个角度阐述，所以让人觉得，这个也对，那个也对。但到底谁的表述更正确呢？其实都是对的，只是解释的视角不同罢了。下面结合个人的理解做一些梳理。

## 1. 来源

从来源的角度看，两者能很好的区分开，这也是两者最显而易见的区别：

- `open`是UNIX系统调用函数（包括LINUX等），返回的是文件描述符（File Descriptor），它是文件在文件描述符表里的索引。
- `fopen`是ANSIC标准中的C语言库函数，在不同的系统中应该调用不同的内核api。返回的是一个指向文件结构的指针。

  > PS:从来源来看，两者是有千丝万缕的联系的，毕竟C语言的库函数还是需要调用系统API实现的。

## 2. 移植性

这一点从上面的来源就可以推断出来，`fopen`是C标准函数，因此拥有良好的移植性；而`open`是UNIX系统调用，移植性有限。如windows下相似的功能使用API函数`CreateFile`。

## 3. 适用范围

- `open`返回文件描述符，而文件描述符是UNIX系统下的一个重要概念，UNIX下的一切设备都是以文件的形式操作。如网络套接字、硬件设备等。当然包括操作普通正规文件（Regular File）。
- fopen是用来操纵普通正规文件（Regular File）的。

## 4. 文件IO层次

如果从文件IO的角度来看，前者属于低级IO函数，后者属于高级IO函数。低级和高级的简单区分标准是：谁离系统内核更近。低级文件IO运行在内核态，高级文件IO运行在用户态。

## 5. 缓冲

1. 缓冲文件系统
   缓冲文件系统的特点是：在内存开辟一个“缓冲区”，为程序中的每一个文件使用；当执行读文件的操作时，从磁盘文件将数据先读入内存“缓冲区”，装满后再从内存“缓冲区”依此读出需要的数据。执行写文件的操作时，先将数据写入内存“缓冲区”，待内存“缓冲区”装满后再写入文件。由此可以看出，内存“缓冲区”的大小，影响着实际操作外存的次数，内存“缓冲区”越大，则操作外存的次数就少，执行速度就快、效率高。一般来说，文件“缓冲区”的大小随机器 而
   定。`fopen, fclose, fread, fwrite, fgetc, fgets, fputc, fputs, freopen, fseek, ftell, rewind`等。
2. 非缓冲文件系统
   缓冲文件系统是借助文件结构体指针来对文件进行管理，通过文件指针来对文件进行访问，既可以读写字符、字符串、格式化数据，也可以读写二进制数据。**非缓冲文件系统依赖于操作系统，通过操作系统的功能对文件进行读写，是系统级的输入输出**，它不设文件结构体指针，只能读写二进制文件，但效率高、速度快，由于ANSI标准不再包括非缓冲文件系统，因此建议大家最好不要选择它。`open, close, read, write, getc, getchar, putc, putchar`等。

一句话总结一下，就是`open`无缓冲，`fopen`有缓冲。前者与`read`，`write`等配合使用，后者与`fread`，`fwrite`等配合使用。

使用`fopen`函数，由于在用户态下就有了缓冲，因此进行文件读写操作的时候就减少了用户态和内核态的切换（切换到内核态调用还是需要调用系统调用API:`read`，`write`）；而使用`open`函数，在文件读写时则每次都需要进行内核态和用户态的切换；表现为，如果顺序访问文件，`fopen`**系列的函数**要比直接调用`open`系列的函数快；如果随机访问文件则相反。

这样一总结梳理，相信大家对于两个函数及系列函数有了一个更全面清晰的认识，也应该知道在什么场合下使用什么样的函数更合适，效率更高。