**linux系统调用和库函数调用的区别**

Linux下对文件操作有两种方式：系统调用（system call）和库函数调用（Library functions）。可以参考《Linux程序设计》（英文原版为《Beginning Linux Programming》，作者是Neil Matthew和Richard Stones）第三章: Working with files。系统调用实际上就是指最底层的一个调用，在linux程序设计里面就是底层调用的意思。面向的是硬件。而库函数调用则面向的是应用开发的，相当于应用程序的api,采用这样的方式有很多种原因。

第一：双缓冲技术的实现。第二，可移植性。第三，底层调用本身的一些性能方面的缺陷。第四：让api也可以有了级别和专门的工作面向。

1、系统调用

系统调用提供的函数如open, close, read, write, ioctl等，需包含头文件unistd.h。以write为例：其函数原型为 size\_t write(int fd, const void \*buf, size\_t nbytes)，其操作对象为文件描述符或文件句柄fd(file descriptor)，要想写一个文件，必须先以可写权限用open系统调用打开一个文件，获得所打开文件的fd，例如 fd=open(\"/dev/video\", O\_RDWR)。fd是一个整型值，每新打开一个文件，所获得的fd为当前最大fd加1。Linux系统默认分配了3个文件描述符值：0－standard input，1－standard output，2－standard error。

**系统调用通常用于底层文件访问（low-level file access），例如在驱动程序中对设备文件的直接访问。系统调用是操作系统相关的，因此一般没有跨操作系统的可移植性。**

**系统调用发生在内核空间，因此如果在用户空间的一般应用程序中使用系统调用来进行文件操作，会有用户空间到内核空间切换的开销。**事实上，即使在用户空间使用库函数来对文件进行操作，因为文件总是存在于存储介质上，因此不管是读写操作，都是对硬件（存储器）的操作，都必然会引起系统调用。也就是说，库函数对文件的操作实际上是通过系统调用来实现的。例如C库函数fwrite()就是通过write()系统调用来实现的。

这样的话，使用库函数也有系统调用的开销，为什么不直接使用系统调用呢？这是因为，读写文件通常是大量的数据（这种大量是相对于底层驱动的系统调用所实现的数据操作单位而言），这时，使用库函数就可以大大减少系统调用的次数。这一结果又缘于缓冲区技术。**在用户空间和内核空间，对文件操作都使用了缓冲区，例如用fwrite写文件，都是先将内容写到用户空间缓冲区，当用户空间缓冲区满或者写操作结束时，才将用户缓冲区的内容写到内核缓冲区，同样的道理，当内核缓冲区满或写结束时才将内核缓冲区内容写到文件对应的硬件媒介。**

2、库函数调用

标准C库函数提供的文件操作函数如fopen, fread, fwrite, fclose, fflush, fseek等，需包含头文件stdio.h。以fwrite为例，其函数原型为size\_t fwrite(const void \*buffer, size\_t size, size\_t item\_num, FILE \*pf)，其操作对象为文件指针FILE \*pf，要想写一个文件，必须先以可写权限用fopen函数打开一个文件，获得所打开文件的FILE结构指针pf，例如pf=fopen(\"~/proj/filename\", \"w\")。实际上，由于库函数对文件的操作最终是通过系统调用实现的，因此，每打开一个文件所获得的FILE结构指针都有一个内核空间的文件描述符fd与之对应。同样有相应的预定义的FILE指针：stdin－standard input，stdout－standard output，stderr－standard error。

**库函数调用通常用于应用程序中对一般文件的访问。库函数调用是系统无关的，因此可移植性好。由于库函数调用是基于C库的，因此也就不可能用于内核空间的驱动程序中对设备的操作。**

※函数库调用 VS 系统调用

