标准I/O库

流

雪山上的雪逐渐融化，形成一股股的小溪流，并最终汇聚在一起，变成了滚滚流动的江水

江水的流动是个非常连续的过程，除非遇到旱灾而断流，否则你永远不会见到: 上一秒有水流过，下一秒就没有了，再过一会儿又有了 ⊙ 这是流的一个非常重要的特性: 不成块

将上面的例子搬到计算机领域中，IO流其实就是数据从一个源头不间断地、连续地流向另一个目的地 ⊙ 水壶里面的开水倒到杯子里，这个过程不是一瞬间就完成的，而是流动的过程。同理，硬盘的数据传输至内存也不是一瞬间就完成的，而是逐渐地、连续地被搬运至内存

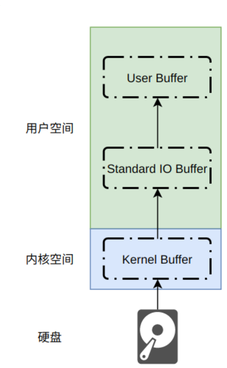
流的分类
- 字节流 ⊙ 流中的每一个字节我们认为它表示一个字符，通常用于ASNII字符集
- 字符流 ⊙ 流中的一个或者多个字节才能表示一个字符，通常用于国际字符集，例如UTF-8: 1个字节表示ASCII码，2~3个字节表示一个中文

有时也会称之为流的定向

流的定向
- 标准库调用: freopen() ⊙ 清除一个流的定向
- 标准库调用: fwide() ⊙ 设置流的定向

缓冲

在read(int fd, void *buffer, size_t count)系统调用中，buffer由用户提供，用于接收内核大小为count的数据 ⊙ 而在标准I/O库中，提供了一个位于用户空间的缓冲区作为用户读取数据的源头



分类
- 全缓冲 ⊙ 在填满标准I/O缓冲区后才进行实际I/O操作，磁盘文件I/O通常是全缓冲的 ⊙ 即当向磁盘写入数据时，通常会在标准缓冲区已满时，才会将数据写入磁盘(内核缓冲区)
- 行缓冲 ⊙ 当输入和输出遇到换行符时，标准I/O库执行I/O操作，终端设备通常是行缓冲的
- 无缓冲 ⊙ 标准I/O库不对字符进行缓冲存储，写入数据时，立即写入至内核缓冲区中，标准错误通常是不带缓冲的，这样用户能够更快的看到错误输出

标准库函数: fflush(FILE *fp) ⊙ 将fp所指向的流的缓冲数据强制刷新至内核

```c
int main(int *args, char **argv) {
    printf("This is printf out | ");

    fprintf(stderr, "This is a error output | ");

    printf("BiuBiu\n");

    printf("Hello Aean: ");

    char *s = "I would have written you a short letter. | ";
    write(STDOUT_FILENO, s, 43);

    return 0;
}
```

⊙

运行结果:
This is a error output | This is printf out | BiuBiu
I would have written you a short letter. | Hello Aean:

当我们混合使用系统调用与标准I/O库函数时，得到的输出结果顺序可能并不是我们需要的

打开流

函数调用
- FILE *fopen(const char *restrict pathname, const char *restrict type); ⊙ 打开路径名为pathname的文件
- FILE *freopen(const char *restrict pathname, const char *restrict type, FILE *restrict fp); ⊙ 把一个新的文件名pathname与给定的打开的流fp关联，同时关闭流中的旧文件
  - freopen("hello.txt", w+, stdout)，则会将原来输出至stdout的内容输出至hello.txt文件中
- FILE *fdopen(int fd, const char *restrict type); ⊙ 取一个已有的文件描述符，并使得一个标准的I/O流与该描述符相结合
  - 例如从open, dump, socket, accept等系统调用返回的描述符，可以使用fdopen()与标准I/O流结合

type参数
- 用于指定对该I/O流的读、写方式，相当于open()系统调用中的flags参数
- r或rb ⊙ 为读而打开 ⊙ O_RDONLY
- w或wb ⊙ 将文件截至0长，若文件不存在，则创建并写入 ⊙ O_WRONLY | O_CREAT | O_TRUNC
- a或ab ⊙ 向文件末尾追加内容，若文件不存在，则创建并写入 ⊙ O_WRONLY | O_CREAT | O_APPEND
- r+或rb+ ⊙ 读写模式 ⊙ O_RDWR
- w+或wb+ ⊙ 将文件截至0长，进行读写 ⊙ O_RDWR | O_CREAT | O_TRUNC
- a+或ab+ ⊙ 读取并以追加的方式写入 ⊙ O_RDWR | O_CREAT | O_APPEND

字符b表示binary，二进制，用于区分文本文件和二进制文件
但是在Linux中，不区分文本和二进制，所以可忽略

流的读取

相较于read()系统调用，标准I/O提供了更加灵活的数据读取方式 ⊙ 当使用read()读取带有中文的文件时，在utf-8编码下，中文占3~4个不等的字节数量。如果想要读取某一行，首先得计算出文件偏移量以及行的字节数(包括中文和ASCII码) ⊙ 说实话，这很麻烦

分类
- int fgetc(FILE *fp); ⊙ 每次读取一个字符(ASCII)，诸如中文等多字节文字，每次读取时则依次读取 ⊙ 例如"看"这个字符，fgetc需调用3次才能将其全部字节读出
  - 当读取到文件末尾或者出错时，返回EOF，可使用feof(FILE *fp)来判断是否是达到文件末尾，或者是ferror(FILE *fp)来判断是否出错
- char *fgets(char *restrict buf, int n, FILE *restrict fp); ⊙ 每次读取一行内容，但是必须指定最大行长度 ⊙ sizeof(buf)需大于等于n
  - 函数调用成功将返回buf，若到达文件末尾或者是出错时，将返回NULL ⊙ 可使用feof()或者是ferror()判断
- size_t fread(void *restrict ptr, size_t size, size_t nobj, FILE *restrict fp) ⊙ 二进制I/O, 读取一个保存在文件中的结构或者是数组，函数返回读取的对象数量