文件IO

无缓冲IO

1. 函数open：https://www.man7.org/linux/man-pages/man2/open.2.html

- 原型1

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
参数pathname：是文件的名字
参数flags：O_RDONLY 只读打开（o_rdonly）
            O_WRONLY 只写打开(o_wronly)
            O_RDWR 读写打开(o_rdwr)
```

- 原型2

```
int openat(int dirfd, const char *pathname, int flags);
int openat(int dirfd, const char *pathname, int flags, mode_t mode);
```

open和openat的区别：openat可以指定相对路径的文件

-

2. creat函数

```
通过 open(path,O_RDWR|O_CREAT|O_TRUNC)来达到creat函数的目的
```

3. close函数

```
int close(int fd);
```

- 会关闭记录锁
- 进程终止，内核会自动关闭文件对象（RAII）

4. lseek函数

```
 off_t lseek(int fd, off_t offset, int whence);

```

-

5. read函数

```
 ssize_t read(int fd, void *buf, size_t count);
 count：要读取的字节数量
```

```
buf：将读取到的数据存入buf中
返回值：读取到的字节数。如果读取到文件末尾，则返回0

```

6. write函数

```
ssize_t write(int fd, const void *buf, size_t count);
count:要写入的字节数
返回值：写入的字节数
```

7. pread函数（lseek+read）原子操作
8. pwrite函数（lseek+write）原子操作
9. sync函数：void sync()

fsync函数：**int fsync(int fd);**

fdatasync函数 同步数据： **int fdatasync(int fd);**

就是提交文件系统的缓存到磁盘上 8

10. dup函数和dup2函数：复制一个现有的文件描述符

```
int dup(int oldfd);
int dup2(int oldfd, int newfd);//close+dup
newfd指向oldfd
```

这两个函数的功能是复制文件描述符

- 不使用共享机制的时候，两个不同的进程操作同一个文件会造成数据覆盖的问题，这个时候需要借助这两个函数来避免这个问题。
- 详细可参考https://blog.csdn.net/u012351051/article/details/106307164

11. fcntl函数：

```
int fcntl(int fd, int cmd, ... /* arg */ );
```

- 复制一个已有的描述符
- 获取/设置文件描述符标志
- 获取/设置文件状态标志
- 获取/设置异步IO
- 获取设置记录锁

不同的功能使用不同的cmd参数：具体参考 man 2 fcntl

12. ioctl:设备驱动的通用接口

标准I/O（有缓冲io）

1. 流缓冲缓冲的类型：

- 全缓冲：填满IO缓冲区后才进行实际的操作
- 行缓冲：输入输出遇到换行符时执行I/O操作；由于每行缓冲区固定，所以一旦满了即使没有遇到换行符也会执行I/O操作；任何时候只要通过标准io库，要求从一个不带缓冲的流或者一个行缓冲的流（从内核请求数据的实时机）得

到输入数据，那么就会冲洗掉所有行缓冲输出流

2. stdin和stdou不指向交互设备时，是全缓冲类型，如果指向终端设备，则是行缓冲；stderr是不带缓冲
3. FILE对象

- fd
- buffer指针
- buffer尺寸
- buffer当前字节数
- 出错标志
- 文件结束标志

4. 流（stream）

- 无缓冲I/O围绕fd展开
- 有缓冲I/O围绕stream展开
- 流的定向（stream's orientation）
- 进程预定流
- 流缓冲

5. API

- fwide()设置流的定向
- setbuf、setvbuf：流缓冲相关的操作，更改缓冲类型

```
void setbuf(FILE *stream, char *buf);
setbuf函数打开或关闭缓冲机制，buf必须指向一个长度为BUFSIZ的缓冲区。调用此函数之后，
该流就是全缓冲的。单数如果该流是与一个终端设备相关，那么某些系统也可将其设置为行缓冲。
为了关闭缓冲，将buf设置为NULL
int setvbuf(FILE *stream, char *buf, int mode, size_t size);
使用setvbuf，我们可以精准的说明所需的缓冲类型，这是mode参数进行设置的
_IOFBF 全缓冲
_IOLBF 行缓冲
_IONBF 不带缓冲
```

- fflush函数:强制冲刷一个流

```
 int fflush(FILE *stream);
 如果参数是NULL，则冲刷所有打开的流
```

- 打开一个标准流

```
FILE *fopen(const char *pathname, const char *mode);
FILE *freopen(const char *pathname, const char *mode, FILE *stream);
 FILE *fdopen(int fd, const char *mode);
```

fopen函数打开路径名为pathname的一个指定的文件

freopen函数在一个指定的流上打开一个指定的文件，如果该流已经打开，则先关闭该流。若该流已经定向，则使用fopen清除该定向。该函数一般用于将一个指定的文件打开为一个预定义的流：标准输入、标准输出或标准错误

fdopen函数取一个已有的文件描述符，，并使一个标准IO流与该描述符相结合。此函数经常用于由创建管道和网络通信通道函数返回的描述符

```
mode
r" = "rt"
打开一个文本文件，文件必须存在，只允许读
"r+" = "rt+"
打开一个文本文件，文件必须存在，允许读写
"rb"
打开一个二进制文件，文件必须存在，只允许读
“rb+”
打开一个二进制文件，文件必须存在，允许读写
"w" = “wt”
新建一个文本文件，已存在的文件将内容清空，只允许写
"w+" = "wt+"
新建一个文本文件，已存在的文件将内容清空，允许读写
“wb”
新建一个二进制文件，已存在的文件将内容清空，只允许写
“wb+”
新建一个二进制文件，已存在的文件将内容清空，允许读写
"a" = "at"
打开或新建一个文本文件，只允许在文件末尾追写
"a+" = "at+"
打开或新建一个文本文件，可以读，但只允许在文件末尾追写
“ab”
打开或新建一个二进制文件，只允许在文件末尾追写
“ab+”
打开或新建一个二进制文件，可以读，但只允许在文件末尾追写
对于文件使用方式有以下几点说明：
1) 文件使用方式由r,w,a,t,b，+六个字符拼成，各字符的含义是：
r(read)：只读
w(write)：只写
a(append)：追加
t(text)：文本文件，可省略不写
```

```
b(binary)：二进制文件
```

```
+: 读和写
2）凡用“r”打开一个文件时，该文件必须已经存在，且只能从该文件读出。
3）用“w”打开的文件只能向该文件写入。若打开的文件不存在，则以指定的文件名建立该文件，若打开的文件
4）若要向一个已存在的文件追加新的信息，用“a”方式打开文件。如果指定文件不存在则尝试创建该文件。
5）在打开一个文件时，如果出错，fopen将返回一个空指针值NULL。在程序中可以用这一信息来判别是否完成
```

6. 关闭一个流

```
int close(int fd);
```

7. 读写

fread

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
ptr -- 这是指向带有最小尺寸 size*nmemb 字节的内存块的指针
size -- 这是要读取的每个元素的大小，以字节为单位。
nmemb -- 这是元素的个数，每个元素的大小为 size 字节。
stream -- 这是指向 FILE 对象的指针，该 FILE 对象指定了一个输入流。
```

fwrite

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)
ptr-- 这是指向要被写入的元素数组的指针。
size-- 这是要被写入的每个元素的大小，以字节为单位。
nmemb-- 这是元素的个数，每个元素的大小为 size 字节。
stream-- 这是指向 FILE 对象的指针，该 FILE 对象指定了一个输出流。
```

```
    #include <stdio.h>
    int fgetc(FILE *stream);
    char *fgets(char *s, int size, FILE *stream);每次读取一行IO
    int getc(FILE *stream);
    int getchar(void);
    int ungetc(int c, FILE *stream);
    size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

        int fputc(int c, FILE *stream);
       int fputs(const char *s, FILE *stream);//将一个以null字符终止的字符串写到标准输出
       int putc(int c, FILE *stream);
       int putchar(int c);
```

```
13          int puts(const char *s);
14  size_t fwrite(const void *ptr, size_t size, size_t nmemb,
15                        FILE *stream);-
```

8.
9.
10.
11.
12.
13.
14.
15.