# 230704——IO

# 标准IO

## 1.文件

```
文件：  数据的集合
分为:
    ASCII文件
    二进制文件
文件类型:
    - 普通文件
    d 目录文件(相当于windows下的文件夹)
    l 链接文件(相当于windows下的快捷方式)
    c 字符设备(串口/dev/ttyUSB0.. 摄像头 /dev/video0 ..)
    b 块设备 （磁盘文件 /dev/sda）
    p 管道文件 (用于本地进程间通信)
    s 套接字文件 （用于网络间通信）
```

## 2.库函数和系统调用

```
库函数:
  为了实现某一功能而封装起来的API接口，提供通用的接口；
  是语言或者应用进程的一部分;
printf(); fopen() fprintf() fread() fwrite()
fclos()...
  成熟的操作系统都具备c库;
系统调用:
  为了让普通用户访问内核数据封装的API接口;
  是操作系统的一部分;
open() read() write() close() ...;
  上层应用向操作系统发出请求，操作系统接收并处理请求，并将处理结
果再
   返回给应用层;
```

## 3.标准IO

```
标准IO是在文件IO的基础上封装的一些接口，作用：减少系统调用的次数，
减少系统开销;适用于普通文件；
缓冲区：
    缓冲区是应用层空间的一部分，为了让输入输出设备和cpu工作相协调；
   (为了让低速的输入输出设备和高速的cpu工作相协调，解放cpu)
  缓冲区又分为输入缓冲区和输出缓冲区；
当缓冲区满的时候执行写操作，(当缓冲区空的时候读操作）
标准IO的相关接口：
    fopen() fprintf() fread() fwrite() fclos()...
```

## 3.1 流

```
将数据的输入和输入抽象成向水流一样的数据的流进和流出；
  文本流；
  二进制流；
文件流指针：
    每个在被使用的文件它的相关信息都保存在一个名叫 FILE结构体的空间中，文件流指针就指向这个结构体；
  标准IO都是围绕文件流指针(FILE *)来进行的；

 sudo grep "_IO_FILE"  /usr/include -R

 typedef struct _IO_FILE FILE;
 struct _IO_FILE
{
  int _flags;           /* High-order word is _IO_MAGIC; rest is flags. */

  /* The following pointers correspond to the C++ streambuf protocol. */
  char *_IO_read_ptr;   /* Current read pointer */
  char *_IO_read_end;   /* End of get area. */
  char *_IO_read_base;  /* Start of putback+get area. */
  char *_IO_write_base; /* Start of put area. */
  char *_IO_write_ptr;  /* Current put pointer. */
  char *_IO_write_end;  /* End of put area. */
  char *_IO_buf_base;   /* Start of reserve area. */
  char *_IO_buf_end;    /* End of reserve area. */

  /* The following fields are used to support backing up and undo. */
  char *_IO_save_base; /* Pointer to start of non-current get area. */
  char *_IO_backup_base;  /* Pointer to first valid character of backup area */
  char *_IO_save_end; /* Pointer to end of non-current get area. */

```

```
  struct _IO_marker *_markers;

  struct _IO_FILE *_chain;

  int _fileno; //文件描述符
  int _flags2;
  __off_t _old_offset; /* This used to be _offset but it's too small.  */

  /* 1+column number of pbase(); 0 is unknown. */
  unsigned short _cur_column;
  signed char _vtable_offset;
  char _shortbuf[1];

  _IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};

```

## 3.2 缓冲区的类型

```
全缓冲：一般都默认为全缓冲;（缓冲区满执行写操作）
行缓冲:和终端相关的缓冲，遇'\n'刷新缓冲区
无缓冲:错误信息的输出;
```

## 3.3 刷新缓冲区的机制

```
默认指：输出缓冲区
  1.缓冲区满的时候会刷新;
  2.行缓冲遇'\n'刷新;
  3.当程序正常结束或者正常关闭文件流指针会自动刷新;
  4.遇到输入函数的时候也会刷新;
  5.fflush 强制刷新;
```

## 3.4 三个系统预定义的流

```
标准输入流
        stdin    0    STDIN_FILENO
标准输出流
        stdout   1    STDOUT_FILENO
标准错误流
        stderr   2    STDERR_FILENO
```

## 3.5 打开文件

```
FILE *fopen(const char *pathname, const char *mode)
{
  功能：打开一个文件（打开流）
  参数:
      pathname:文件名（包含路径）
```

```
        mode:打开的方式
             r:  只读 文件必须存在
             r+: 读写 文件必须存在
             w:  只写 文件不存在则创建，文件存在则清空
             w+: 读写 文件不存在则创建，文件存在则清空
             a:  只写 文件不存在则创建，从文件的末尾开始写;
             a+: 读写 文件不存在创建，从头读从末尾写;
                注意:
                    a+ 出现写操作就不会从头读，一开始从头读要写的时候
                    还是会去末尾写;
    //创建出来的文件权限默认为 0664;

 返回值：成功：返回一个文件流指针（返回保存文件相关信息结构体的地址）
          失败: 返回NULL并且设置错误号;
}
```

## 3.6 错误处理

```
错误号:（全局可见的变量）
  当函数出错的时候会去修改错误号;
#inlude <errno.h>
extern int errno

根据错误编号可以得到具体的错误信息;

① char *strerror(int errnum);
{
  参数:  错误号
  返回值:错误信息
}
② void perror(char *s)
{
  参数:
     s：一般就给要检查错误的这个函数的函数名
  功能: 打印错误信息，先打印s指向的字符串，再打印错误信息;
}
```

## 3.7 流的关闭

```
 int fclose(FILE *stream);
{
  参数:
    stream: 文件流指针;
  返回值:成功返回0，失败返回-1（EOF）并设置错误号;

  功能:
       关闭文件流指针，一个文件被关闭就不能再进行其他操作;
       一个文件流指针被正常关闭那么缓冲区的内容会自动刷新;
  打开流的个数是有限的，在打开文件使用完之后需要关闭流;
}
```

## 3.8 文件的读写

### 3.81 按字符读写

```
1)按字符来读 (输入)
  int fgetc(FILE *stream);
{
  功能:向一个打开的流读取（输入）一个字符;
  参数:
       stream: 输入流
  返回值:成功返回读取到的字符，失败或者读到文件的末尾返回EOF;
}
 2)按字符来写
  int fputc(int c, FILE *stream);
{
    功能:向一个打开的流写入(输出)一个字符;
    参数:
      c:要输出的字符;
     stream:输出流;
    返回值:成功返回写入的字符，失败返回-1;
}
eg:
   实现一个文件的复制;
```

### 3.82 按行读写

```
1） 按行读取
 char *fgets(char *s, int size, FILE *stream);
{
  功能:向打开的流读取一行数据;
  参数:
       s:指向一片空间用来保存读取到的内容;
       size: 要读取的字节数;
       stream: 输入流;
  返回值: 成功返回s,失败或者读到文件的末尾返回NULL;
  结束条件：遇到'\n' 或者读取到的字节数为size-1时结束;
}
当为标准输入的时候:
```

```
  char buf[5];
    输入的字符个数小于size-1会读取到'\n';
   eg:
      输入 he
      结果: 'h' 'e' '\n' '\0'
    输入的字符个数大于等于size-1个不会读取到'\n',
      eg:
        输入 hello
         结果: 'h' 'e' 'l' 'l' '\0'
    因为fgets从终端上读总会保留一个位置给'\0'结尾
处理'\n':
  eg:
      char str[32] =  "hello world\n";
      '\n'下标为 11
      strlen(str) == 12
      str[strlen(str) - 1] = '\0';

2）按行写入
  int fputs(const char *s, FILE *stream);
{
  功能:向一个打开的流输出内容;
  参数:
      s:待输出的内容的空间首地址;
      stream:输出流;
  返回值:正确的时候返回一个非负数，  错误的时候返回-1;
}

作业:
1.)统计一个文件的行数;
2).编程读写一个文件test.txt，每隔1秒向文件中写入一行数据，
  类似这样:
    1,  2007-7-30 15:16:42
    2,  2007-7-30 15:16:43
该程序应该无限循环，直到按Ctrl-C中断程序。
再次启动程序写文件时可以追加到原文件之后，并且序号能够接续上次的
序号，
      比如:
    1,  2007-7-30 15:16:42
    2,  2007-7-30 15:16:43
    3,  2007-7-30 15:19:02
    4,  2007-7-30 15:19:03
    5,  2007-7-30 15:19:04
      //如何获取时间?

```

## 3.83 按对象读写

```
```