文档版本	说明	作者	创建日期
V0.1	Linux系统编程：入门篇视频配套PPT	王利涛	2018年10月14日
V0.2	第01期：揭开文件系统的神秘面纱	王利涛	2018年11月07日
V0.3	第02期：文件IO编程实战	王利涛	2018年11月25日

Linux系统编程

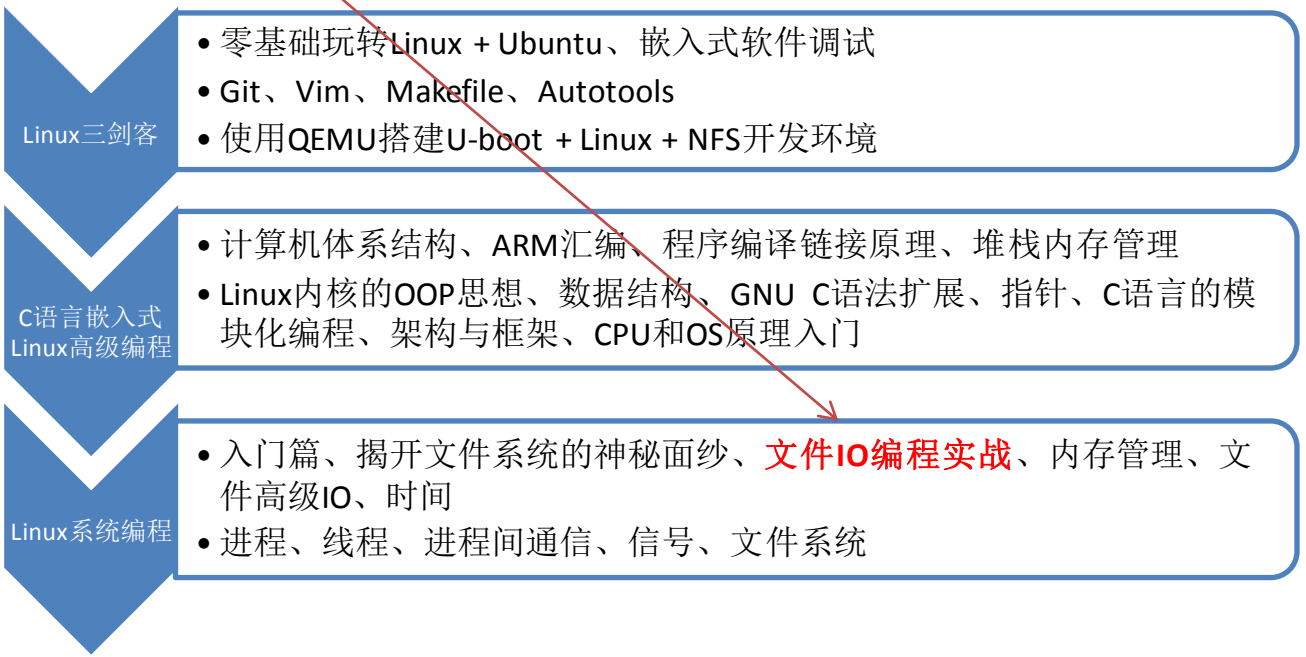
第02期：文件IO编程实战

《嵌入式工程师自修养》视频教程

- 第00步：Linux三剑客
- 第01步：C语言嵌入式Linux高级编程
- 第03步：Linux系统编程
- 第04步：Linux内核编程
- 第05步：嵌入式驱动开发
- 第06步：项目实战
- ----------
- 详情咨询QQ：3284757626
- 视频淘宝店：wanglitao.taobao.com
- 博客：www.zhaixue.cc
- 微信公众号：

学习路线图

- We are here…



系统编程

- 系统调用API
  - 应用开发：基于API的第三方库、框架、工具、系统软件
  - 底层开发：API的内核、底层驱动实现、读写流程

推荐书籍

- Unix环境高级编程(第三版)
- Linux系统编程
- Linux/Unix系统编程手册(上)、(下)

本期课程内容

- 文件IO常用API
  - 文件打开、关闭、读写、定位
  - 文件属性信息获取
  - 目录读取、遍历
- 音频播放器V2.0
- 实现自己的shell命令
  - cat、wc、pwd
  - ls -a -c -l

文件的打开模式

文件的基本操作

- int fd;
- fd = open (filename，flags，mode)；
- write (fd, buf, write_len);
- fsync (fd);　　刷新对应的缓冲区,sync刷新所有的缓冲区
- lseek (fd, 0, SEEK_SET);
- read (fd, buf, read_len);
- close (fd);

系统调用：open

- 函数处理流程
  - int open (const char *pathname, int flags);
  - int open (const char *pathname, int flags, mode_t mode);
  - 执行成功时：
    - 返回一个文件描述符
    - 将文件的读写位置偏移设置为0
    - 根据flags标志位给出的模式打开文件
  - 执行失败时
    - 返回-1
    - 设置errno

文件的打开模式

- 主参数
  - 只读模式：O_RDONLY
  - 只写模式：O_WRONLY
  - 读写模式：O_RDWR

文件的打开模式

- 副参数
  - O_CREAT：当文件不存在时创建该文件，文件存在时该位无效 与主参数一起使用
  - O_EXCL：与O_CREAT组合使用，当文件存在时，open调用失败，防止创建文件时出现竞争。
  - O_DIRECT：直接I/O
  - O_SYNC：同步I/O
  - O_ASYNC：用于终端或套接字、指定文件读写时产生一个信号
  - O_APPEND：追加模式
  - O_NOFOLLOW：若文件是一个软链接，则open调用失败
  - O_NOBLOCK：非阻塞模式打开
  - O_TRUNC：若文件存在，将文件长度截断为0,(FIFO、终端设备无效)
  - Create：O_WRONLY|O_CREAT|O_TRUNC组合封装

使用场景

- 当你…
  - 仅仅读一个文件：O_RDONLY
  - 仅仅写一个文件：O_WRONLY
  - 写一个文件，文件可能不存在：O_WRONLY | O_CREAT
  - 既读又写：O_RDWR

文件的关闭

- 系统调用：close
  - int close (int fd);
  - 成功：返回0
  - 失败：返回-1，并设置errno
- TIPS
  - 文件关闭并不意味该文件的数据已经被写到磁盘
  - 一个进程运行结束，会自动关系其打开的文件描述符
  - 显式关闭文件、尤其是对于对大量文件进行读写的后台程序

文件的创建

- 当文件不存在时
  - int open(const char *pathname, int flags);
  - int open(const char *pathname, int flags, mode_t mode); mode是权限
  - 创建文件要指定文件的权限，默认为未定义

创建文件可以指定mode,必须要有flags且有O_CREAT:0666 读写

文件的读写权限

文件的读写权限

- 创建文件：open
  - int open (const char *pathname, int flags);
  - int open (const char *pathname, int flags, mode_t mode);
  - 所有者 群组 其它人

    - R W X R W X R W X
    - 读： 4 R
    - 写： 2 W
    - 执行：1 X
    - 未定义： 0

rwx的组合最大值为7,用3个bit去表示正好,表示权限通常同8进制去表示权限
0666 : 所有者:6 群组:6 其他人:6 ==>能组合成6只有rw

修改读写权限

- 命令
  - $ chmod o+w hello.c
  - $ chmod a-x   hello.c
  - +：增加权限
  - -：删除权限
  - =：重新赋值，使之成为唯一的权限
  - u：所有者
  - g：所有者所在群主(group)
  - o：其他人(others)
  - a：所有人

修改读写权限

- 系统调用
  - int chmod (const char *pathname, mode_t mode);
  - int fchmod (int fd, mode_t mode);
  - pathname：可以是文件名、硬链接、软链接
  - mode：新设置的权限为，可以是8进制，也可以是掩码形式
    - 0777、0666　　最常用
    - S_IRUSR | S_IWUSR | S_IRGRP
    - S_IRUSR：用户读权限 (Permits the file's owner to read it)
    - S_IWUSR：用户写权限 (Permits the file's owner to write to it)
    - S_IXUSR：用户执行权限 (Permits the file's owner to execute it)
    - S_IRGRP：用户组读权限 (Permits the file's group to read it)
    - S_IWGRP：用户组写权限 (Permits the file's group to write to it)
    - S_IROTH：其他组读权限 (Permits others to read it)
    - S_IWOTH：其他组写权限 (Permits others to write to it)

问题

- 使用open打开并创建一个新文件，打开模式mode参数设置为0777
- 观察刚创建文件的读写权限
- 有什么异常？为什么？
- Umask ---： ----w—w-

第一次会用Umask屏蔽掉所有人和用户的写和执行
但是下一次设置就会生效!

编程作业

- 实现shell命令：chmod

文件的读写函数

文件读写

- 基本函数
  - ssize_t read (int fd, void *buf, size_t count);
    - 从fd指向的文件读取count个字节数据到缓冲区buf中
    - 读取成功：返回读取数据的长度、移动文件位置指针
    - 读取失败：返回-1，并设置errno值
  - ssize_t write (int fd, const void *buf, size_t count);
    - 将缓冲区buf中的数据写入fd指向的文件中
    - 写入成功：返回写入的字节数，并更新文件位置指针
    - 写入失败：返回-1，并设置errno值
    - 写入零个字节：返回0

read函数解析

- read的返回值
  - ssize_t read (int fd, void *buf, size_t count);
  - 返回值为count：成功读取count个字节
  - 返回值为(0,count)
    - 文件位置指针到了文件末尾
    - 读取的文件长度小于count
    - 系统调用被信号打断
    - 管道可能被破坏
  - 返回值为0：文件位置指针到了文件末尾（EOF）
  - 返回值为-1：读写错误，并设置errno值

编程实战

- 编写程序，实现shell命令cat的功能
  - 第1步：打开一个文本文件
  - 第2步：读取文本文件的内容，保存一个内存缓冲区内
  - 第3步：打印缓冲区里的数据到标准输出终端
  - 第4步：循环第2步，直到文件末尾
  - 第5步：关闭打开的文件
- 编程作业
  - 完善cat的功能
  - 支持显示多个文件
  - 支持行号显示

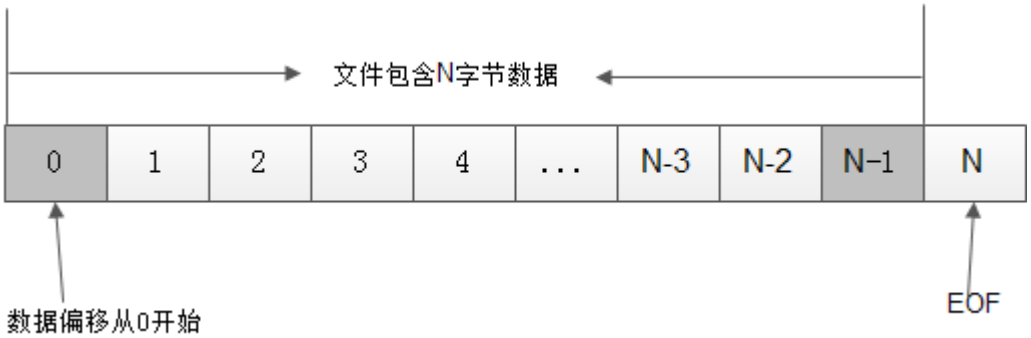
问题

- write函数调用成功，你的数据真得写到磁盘上了吗？
- void sync (void);
- int syncfs (int fd);

文件读写位置与定位

文件读写位置

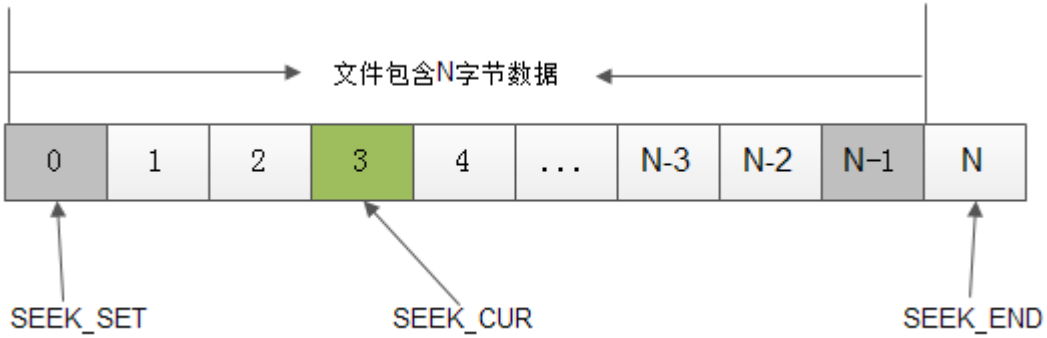
- 改变文件位置指针
  - 成功的read
  - 成功的write
  - 主动的lseek



改变文件偏移量

- lseek函数
  - off_t lseek(int fd, off_t offset, int whence);
  - 参数offset：可以为正、可以为负、可以为0
  - 参数whence
    - SEEK_SET：文件开头
    - SEEK_CUR：文件当前偏移位置
    - SEEK_END：文件尾部

位置:whence+offset



文件空洞

- 文件结束标志EOF
  - 文件存储数据最后一个字节的下一个位置
  - 当文件指针到了EOF，继续读写，会发生怎么情况？
    - read：返回 0
    - write：可以在任意位置写
  - 文件空洞：从文件结尾到新写入数据之间的空间

TIPS

- 字符设备、Socket、终端设备等流式设备不支持lseek
- 管道、FIFO不支持lseek

获取文件的属性信息

文件属性

- 文件数据的存储
  - 纯数据：存储在data block中
  - 元数据：存储在inode table中
  - 文件名：存储在目录文件的目录项中
- 元数据
  - 文件时间戳
  - 文件权限
  - 文件所有权
  - 文件存储地址
  - 链接数

获取文件的元数据

- 系统调用接口
  - int stat (const char *pathname, struct stat *buf);
  - int fstat (int fd, struct stat *buf);
  - int lstat (const char *pathname, struct stat *buf);
  - TIPS：
    - lstat获取软链接符号文件本身的属性信息
    - stat获取命名文件的属性信息

获取文件元数据

- 结构体：stat

```
struct  stat{
        dev_t      st_dev;          /* ID of device containing file */
        ino_t      st_ino;          /* inode number */
        mode_t     st_mode;         /* protection */
        nlink_t    st_nlink;        /* number of hard links */
        uid_t      st_uid;          /* user ID of owner */
        gid_t      st_gid;          /* group ID of owner */
        dev_t      st_rdev;         /* device ID (if special file) */
        off_t      st_size;         /* total size, in bytes */
        blksize_t  st_blksize;      /* blocksize for filesystem I/O */
        blkcnt_t   st_blocks;       /* number of 512B blocks allocated */
        struct timespec   st_atim;  /* time of last access */
        struct timespec   st_mtim;  /* time of last modification */
        struct timespec   st_ctim;  /* time of last status change */
};
```

实现shell命令：ll

getpwuid函数

- 函数原型
  - struct passwd *getpwuid(uid_t uid);
  - 功能：根据用户的UID查找该用户的/etc/passwd数据
  - 运行成功：返回一个跟uid相关的password结构体指针
  - 运行失败：返回NULL指针，并设置errno的值

```
struct passwd
{
    char        *pw_name;       /* Username.  */
    char        *pw_passwd;     /* Password.  */
    __uid_t     pw_uid;         /* User ID.  */
    __gid_t     pw_gid;         /* Group ID.  */
    char        *pw_gecos;      /* Real name.  */
    char        *pw_dir;        /* Home directory.  */
    char        *pw_shell;      /* Shell program.  */
};
```

getgrgid函数

- 函数原型
  - struct group *getgrgid(gid_t gid);
  - 功能：根据gid参数去搜索/etc/group文件
  - 成功运行：返回指定组ID：gid的group结构体指针
  - 失败运行：返回NULL指针，并设置errno

```
struct group
 {
        char *gr_name;          /* Group name.      */
        char *gr_passwd;        /* Password.        */
        __gid_t gr_gid;         /* Group ID.        */
        char **gr_mem;          /* Member list.     */
 };
```

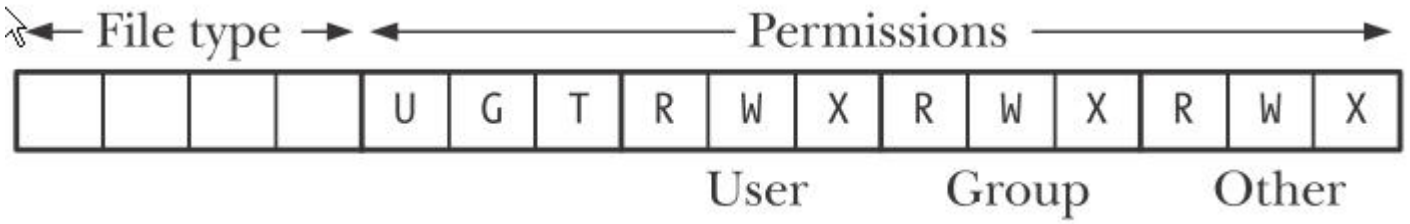
文件元数据

- stat结构体

```
struct  stat {
        dev_t     st_dev;       /* ID of device containing file */
        ino_t     st_ino;       /* inode number */
        mode_t    st_mode;      /* protection */
        nlink_t   st_nlink;     /* number of hard links */
        uid_t     st_uid;       /* user ID of owner */
        gid_t     st_gid;       /* group ID of owner */
        dev_t     st_rdev;      /* device ID (if special file) */
        off_t     st_size;      /* total size, in bytes */
        blksize_t st_blksize;   /* blocksize for filesystem I/O */
        blkcnt_t  st_blocks;    /* number of 512B blocks allocated */
        struct timespec  st_atim;  /* time of last access */
        struct timespec  st_mtim;  /* time of last modification */
        struct timespec  st_ctim;  /* time of last status change */
};
```

文件类型+权限

- mod_t数据信息
  - 一个32位的整型变量
  - 文件类型：普通文件、目录、字符设备、块设备、管道、软链接



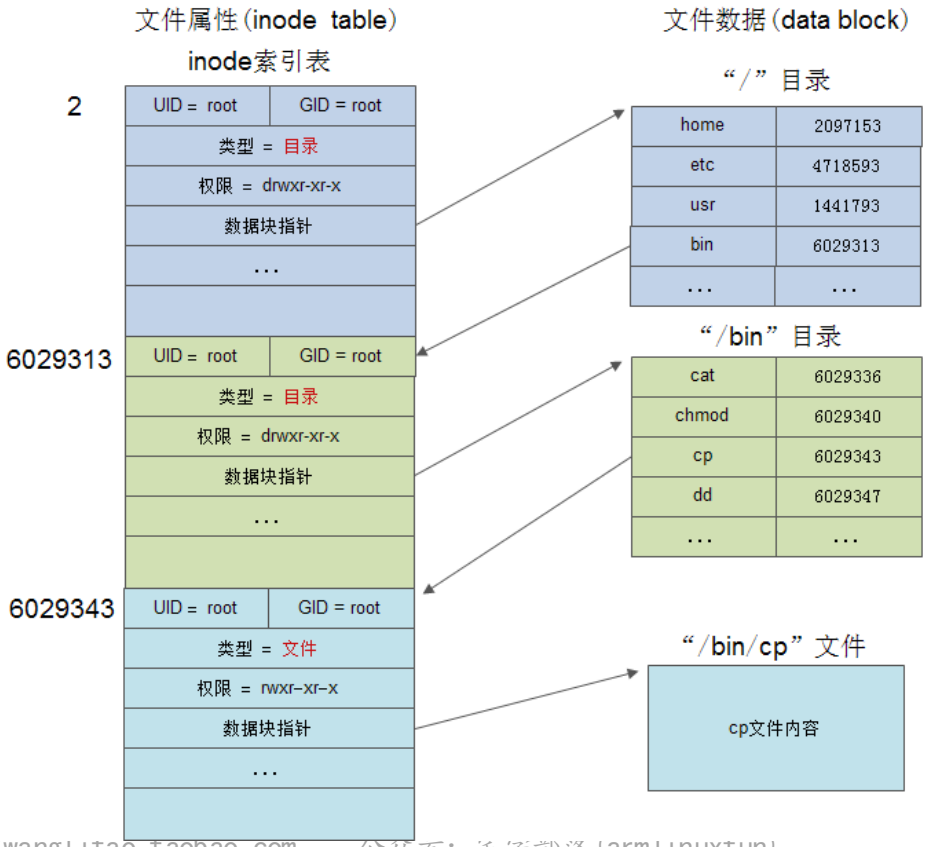
测试宏

```
#define S_IFMT  00170000
#define S_IFSOCK 0140000
#define S_IFLNK  0120000
#define S_IFREG  0100000
#define S_IFBLK  0060000
#define S_IFDIR  0040000
#define S_IFCHR  0020000
#define S_IFIFO  0010000
#define S_ISUID  0004000
#define S_ISGID  0002000
#define S_ISVTX  0001000

#define S_ISLNK(m)      (((m) & S_IFMT) == S_IFLNK)
#define S_ISREG(m)      (((m) & S_IFMT) == S_IFREG)
#define S_ISDIR(m)      (((m) & S_IFMT) == S_IFDIR)
#define S_ISCHR(m)      (((m) & S_IFMT) == S_IFCHR)
#define S_ISBLK(m)      (((m) & S_IFMT) == S_IFBLK)
#define S_ISFIFO(m)     (((m) & S_IFMT) == S_IFIFO)
#define S_ISSOCK(m)     (((m) & S_IFMT) == S_IFSOCK)

#define S_IRWXU 00700
#define S_IRUSR 00400
#define S_IWUSR 00200
#define S_IXUSR 00100

#define S_IRWXG 00070
#define S_IRGRP 00040
#define S_IWGRP 00020
#define S_IXGRP 00010

#define S_IRWXO 00007
#define S_IROTH 00004
#define S_IWOTH 00002
#define S_IXOTH 00001
```

读取目录文件

目录是一个文件

- 文件存储
  - 文件数据：存放在data block中
  - 文件属性：存放在inode中
  - 文件名：若干目录项组成的表格，存放在目录块中(data block)



读取目录内容

- 打开目录文件
  - DIR *opendir (const char *name);
  - DIR *fdopendir (int fd);

```
typedef  struct __dirstream
  {
        void *      __fd;
        char *      __data;
         int        __entry_data;
        char *      __ptr;
        int         __entry_ptr;
        size_t      __allocation;
        size_t      __size;
         __libc_lock_define (, __lock)
  } DIR;
```

读取目录内容

- 目录项
  - struct dirent *readdir(DIR *dirp);

    对一个已经打开目录进行操作,挨个读取目录中文件对象并返回指针,读取指向下一个

```
struct dirent
{
        long            d_ino;     //inode编号
        off_t           d_off;     //该目录项在目录文件中的偏移
        unsigned short d_reclen; //文件名长度
        unsigned char  d_type;    //文件类型：普通文件、目录…
        char d_name [NAME_MAX+1]; //文件名
}
```

编程

- 实现ls 命令，显示当前目录下的所有文件(普通文件、目录)
- 第1步：打开一个目录文件
- 第2步：读取目录文件，返回一个目录项
- 第3步：显示这个目录项
- 第4步：循环第2、3步，直到读取文件末尾
- 第5步：关闭这个目录文件

实现ls命令：支持多个目录

支持多个目录

- 实现思路
  - 当ls命令无参数时，默认显示当前目录的内容
  - 当有参数时，显示指定目录的内容
  - 支持多个目录参数

实现ls命令：支持-c参数

支持-c参数

- 实现思路
  - 第1步：在main函数中解析命令行参数，看一下是否有-c这个参数
  - 第2步：在do_ls函数中，增加对-c参数的支持

编程作业

- 实现ls命令，支持-a参数

音频播放器：实现循环播放列表

实现ls命令：支持-l参数

编程作业

- 实现ls命令，增加-a参数

目录的其它相关操作

目录操作

- 基本操作
  - 获取当前工作目录：char *getcwd(char *buf, size_t size);
  - 改变当前工作目录： int chdir(const char *path);
  - 创建目录： int mkdir(const char *pathname, mode_t mode);
  - 删除目录： int rmdir(const char *pathname);

路径操作

- 路径解析
  - 获取绝对路径： char *realpath(const char *path, char *resolved_path);
  - 获取路径名： char *dirname(char *path);
  - 获取文件名： char *basename(char *path);

链接

- 基本操作
  - 硬链接： int link(const char *oldpath, const char *newpath);
  - 软链接： int symlink(const char *target, const char *linkpath);
  - 删除链接： int unlink(const char *pathname);

编程作业

- 实现shell命令：mkdir
- 实现shell命令：ln -s

相对路径转绝对路径

实现方法

- 系统调用函数
  - char *getcwd(char *buf, size_t size);
  - char *getwd(char *buf);
  - char *get_current_dir_name(void);
  - 功能：将当前目录的绝对路径拷贝到buf中
  - 成功：返回当前的工作目录
  - 失败：若buf的大小size无法保存路径，则返回NULL，并设置error

编程

- 实现shell命令：pwd
- 将一个文件名转换为绝对路径+ 文件名的形式

编程实战：实现 wc 命令

编程实战

- 编写一个工具wc，统计Linux最新内核源码：
- 一共有多少个C文件？
- 一共有多少个H文件？
- 一共有多少个汇编文件？
- 一共有多少个words、行？
- 整个项目一共有多少行代码？

功能分析

- 实现思路
  - 先统计单个C文件的字数、代码行数
  - 然后遍历一个目录下的所有C文件
  - 如果该目录下面还有子目录、递归遍历
  - 统计、累加
  - 打印输出

功能实现

- 文件遍历
  - 各个目录、子目录递归遍历
  - 当一个目录下面有文件、子目录时的处理
  - 搜索指定格式的文件：C文件、H文件、汇编文件

编程作业

- 实现shell命令：wc，增加–c -h –S 参数