文件系统

文件存储

首先了解如下文件存储相关概念：inode、 dentry、 数据存储、文件系统。

inode

其本质为结构体，存储文件的属性信息。如：权限、类型、大小、时间、用户、盘块位置……也叫作文件属性管理结构，大多数的 inode 都存储在磁盘上。

少量常用、近期使用的 inode 会被缓存到内存中。

dentry

目录项，其本质依然是结构体，重要成员变量有两个 {文件名，inode，...}，而文件内容(data)保存在磁盘盘块中。

文件系统

文件系统是，一组规则，规定对文件的存储及读取的一般方法。文件系统在磁盘格式化过程中指定。

常见的文件系统有：fat32 ntfs exfat ext2 、ext3 、ext4

文件操作

stat 函数

获取文件属性，(从 inode 结构体中获取)

int stat(const char *path, struct stat *buf); 成返回 0；失败返回-1 设置 errno 为恰当值。

参数 1：文件名

参数 2：inode 结构体指针 (传出参数)

文件属性将通过传出参数返回给调用者。

练习：使用 stat 函数查看文件属性

【stat.c】

lstat 函数

int lstat(const char *path, struct stat *buf); 成返回 0；失败返回-1 设置 errno 为恰当值。

练习：给定文件名，判断文件类型。
【get_file_type.c】

文件类型判断方法：st_mode 取高 4 位。 但应使用宏函数：

S_ISREG(m) is it a regular file?
S_ISDIR(m) directory?
S_ISCHR(m) character device?
S_ISBLK(m) block device?
S_ISFIFO(m) FIFO (named pipe)?
S_ISLNK(m) symbolic link? (Not in POSIX.1-1996.)
S_ISSOCK(m) socket? (Not in POSIX.1-1996.)

穿透符号链接：stat：会；lstat：不会

truncate 函数

截断文件长度成指定长度。常用来拓展文件大小，代替 lseek。

int truncate(const char *path, off_t length); 成功：0；失败：-1 设置 errno 为相应值
int ftruncate(int fd, off_t length);

link 函数

思考，为什么目录项要游离于 inode 之外，画蛇添足般的将文件名单独存储呢？？这样的存储方式有什么样的好处呢？

其目的是为了实现文件共享。Linux 允许多个目录项共享一个 inode，即共享盘块(data)。不同文件名，在人类眼中将它理解成两个文件，但是在内核眼里是同一个文件。

link 函数，可以为已经存在的文件创建目录项(硬链接)。

int link(const char *oldpath, const char *newpath); 成功：0；失败：-1 设置 errno 为相应值

注意：由于两个参数可以使用“相对/绝对路径+文件名”的方式来指定，所以易出错。

如：link("../abc/a.c", "../ioc/b.c")若 a.c，b.c 都对， 但 abc，ioc 目录不存在也会失败。

mv 命令既是修改了目录项，而并不修改文件本身。

unlink 函数

删除一个文件的目录项；

int unlink(const char *pathname); 成功：0；失败：-1 设置 errno 为相应值

练习：编程实现 mv 命令的改名操作

【imp_mv.c】

注意 Linux 下删除文件的机制：不断将 st_nlink -1，直至减到 0 为止。无目录项对应的文件，将会被操作系统择机释放。(具体时间由系统内部调度算法决定)

因此，我们删除文件，从某种意义上说，**只是让文件具备了被释放的条件。**

unlink 函数的特征：清除文件时，如果文件的硬链接数到 0 了，没有 dentry 对应，但该文件仍不会马上被释放。要等到所有打开该文件的进程关闭该文件，系统才会挑时间将该文件释放掉。 【unlink_exe.c】

隐式回收

当进程结束运行时，所有该进程打开的文件会被关闭，申请的内存空间会被释放。系统的这一特性称之为隐式回收系统资源。

目录操作

工作目录：“./”代表当前目录，指的是进程当前的工作目录，默认是进程所执行的程序所在的目录位置。

getcwd 函数

获取进程当前工作目录 (卷 3，标库函数)

char *getcwd(char *buf, size_t size); 成功：buf 中保存当前进程工作目录位置。失败返回 NULL。

chdir 函数

改变当前进程的工作目录

int chdir(const char *path); 成功：0；失败：-1 设置 errno 为相应值

练习：获取及修改当前进程的工作目录，并打印至屏幕。

【imp_cd.c】

文件、目录权限

注意：目录文件也是“文件”。其文件内容是该目录下所有子文件的目录项 dentry。 可以尝试用 vim 打开一个目录。

	r	w	x
文件	文件的内容可以被查看 cat、more、less…	内容可以被修改 vi、> …	可以运行产生一个进程 ./文件名
目录	目录可以被浏览 ls、tree…	创建、删除、修改文件 mv、touch、mkdir…	可以被打开、进入 cd

目录设置黏住位：若有 w 权限，创建不变，删除、修改只能由 root、目录所有者、文件所有者操作。

opendir 函数

根据传入的目录名打开一个目录 (库函数)　　DIR * 类似于 FILE *

DIR *opendir(const char *name);　成功返回指向该目录结构体指针，失败返回 NULL

参数支持相对路径、绝对路径两种方式：例如：打开当前目录：① getcwd() , opendir() ② opendir(".");

closedir 函数

关闭打开的目录

int closedir(DIR *dirp);　　成功：0；失败：-1 设置 errno 为相应值

readdir 函数

读取目录　　(库函数)

struct dirent *readdir(DIR *dirp);　成功返回目录项结构体指针；失败返回 NULL 设置 errno 为相应值

需注意返回值，读取数据结束时也返回 NULL 值，所以应借助 errno 进一步加以区分。

struct 结构体：

```
struct dirent {
    ino_t          d_ino;      inode 编号
    off_t          d_off;
    unsigned short d_reclen;   文件名有效长度
```

```
        unsigned char    d_type;        类型(vim 打开看到的类似@*/等)
        char             d_name[256];文件名
    };
```

其成员变量重点记忆两个：d_ino、d_name。实际应用中只使用到 d_name。

练习 1：实现简单的 ls 功能。

【imp_ls.c】

练习 2：实现 ls 不打印隐藏文件。每 5 个文件换一个行显示。

【imp_ls2.c】

拓展 1：实现 ls -a -l 功能。

拓展 2：统计目录及其子目录中的普通文件的个数

递归遍历目录

查询指定目录，递归列出目录中文件，同时显示文件大小。

【ls_R.c】

重定向

dup 函数

功能:文件描述符拷贝。

使用现有的文件描述符，拷贝生成一个新的文件描述符，且函数调用前后这个两个文件描述符指向同一文件。

int dup(int oldfd); 成功：返回一个新文件描述符；失败：-1 设置 errno 为相应值

dup2 函数

功能:文件描述符拷贝。重定向文件描述符指向。

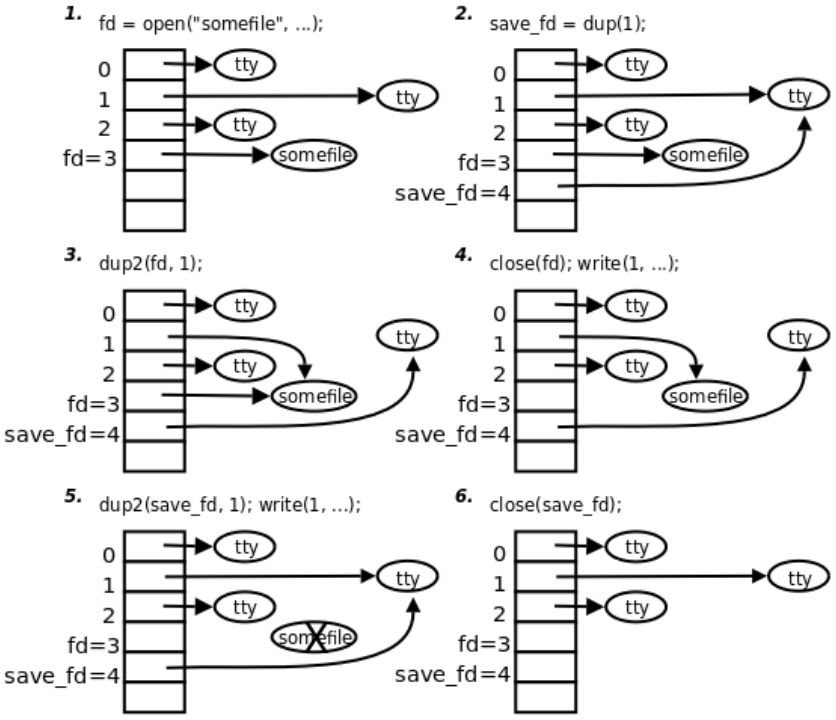
通过该函数可实现命令行“重定向”功能。使得原来指向某文件的文件描述符，指向其他指定文件。

int dup2(int oldfd, int newfd);

成功：返回一个新文件描述符；

如果 oldfd 有效，则返回的文件描述符与 oldfd 指向同一文件。

失败：如果 oldfd 无效，调用失败，关闭 newfd。返回-1，同时设置 errno 为相应值。



重定向示

记忆方法两种：

1. 文件描述符的本质角度理解记忆。
2. 从函数原型及使用角度，反向记忆。

练习：借助 dup 函数编写 mycat 程序，实现 cat file1 > file2 命令相似功能。

【mycat.c】

fcntl 函数

当 fcntl 的第二个参数为 F_DUPFD 时， 它的作用是根据一个已有的文件描述符，复制生成一个新的文件描述符。此时，fcntl 相当于 dup 和 dup2 函数。

参 3 指定为 0 时，因为 0 号文件描述符已经被占用。所以函数自动用一个最小可用文件描述符。

参 3 指定为 9 时，如果该文件描述符未被占用，则返回 9。否则，返回大于 9 的可用文件描述符。

【fcntl_dup.c】