3-文件IO

3.1 文件描述符

- 标准输入、标准输出、标准错误（STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO）定义在<unistd.h>
- 文件描述符的变化范围在0~OPEN_MAX - 1

3.2 open和openat函数

```
#include <fcntl.h>
int open(const char *path, int oflag,... /* mode_t mode */);
int openat(int fd, const char *path, int oflag,... /* mode_t mode */);
// 成功返回文件描述符，失败返回-1
```

- oflag在头文件<fcntl.h>中定义
- openat希望解决问题
  - 线程可以使用相对路径名打开文件
  - 避免time-of-check-to-time-of-use错误：如果有两个基于文件的函数调用，由于调用不是原子操作，可能出现文件被改变的情况

3.2.1 oflag标记

- O_NONBLOCK or O_NDELAY：
  - 如果path引用一个FIFO、块特殊设备或一个字符特殊设备；为文件的本次打开操作和后续的IO操作设置非阻塞方式
  - 非阻塞：进程不能进行设备操作时并不挂起；或者**放弃，或者不停的查询**，直到可以进行操作为止.
- O_CLOEXEC：
  - 把FD_CLOEXEC设置为文件标志；exec族执行其他程序的时候，全新的程序会替换子进程中的地址空间，数据段，堆栈，此时保存与父进程文件描述符也就不存在了，也无法进行关闭，这时候就需要FD_CLOEXEC, **表示子进程在执行exec的时候，该文件描述符就需要进行关闭**
- O_EXEC:
  - 只能打开文件
- O_PATH：
  - 对普通文件，有打开权限和执行权限；但是没有读权限

```
1   char buf[PATH_MAX];
2   fd = open("some_prog", O_PATH);
3   snprintf(buf, PATH_MAX, "/proc/self/fd/%d", fd);
4   execl(buf, "some_prog", (char *) NULL);
```

- O_EXCL：
  - 测试一个文件是否已经存在，如果不存在，创建此文件
- O_NOFOLLOW：
  - 如果path引用一个符号链接，出错
- O_SYNC:
  - 每次write等待物理IO操作完成
- O_DSYNC：
  - 每次write等待物理IO操作完成；但是如果写操作不影响读取刚写入的数据，**不需要等待文件属性被更新**
- O_RSYNC:
  - 使同一fd上的read操作阻塞，直到对该fd的所有写操作都完成
- O_TRUNC：
  - 文件存在且被写或读写打开，长度截断为0

3.2 creat函数

- creat以只写权限打开文件，如果要读必须再次调用read调用

```
#include <fcntl.h>
int creat(const char *path, mode_t mode);
open(path, O_WRONLY | O_CREAT | O_TRUNC, mode);
```

3.3 close函数

- 关闭文件还会释放进程加在文件上的所有记录锁

```
#include <fcntl.h>
int close(int fd);
```

3.4 lseek函数

```
#include <fcntl.h>
off_t lseek(int fd, off_t offset, int whence);
```

- 偏移量一般为正值；Intel x86处理器下的FreeBSD设备/dev/kmem支持负 的偏移量
- off_t类型用于指示文件的偏移量，常就是long类型，其默认为一个32位的整数，在gcc编译中会被编译为long int类型，在64位的Linux系统中则会被编译为long long int，这是一个64位的整数，其定义在unistd.h头文件中可以查看。
- 尽管off_t可以是64位，但是是否可以创建一个超过2GB的文件，需要依赖于底层文件系统的实现

3.5 read函数

- 返回值ssize_t是带符号类型（返回-1，0，正整数）；size_t是无符号类型

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t n);
```

- 从终端设备中读只能一行一行读取
- 可能读取的实际字节数小于要求读的字节数

- 如果被信号终端；如果errno是EINTR，如果系统调用不能自动重启，我们需要捕获该错误重新进行read

3.6 write函数

- write出错可能是磁盘已经写满或者超过一个进程的文件长度限制

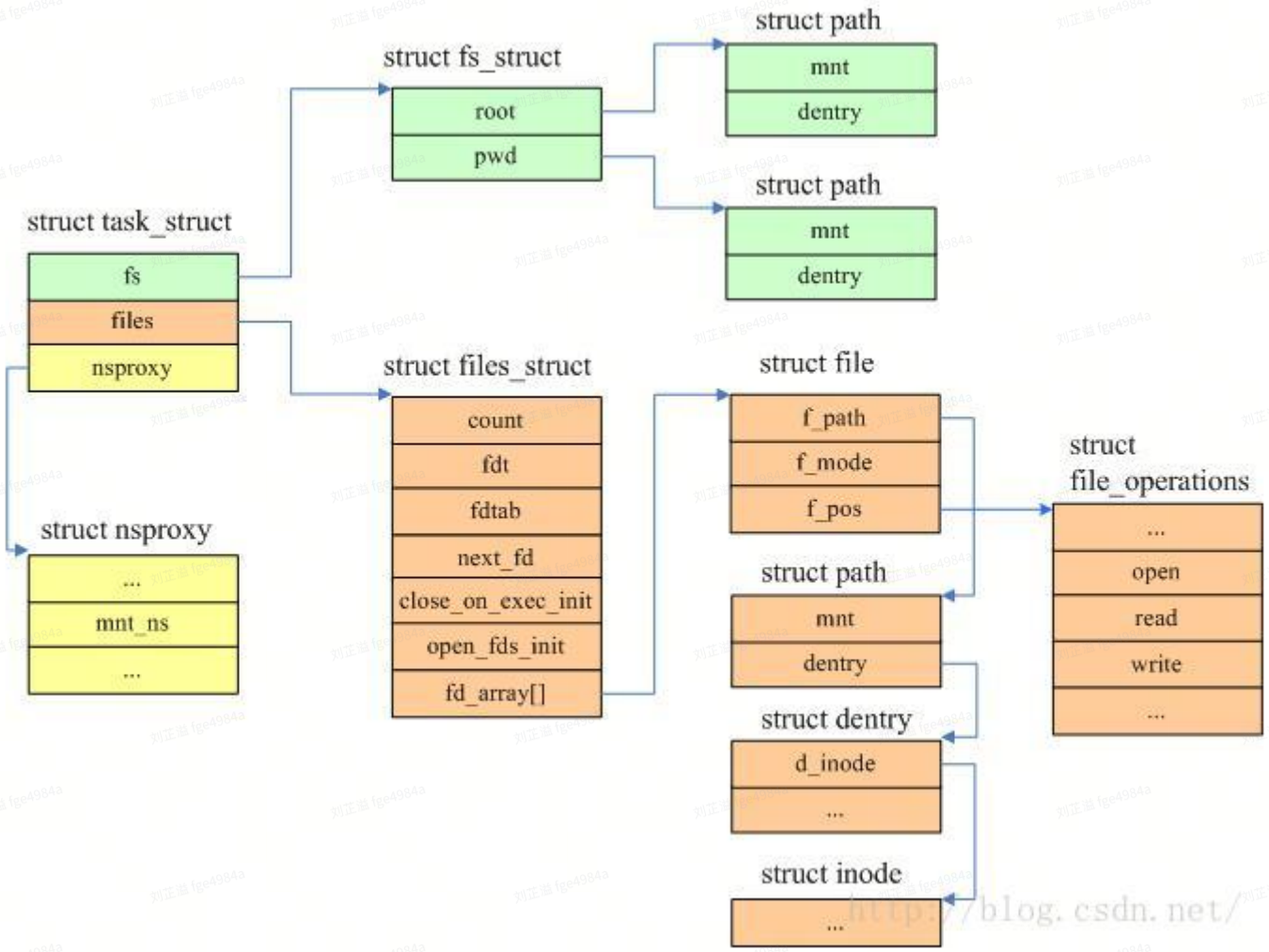
```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t nbytes);
```

3.7 IO效率

- 大多数文件系统为改善性能通常采用预读技术：读出比需要更多的数据
- 使用缓冲区进行读写；如果从磁盘上读或者写入磁盘，采用顺序读写

3.8 文件共享

- Linux VFS管理文件结构
- dentry是内存中缓存目录项的结构
- 文件描述符标志只作用于一个进程的一个描述符
- 文件状态标志指向给定文件表项的任何进程中的所有描述符



3.9 原子操作

- O_APPEND是一种原子操作方法，每次将文件偏移定到文件尾端后进行写入

```
#include <unistd.h>
int pread(int fd, void *buf, size_t nbytes, off_t offset);
int pwrite(int fd, void *buf, size_t nbytes, off_t offset);
```

- pread相当于先lseek后调用read；但pread不更新当前偏移量；pwrite也类似
- dup和dup2

```
#include <unistd.h>
int dup(int fd); // fcntl(fd, F_DUPFD, 0);
int dup2(int fd, int fd2);
// close(fd);
// fcntl(fd, F_DUPFD, fd2);
// 但两个语句不是原子操作，dup2是原子操作
```

- 如果fd2已经打开，先将其关闭；如若fd == fd2，dup2返回fd2，但是不关闭；
- 返回的fd2的FD_CLOEXECB文件描述符标志被清除

3.10 函数sync，fsync和fdatasync

- 向文件写入数据，内核通常先将数据复制到缓冲区，排入队列，晚些时候写入磁盘

```
#include <unistd.h>
int fsync(int fd);
int fdatasync(int fd);
void sync(void);
```

- sync将**所有修改过的块缓存区排入写队列，马上返回**
- fsync函数**等待写磁盘操作结束才返回**
- fdatasync函数只影响文件的数据部分；fsync会同步更新文件的属性

3.11 函数fcntl

```
#include <fcntl.h>
int fcntl(int fd, int cmd,... /*int arg*/);
```

- 可以对管道使用fcntl，对已经打开的文件设置文件标志
- F_DUPFD：复制描述符fd，返回值是一个新的描述符
- F_DUPFD_CLOEXEC：复制文件描述符，设置与新描述符关联的FD_CLOEXEC文件描述符标志的值
- F_GETFD，F_SETFD：修改文件描述符标志
- F_GETFL，F_SETFL：修改文件标志

3.12 /dev/fd

- 该目录是进程打开的文件描述符数组