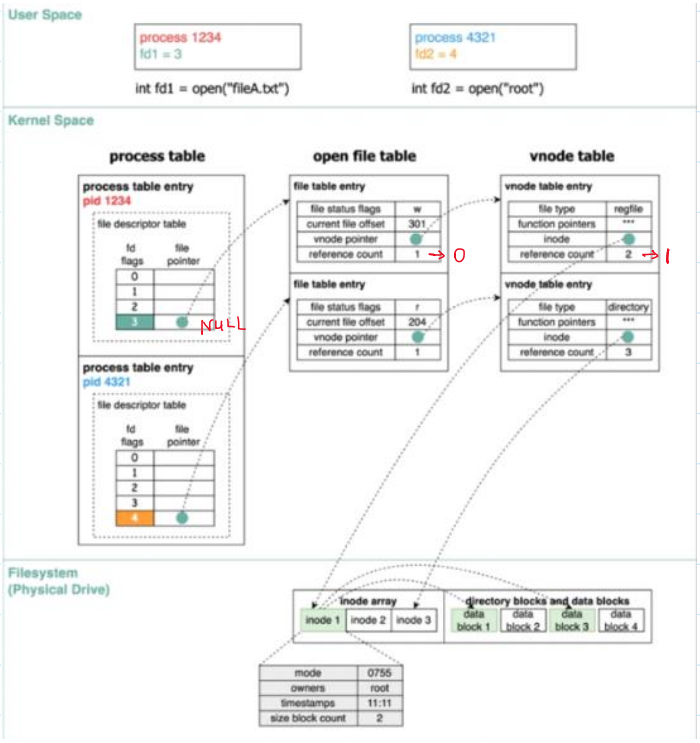
close

2024年5月20日 9:42

```
CLOSE(2)            Linux Programmer's Manu

NAME
       close - close a file descriptor

SYNOPSIS
       #include <unistd.h>

       int close(int fd);
```

RETURN VALUE

close() returns zero on success. On error, -1 is returned, and errno is set appropriately.



fd = 3.

close (fd);

可能会被改数据结构句

read

2024年5月20日 9:47

```
READ(2)                    Linux Programmer's Manual

NAME
       read - read from a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t read(int fd, void *buf, size_t count);
```

数组的起始地址

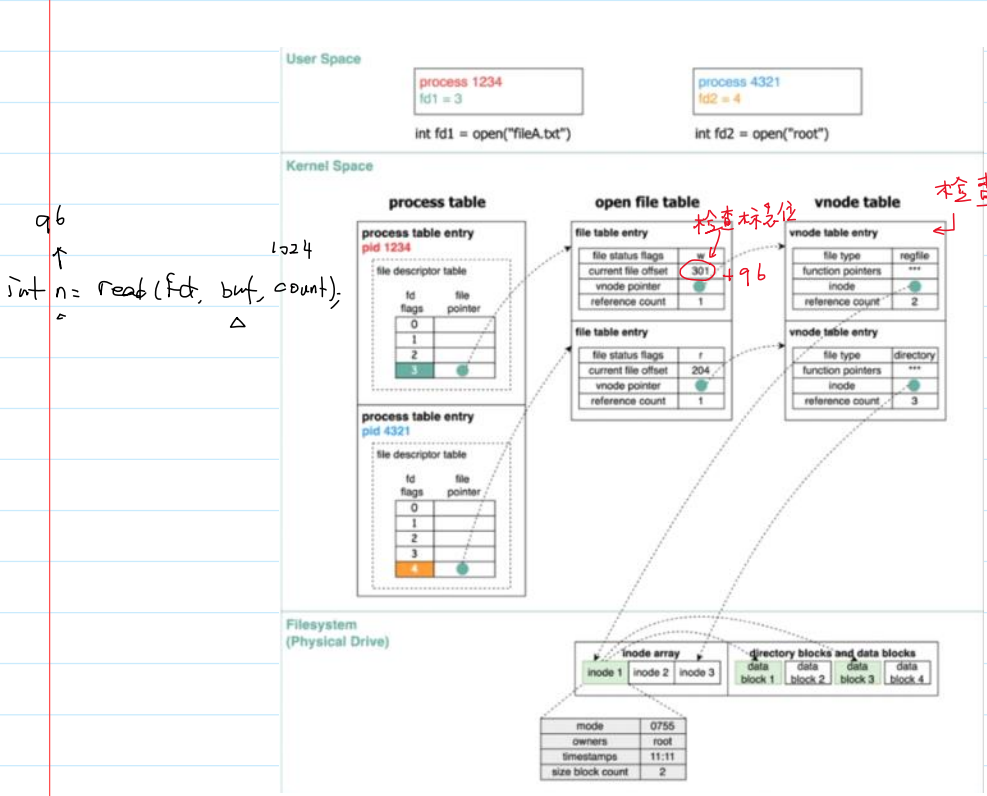
文件描述符

最多读的字节数

```
RETURN VALUE
       On success, the number of bytes read is returned (zero indicates end of
       file), and the file position is advanced by this number.  It is not an er-

       On error, -1 is returned, and errno is set appropriately.
```

成功：实际读取的字节数目（0，表示读到了文件的末尾）

失败：-1，并且设置 errno。



96

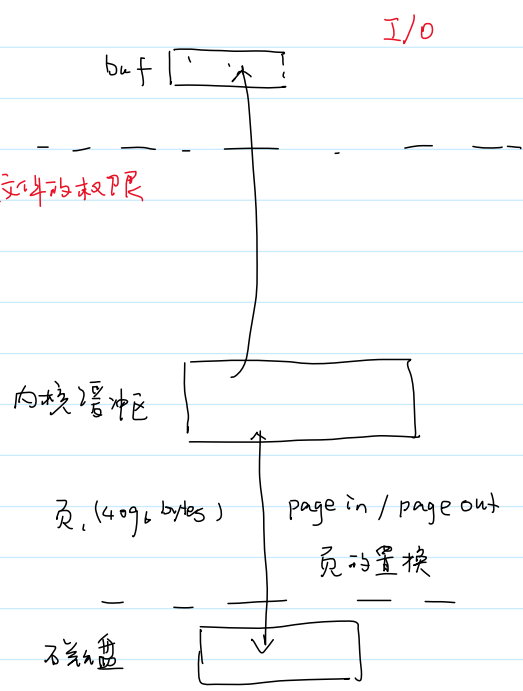
int n = read(fd, buf, count);

1024

检查标志位

496

检查文件的权限



write

2024年5月20日 9:58

WRITE(2) Linux Programmer's Manual

NAME

write - write to a file descriptor

SYNOPSIS

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

96 1024

int n = write(fd, buf, count)

要写对字节数目！

RETURN VALUE

On success, the number of bytes written is returned. On error, -1 is returned, and errno is set to indicate the cause of the error.

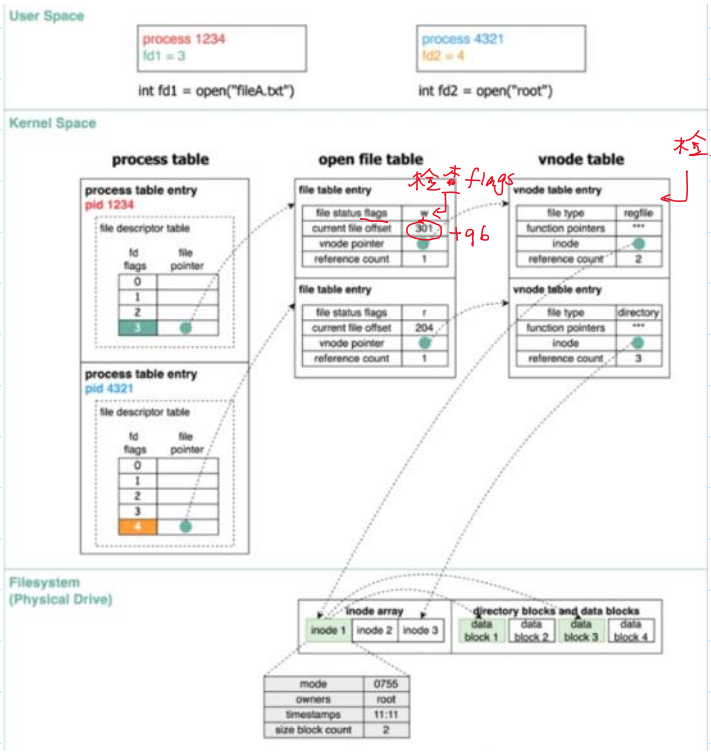
成功：实际写入字节数目. (n ≤ count)

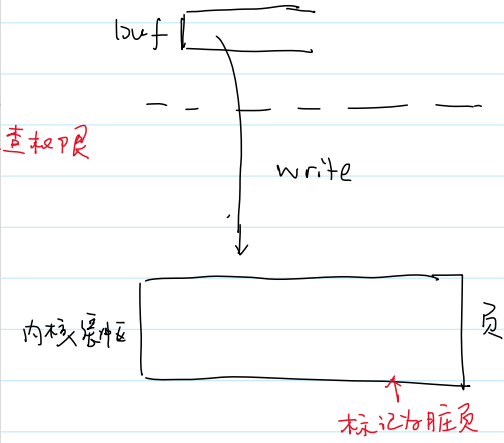
失败：-1，并且设置errno.

网络 TCP 流量控制

96 1024

int n = write(fd, buf, count)





O_APPEND

lseek

2024年5月20日 10:11

```
LSEEK(2)                    Linux Programmer's Manual

NAME
       lseek - reposition read/write file offset

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

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

long

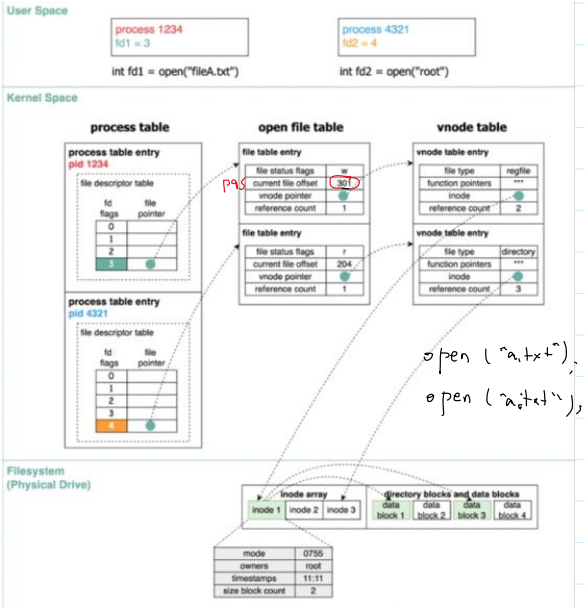
不会读写磁盘

pos = offset　SEEK_SET
The file offset is set to offset bytes.

pos += offset　SEEK_CUR
The file offset is set to its current location plus offset bytes.

pos = size + offset　SEEK_END
The file offset is set to the size of the file plus offset bytes.

文件内容大小



inode：不变量，文件内容数据的信息

一一对应

vnode，inode的缓存。

（设备号）

n对1

open file

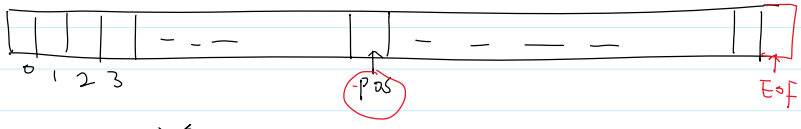
open ("a.txt");

open ("a.txt");

RETURN VALUE

Upon successful completion, **lseek**() returns the resulting offset location as measured in bytes from the beginning of the file. On error, the value (off_t) -1 is returned and errno is set to indicate the error.

成功：文件的位置（距离文件开头的字节数目）



失败：-1，设置 errno。

库函数　　系统调用

fseek (stream, offset, whence) ⟶ lseek (fd, offset, whence)

ftell (stream) ⟶ lseek (fd, 0, SEEK_CUR)

rewind (stream) ⟶ lseek (fd, 0, SEEK_SET)

文件描述符 VS. 文件流

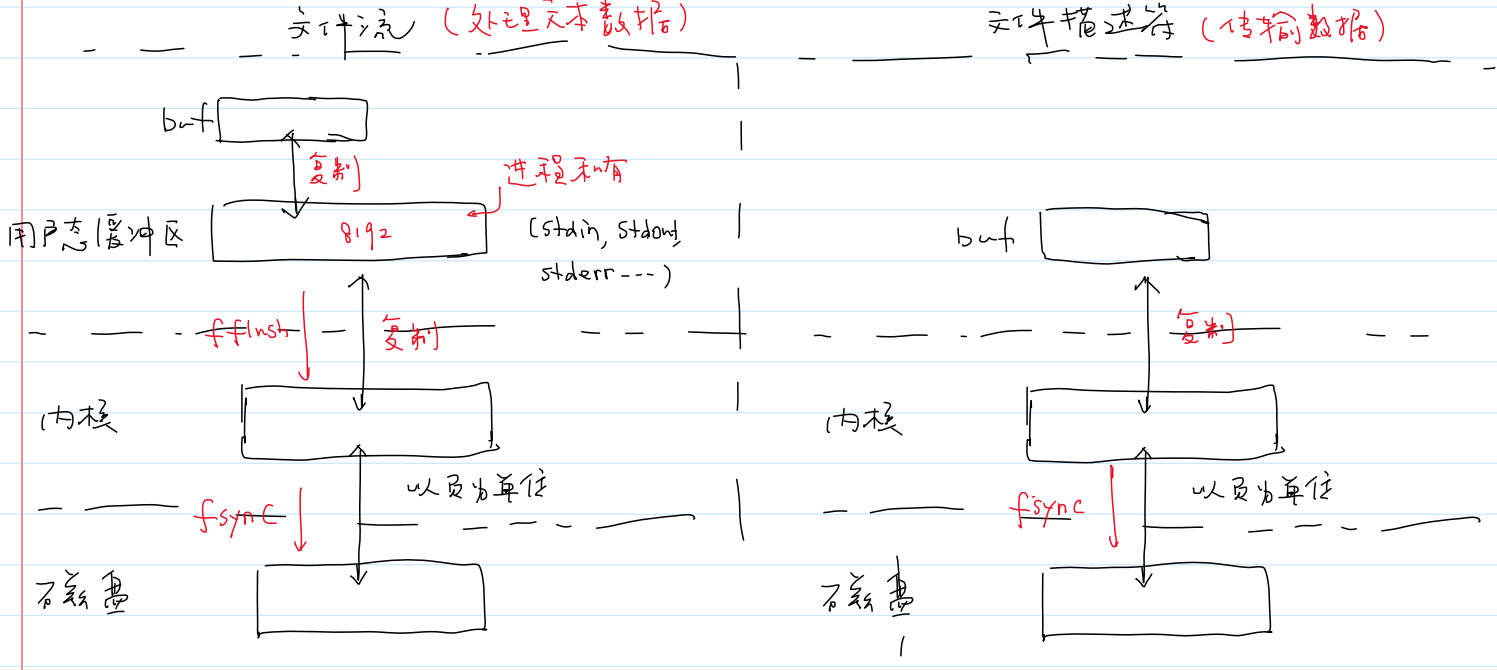
2024年5月20日 10:28

文件流 → ① 可移植性
② 可反使用

fopen
fclose
fread
fwrite
fseek
ftell
rewind

文件描述符

open
close
read
write
lseek
lseek(fd, 0, SEEK_CUR)
lseek(fd, 0, SEEK_SET)



fsync

2024年5月20日 11:11

FSYNC(2) Linux Programmer's Manual FSYNC(2)

NAME

fsync, fdatasync - synchronize a file's in-core state with storage device

SYNOPSIS

```
#include <unistd.h>

int fsync(int fd);
```

→ 将和 fd 相关联的脏页，刷新到磁盘。

↳ 数据库

RETURN VALUE

On success, these system calls return zero. On error, -1 is returned, and errno is set appropriately.

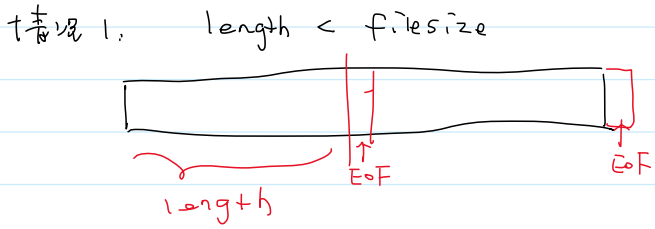
成功：0

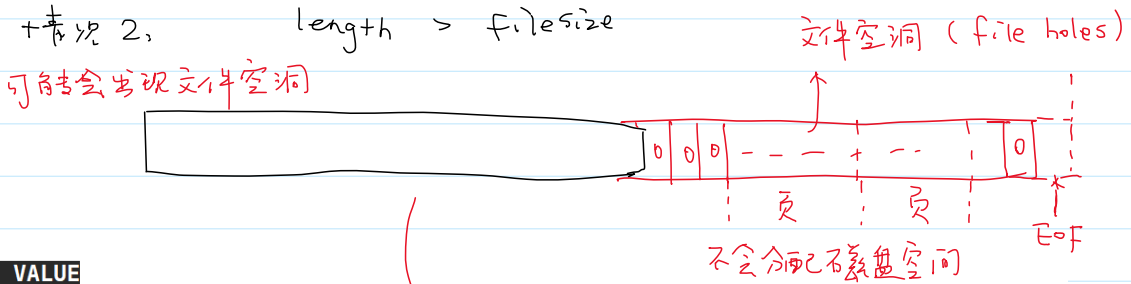
失败：-1，设置 errno

ftruncate

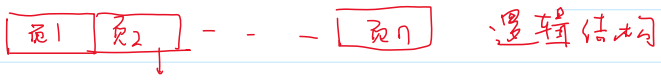
2024年5月20日 11:15

```
TRUNCATE(2)                    Linux Programmer's Manual

NAME
       truncate, ftruncate - truncate a file to a specified length


SYNOPSIS
       #include <unistd.h>
       #include <sys/types.h>

       int ftruncate(int fd, off_t length);
```






```
RETURN VALUE
       On success, zero is returned.  On error, -1 is returned, and errno is set ap-
       propriately.
```



成功：0

失败：-1，设置 errno.

test_ftruncate.c

```c
#include <func.h>

int main(int argc, char* argv[])
{
    // ./test_ftruncate file length
    if (argc != 3) {
        error(1, 0, "Usage: %s file length", argv[0]);
    }

    off_t length;
    sscanf(argv[2], "%ld", &length);

    int fd = open(argv[1], O_WRONLY);
    if (fd == -1) {
        error(1, errno, "open %s", argv[1]);
    }

    if (ftruncate(fd, length) == -1) {
        error(1, errno, "fruncate %d", fd);
    }

    close(fd);

    return 0;
}
```

Block. 512 bytes

```
he@he-vm:~/cpp58/2_Linux/Linux07 (master)$ stat The_Holy_Bible.txt
  File: The_Holy_Bible.txt
  Size: 4351658         Blocks: 8          IO Block: 4096   regular file
Device: 803h/2051d      Inode: 1603095     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/      he)   Gid: ( 1000/      he)
Access: 2024-05-20 11:30:49.373206614 +0800
Modify: 2024-05-20 11:30:33.209901812 +0800
Change: 2024-05-20 11:30:33.209901812 +0800
 Birth: 2024-05-20 11:28:01.261296947 +0800
```

fstat

2024年5月20日 11:37

```
STAT(2)                    Linux Programmer's Manual

NAME
       stat, fstat, lstat, fstatat - get file status

SYNOPSIS
       #include <sys/types.h>
       #include <sys/stat.h>
       #include <unistd.h>

       int fstat(int fd, struct stat *statbuf);

struct stat {
    dev_t     st_dev;         /* ID of device containing file */
    ino_t     st_ino;         /* Inode number */
    mode_t    st_mode;        /* File type and mode */
    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;     /* Block size for filesystem I/O */
    blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */

    /* Since Linux 2.6, the kernel supports nanosecond
       precision for the following timestamp fields.
       For the details before Linux 2.6, see NOTES. */

    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 */

#define st_atime st_atim.tv_sec      /* Backward compatibility */
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
};

RETURN VALUE
       On  success,  zero  is returned.  On error, -1 is returned, and
       errno is set appropriately.
```

保存文件元数信息

↳ 传出参数.

→ vnode

成功：0

失败：-1，设置 errno

test_fstat.c buff

```c
#include <func.h>

int main(int argc, char* argv[])
{
    // ./test_fstat file
    if (argc != 2) {
        error(1, 0, "Usage: %s file", argv[0]);
    }

    int fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        error(1, errno, "open %s", argv[1]);
    }

    struct stat sb; // 存储文件元数据信息
    if (fstat(fd, &sb) == -1) {
        error(1, errno, "fstat %d", fd);
    }

    // 打印 struct stat 里面的成员
    printf("st_ino = %ld\nst_mode=%lo\nst_nlink=%ld\nst_size=%ld\nst_blocks=%ld\n",
           (long)sb.st_ino,
           (long)sb.st_mode,
           (long)sb.st_nlink,
           (long)sb.st_size,
           (long)sb.st_blocks);

    return 0;
}
```

```
he@he-vm:~/cpp58/2_Linux/Linux07 (master)$ ./test_fstat The_Holy_Bible.txt
st_ino = 1603111
st_mode=100664
st_nlink=1
st_size=4351658
st_blocks=8504
```

```
he@he-vm:~/cpp58/2_Linux/Linux07 (master)$ stat The_Holy_Bible.txt
  File: The_Holy_Bible.txt
  Size: 4351658         Blocks: 8504       IO Block: 4096   regular file
Device: 803h/2051d      Inode: 1603111     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/      he)   Gid: ( 1000/      he)
Access: 2024-05-20 14:39:42.058595644 +0800
Modify: 2024-05-20 14:39:42.066595487 +0800
Change: 2024-05-20 14:39:42.066595487 +0800
 Birth: 2024-05-20 14:39:42.058595644 +0800
```

dup

2024年5月20日 14:43

DUP(2) Linux Programmer's Manual

NAME

dup, dup2, dup3 - duplicate a file descriptor

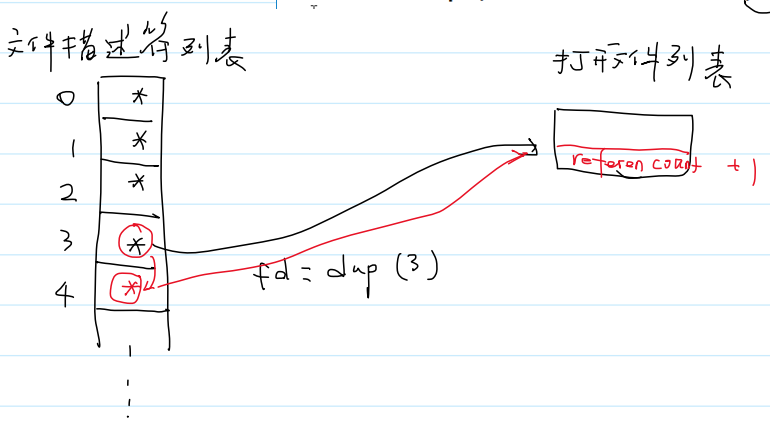
文件描述符的复制

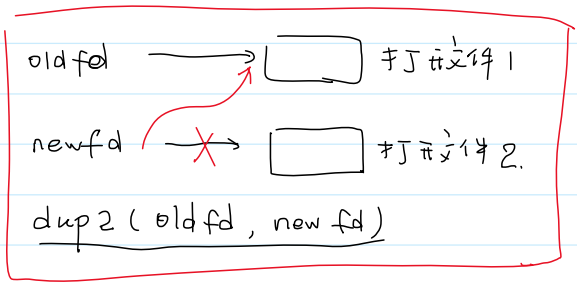
SYNOPSIS

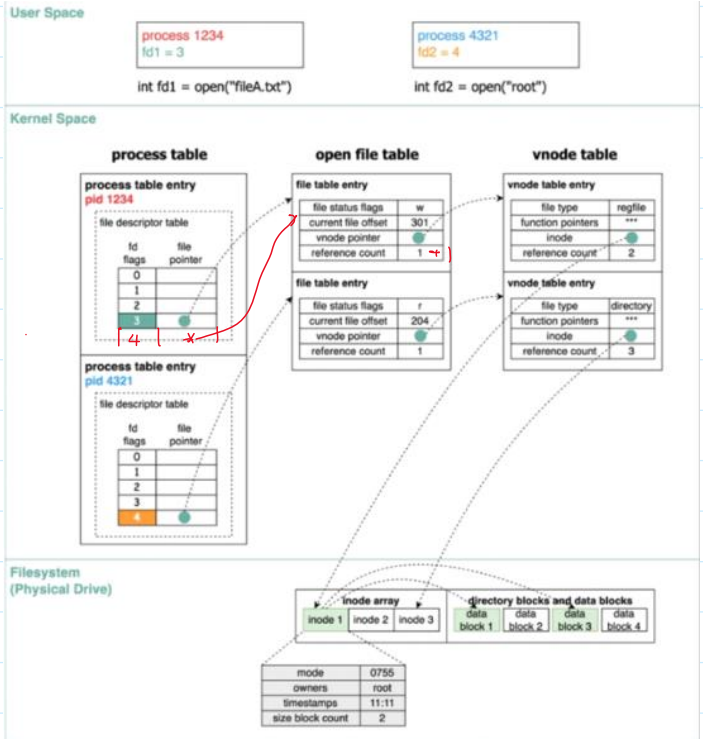
```
#include <unistd.h>

int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

新的文件描述符：最小可用的文件描述符

指定新的文件描述符







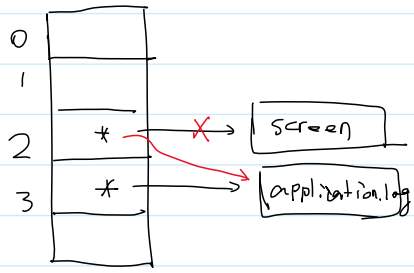
RETURN VALUE

On success, these system calls return the new file descriptor. On error, -1 is returned, and errno is set appropriately.

成功，新的文件描述符

失败：-1，设置 errno

使用：可以用dup实现重定向。



test_dup.c buffers

```c
#include <func.h>

int main(int argc, char* argv[])
{
    // 对 stderr 重定向
    int fd = open("application.log", O_RDWR | O_CREAT | O_APPEND, 0664);
    if (fd == -1) {
        error(1, errno, "open application.log");
    }

    write(STDERR_FILENO, "The first error message\n", 24);
    // 对 stderr 进行重定向
    close(STDERR_FILENO);
    dup(fd);

    write(STDERR_FILENO, "The second error message\n", 25);

    return 0;
}
```

test_dup2.c buffers

```c
#include <func.h>

int main(int argc, char* argv[])
{
    // 对 stderr 重定向
    int fd = open("application.log", O_RDWR | O_CREAT | O_APPEND, 0664);
    if (fd == -1) {
        error(1, errno, "open application.log");
    }

    write(STDERR_FILENO, "The first error message\n", 24);
    // 对 stderr 进行重定向
    // close(STDERR_FILENO);
    if (dup2(fd, STDERR_FILENO) == -1) {
        error(1, errno, "dup2 %d %d", fd, STDERR_FILENO);
    }

    write(STDERR_FILENO, "The second error message\n", 25);

    return 0;
}
```

mmap (零拷贝)

2024年5月20日 15:08

```
MMAP(2)                    Linux Programmer's Manual

NAME
       mmap, munmap - map or unmap files or devices into memory
```

memory map 内存映射　　解除映射

原理　零拷贝

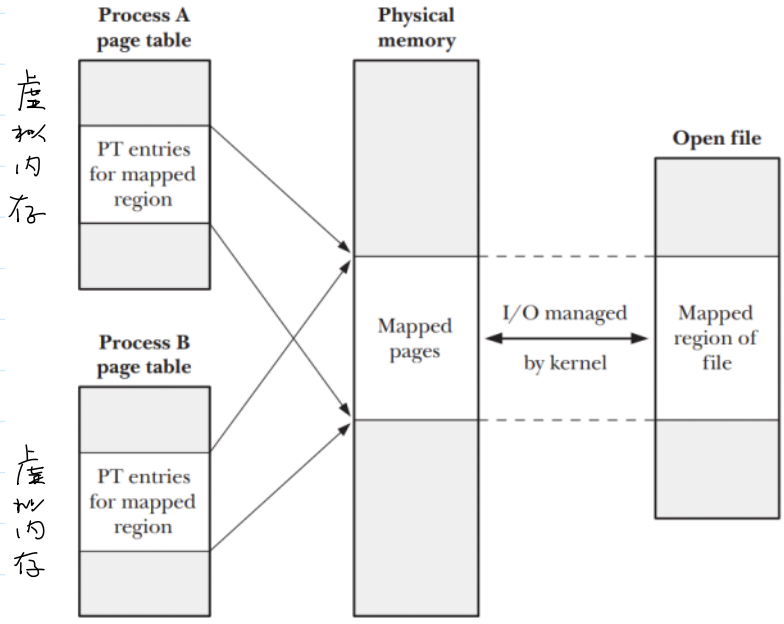
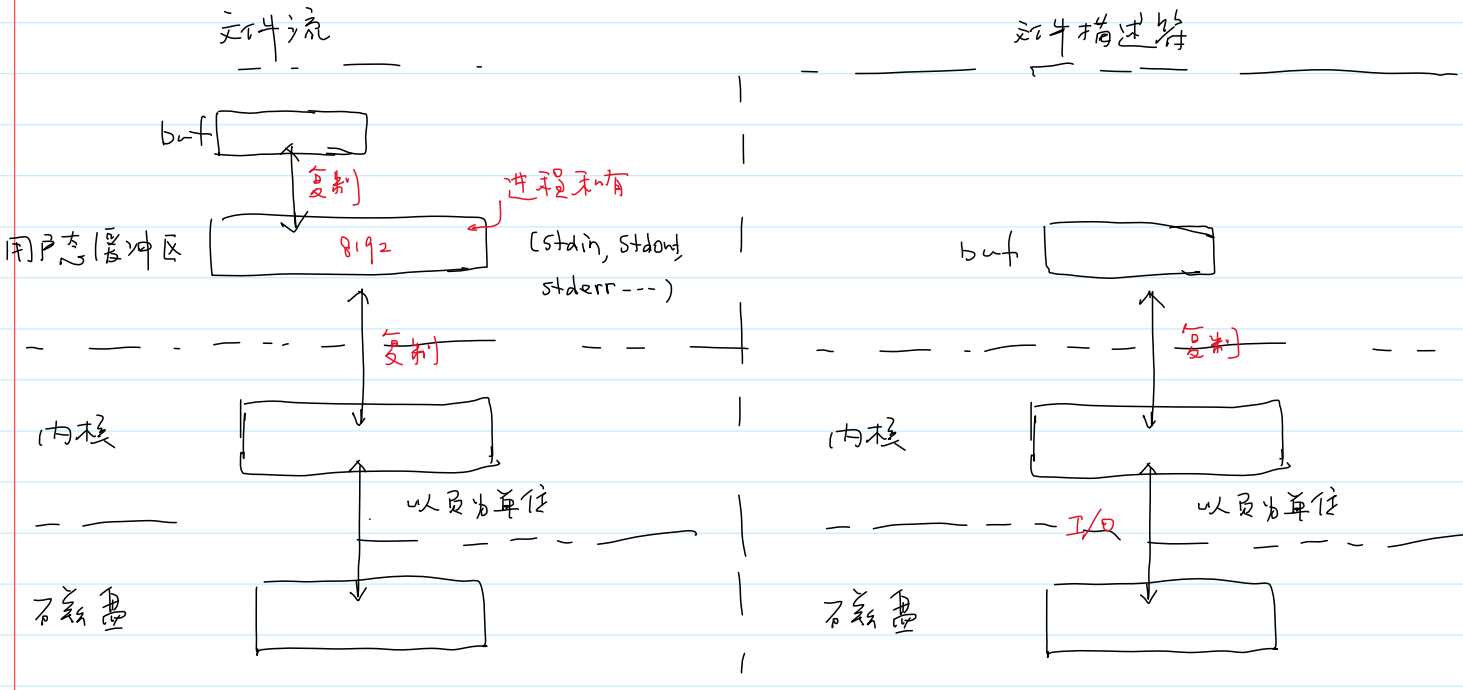
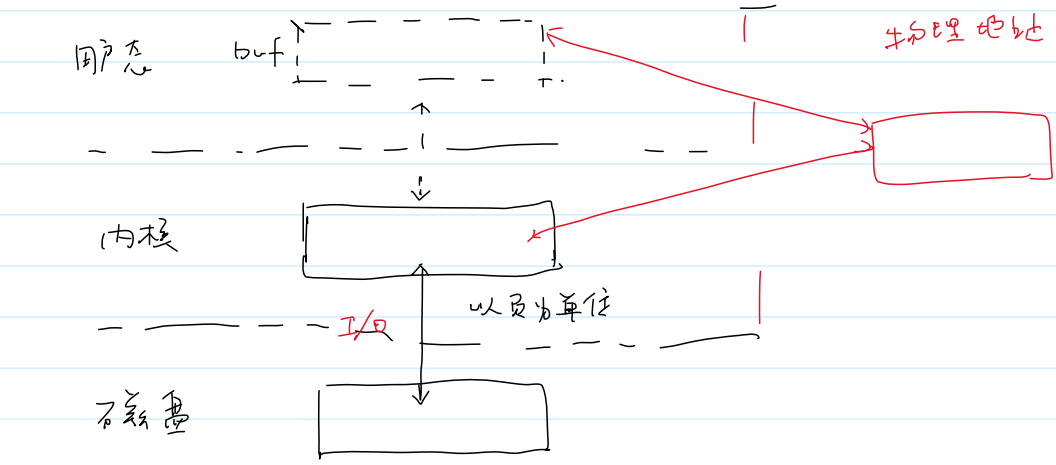


Figure 49-2: Two processes with a shared mapping of the same region of a file



mmap (省去了内核态和用户态之间的拷贝)

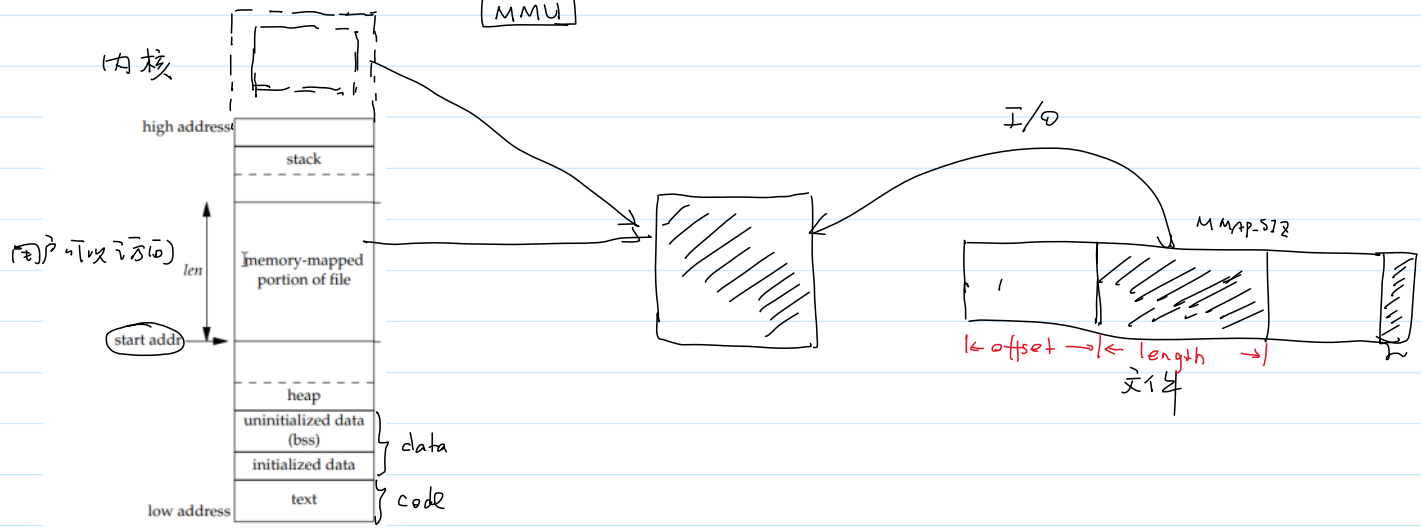
mmap （省去了内核态和用户态之间的拷贝）



进程虚拟内存空间 → 物理内存空间

MMU

磁盘



#2.

SYNOPSIS

memory manage

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int munmap(void *addr, size_t length);
```

起始地址，NULL，表示由内核决定起始地址。

起始地址 start_addr

prot（protection），保护位

PROT_EXEC	Pages may be executed.
PROT_READ	Pages may be read.
PROT_WRITE	Pages may be written.
PROT_NONE	Pages may not be accessed.

flags：标志位，

MAP_SHARED	进程共享的
MAP_PRIVATE	进程私有的（Copy-on-Write：写时复制）

↳ 延迟了复制的时机

RETURN VALUE

On success, **mmap**() returns a pointer to the mapped area. On error, the value **MAP_FAILED** (that is, (void *) -1) is returned, and errno is set to indicate the cause of the error.

On success, **munmap**() returns 0. On failure, it returns -1, and errno is set to indicate the cause of the error (probably to **EINVAL**).

mmap: 成功：映射区的起始地址 (start_addr)

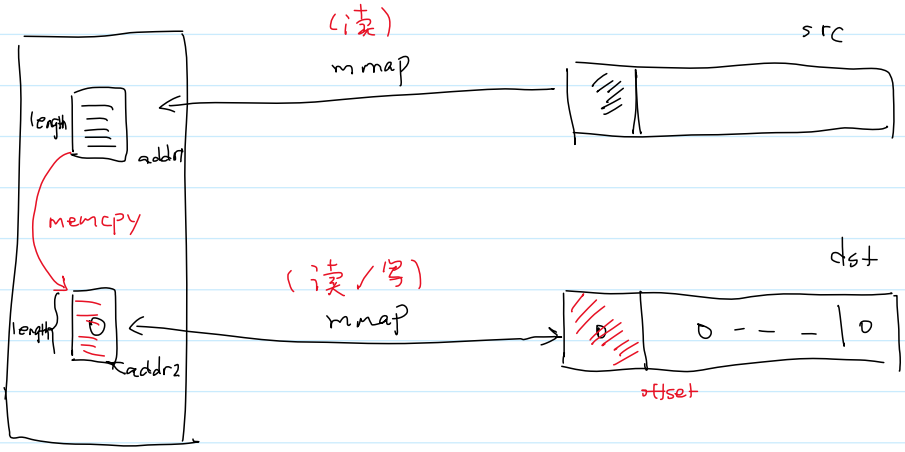
失败：MAP_FAILED，设置 errno

munmap:

成功：0

失败：-1，设置 errno

使用场景：复制大文件



mmap_cp.c

```c
#include <func.h>

#define MMAP_SIZE (4096 * 10)

int main(int argc, char* argv[])
{
    // ./mmap_cp src dst
    if (argc != 3) {
        error(1, 0, "Usage: %s src dst", argv[0]);
    }

    int srcfd = open(argv[1], O_RDONLY);
    if (srcfd == -1) {
        error(1, errno, "open %s", argv[1]);
    }

    int dstfd = open(argv[2], O_RDWR | O_CREAT | O_TRUNC, 0666);
    if (dstfd == -1) {
        close(srcfd);
        error(1, errno, "open %s", argv[2]);
    }

    // 1. 文件大小需要事先固定!
    // 获取src的大小
    struct stat sb;
```

1. offset 应该是页的整数倍

2、读/写

```c
    fstat(srcfd, &sb); // sb.st_size
    off_t fsize = sb.st_size;
    // 将 dst 文件的大小设置为 sb.st_size
    ftruncate(dstfd, fsize);

    off_t offset = 0;   // 已复制的数据
    while (offset < fsize) {
        // 计算映射区的长度
        off_t length;
        if (fsize - offset >= MMAP_SIZE) {
            length = MMAP_SIZE;
        } else {
            length = fsize - offset;
        }

        // 映射
        void* addr1 = mmap(NULL, length, PROT_READ, MAP_SHARED, srcfd, offset);
        if (addr1 == MAP_FAILED) {
            error(1, errno, "mmap %s", argv[1]);
        }

        void* addr2 = mmap(NULL, length, PROT_READ | PROT_WRITE, MAP_SHARED, dstfd, offset);
        if (addr2 == MAP_FAILED) {
            error(1, errno, "mmap %s", argv[2]);
        }

        // addr2 = addr1
        // 复制
        memcpy(addr2, addr1, length);
        offset += length;

        // 解除映射
        int err = munmap(addr1, length);
        if (err) {
            error(1, errno, "munmap %s", argv[1]);
        }

        err = munmap(addr2, length);
        if (err) {
            error(1, errno, "munmap %s", argv[2]);
        }
    }   // offset == fsize

    return 0;
}
```

3. 文件的大小需要事先固定

4. 读/写

5. 就可以通过内存复制，实现文件的复制

6. 不要忘记解除映射！

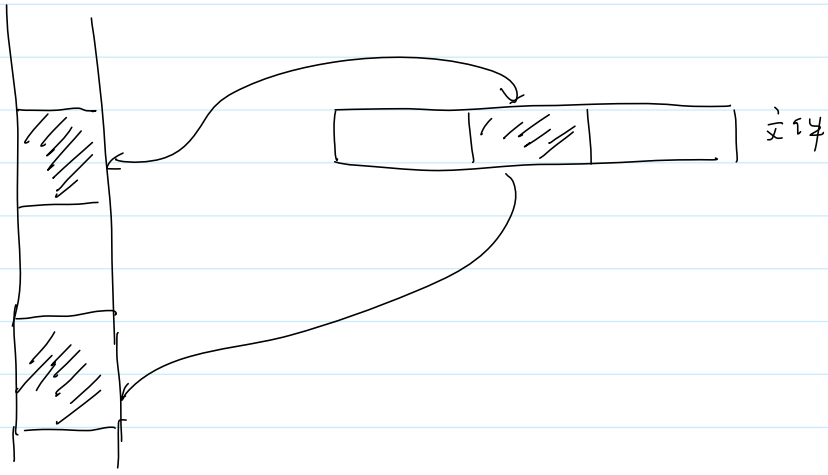
常见问题

2024年5月20日 21:24

#1. mmap

flags: MAP_SHARED

MAP_PRIVATE （写时复制） （x） （不会写入底层的文件）



#2. copy_holes.c

核心问题：判断哪里有空洞

（一整页都是空字符） → 空洞

↳ memcmp

```
char empty[PAGESIZE] = {0};
char buf[PAGESIZE];
```

预告

2024年5月20日 21:19

① 讲作业.

② CPU虚拟化 （[illegible]）

③ fork

④ /_exit / atexit / abort / wait / waitpid.