Lab5

> 班级：222111
>
> 学号：22373340
>
> 姓名：詹佳博

思考题

Thinking 5.1

- kseg0存放内核，用cache访问。如果读写设备的时候也经过cache，则可能导致之后访问内核时，访问到设备的信息而错误。
- 同时，外设的访问一般不存在固定多用的一个设备，可能鼠标、键盘等交替使用，这个时候cache会进行高刷新，miss变多，效率变低，反而失去了cache的功能。

Thinking 5.2

- user/include/fs.h 中定义 FILE2BLK (BLOCK_SIZE / sizeof(struct File)) 即**16**，MAXPATHLEN 1024即一个目录最多指向**1024**个磁盘块。故而一个目录下最多能有 1024*16=**16384** 个子文件。单个文件最大为4KB*1024=**4MB**。

Thinking 5.3

- `./fs/serv.h:15:#define DISKMAX 0x40000000`，也就是1GB。

Thinking 5.4

- #define DISKMAX 0x40000000，#define DISKMAP 0x10000000定义了最大磁盘大小和磁盘初始映射位置。
  - 他们主要用于内存的分配。
- #define MAXPATHLEN 1024定义了目录下最多磁盘块，#define MAXFILESIZE (NINDIRECT * BLOCK_SIZE)定义了单个文件最大大小。#define FTYPE_REG 0定义了普通文件，#define FTYPE_DIR定义了目录文件
  - 他们主要用于用户操作。

Thinking 5.5

文件描述符和定位指针均在用户空间实现，所以fork前后的父子进程**会共享它们**。

- 修改lab5_4测试代码如下：

```
#include <lib.h>

static char *msg = "This is the NEW message of the day!\n";
static char *diff_msg = "This is a different message of the day!\n";

int main() {
    int r;
    int fdnum;
    char buf[512];
```

```
    int n;

    if ((r = open("/newmotd", O_RDWR)) < 0) {
        user_panic("cannot open /newmotd: %d", r);
    }
    fdnum = r;
    debugf("open is good\n");

    if ((n = read(fdnum, buf, 511)) < 0) {
        user_panic("cannot read /newmotd: %d", n);
    }
    if (strcmp(buf, diff_msg) != 0) {
        user_panic("read returned wrong data");
    }
    debugf("read is good\n");

    //new code
    if (fork()) {
        debugf("child_fd == %d\n",r);
        struct Fd *fdd;
        fd_lookup(r,&fdd);
        debugf("child_fd's offset == %d\n",fdd->fd_offset);
    }
    else {
        debugf("father_fd == %d\n",r);
        struct Fd *fdd;
        fd_lookup(r,&fdd);
        debugf("father_fd's offset == %d\n",fdd->fd_offset);
    }
}
```

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
FS is running
superblock is good
read_bitmap is good
open is good
read is good
child_fd == 0
child_fd's offset == 511
father_fd == 0
father_fd's offset == 511
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
[00001802] destroying 00001802
[00001802] free env 00001802
i am killed ...
panic at sched.c:51 (schedule): schedule: no runnable envs
ra:    8002605c  sp:  803ffe80  Status: 00008000
Cause: 00000420  EPC: 00402fe0  BadVA:  7fd7f004
curenv:    NULL
cur_pgdir: 83fce000
```

Thinking 5.6

1. **struct Fd**
   - `u_int fd_dev_id;`：标识外部设备id。
   - `u_int fd_offset;`：表示当前位置（偏移量），用于读写操作，例如在 `fseek` 中使用。
   - `u_int fd_omode;`：指定文件的打开模式，如只读、只写等。
2. **struct Filefd**
   - `struct Fd f_fd;`：表示与文件关联的文件描述符。
   - `u_int f_fileid;`：标识文件id。
   - `struct File f_file;`：指向与文件对应的文件系统控制块。
3. **struct Open**
   - `struct File *o_file;`：指针，指向具体的 `file` 结构。
   - `u_int o_fileid;`：标识文件id。
   - `int o_mode;`：指定文件的打开模式，表示只读、只写等。
   - `struct Filefd *o_ff;`：表示打开位置的偏移量。
4. **超级块**：用于存储文件系统的元数据信息，比如文件系统的大小、空闲块列表、根目录位置等。超级块通常位于磁盘的固定位置，是文件系统的入口点。
5. **位图**：用于跟踪磁盘块的分配情况，记录哪些磁盘块已被占用，哪些是空闲的。位图通常存储在磁盘的某个区域，其结构与磁盘块的布局相对应。
6. **文件控制块**：用于存储文件的元数据信息，比如文件名、大小、权限等。每个文件在文件系统中都对应一个文件控制块，用于管理文件的状态和属性。

Thinking 5.7

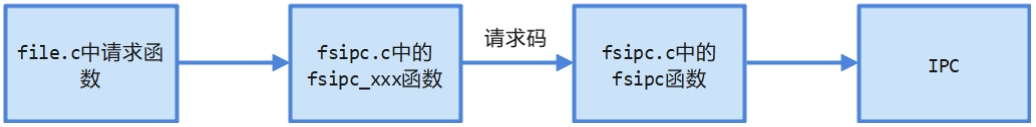
- **发送消息**，用实三角箭头黑实线表示。消息发送者把进程控制传递给消息的接收者，然后**等待**消息接收者的回复消息。

  返回消息，用开三角箭头黑色虚线表示。返回消息和发送消息结合使用，用来接收发送消息的值，处理后返回给发送方。

- 以 `open` 方法为例：
  1. 在用户进程中，首先调用 `open` 方法，该方法会触发一系列操作。
  2. 在 `open` 方法内部，首先调用 `fd_alloc` 分配一个文件描述符（`fd`），然后调用 `fdipc_open` 函数，并将新分配的文件描述符作为参数传递给该函数。
  3. 在 `fdipc_open` 函数中，会调用 `fsipc_open` 函数，并将用户进程请求的服务类型以及文件描述符等信息传递给 `fsipc_open`。
  4. 在 `fsipc_open` 函数中，会设置请求服务类型，并调用 `ipc_send` 方法将请求传输给文件服务进程。
  5. 文件服务进程接收到请求后进行相应的处理，并准备要传回给用户进程的信息。文件服务进程通过适当的处理，将用户进程请求的服务信息传回。
  6. 在 `open` 方法中，接收到文件服务进程传回的信息，将相关信息存储到文件描述符结构体（`fd`）中，完成对文件的打开操作。

难点分析

1. 完成 Exercise 5.1 的时候 `pa >= 0x180003f8 && pa + len <= 0x180003f8 + 0x20` 的后面一个条件没加等号，debug了半天，发现自己植树问题没学好。。。
2. 在 Exercise 5.4 中，blockno不能为0。在文件系统中，通常会将第一个磁盘块（块号为 0）作为保留块，用于存储文件系统的元数据，如超级块、位图等。因此，我们不应该将块号为 0 的磁盘块视为可分配的空闲块。如果 `blockno` 的值为 0，则意味着我们试图释放保留块，这是不允许的，因为这可能会导致文件系统结构的损坏。
3. Exercise 5.8需要对nblock向上取整。
4. 有关文件系统进行文件操作的流程如下：



实验体会

1. Lab5-2的上机测试相当于对用户级别的操作添加了copy和chmod。总体而言都比较简单，这方面自己的理解也比较全面，做起来也是比较顺利，但是在debug copy函数的时候，一直因为自己的笔误把dst_file打成了dst，最后经过助教点拨打印文件路径才得以发现这个bug。。。
2. 发现了一个更优美的写法：nblock = ROUND(src->f_size, BLOCK_SIZE) / BLOCK_SIZE;这是课上的代码，让我了解到有时候活用宏定义的函数，能加快代码理解。而自己在课下写的就是用常规取ceil方法：x/y=(x+y-1)/y。这种代码可能不方便人去理解。
3. os的上机考试也告一段落了。每次上机其实还是觉得有所遗憾，有时就差那几分钟甚至几秒钟就能提交extra。可能人生就是这样，错过了就不会再有弥补的机会了。但是对自己更深刻了解os的知识有所帮助也算是收获颇丰吧！