Exercise 1. i386_init identifies the file system environment by passing the type ENV_TYPE_FS to your environment creation function, env_create. Modify env_create in env.c, so that it gives the file system environment I/O privilege, but never gives that privilege to any other environment.

```
void env_create(uint8_t *binary, size_t size, enum EnvType type)
{
    // If this is the file server (type == ENV_TYPE_FS) give it I/O privileges.
    // LAB 5: Your code here.
    ……
     // #define FL_IOPL_MASK    0x00003000   // I/O Privilege Level bitmask
     // FL_IOPL_3 == FL_IOPL_MASK
     if (type == ENV_TYPE_FS)
         env->env_tf.tf_eflags |= FL_IOPL_MASK;
}
```

代码实现：

检查 type 是否为 ENV_TYPE_FS ,即该创建出的 ENV 是否为 File System，如果是的话，那么交给其 I/O 权限

Exercise 2. Implement the bc_pgfault and flush_block functions in fs/bc.c.

```
static void bc_pgfault(struct UTrapframe *utf)
{
    ......
    // addr may not be aligned to a block boundary, round the addr down
    addr = ROUNDDOWN(addr, PGSIZE);
    // Allocate a page in the disk map region
    if (sys_page_alloc(0, addr, PTE_W | PTE_U | PTE_P) < 0)
        panic("pc_pgfault: can't alloc page\n");
    // read the contents of the block from the disk into that page
    if (ide_read(blockno*BLKSECTS, addr, BLKSECTS) < 0)
        panic("bc_pagfault: ide_read error\n");
    // mark the page not-dirty
    if (sys_page_map(0, addr, 0, addr, PTE_SYSCALL) < 0)
        panic("bc_pagfault: sys_page_map error");
    ......
}
```

```
void flush_block(void *addr)
{
    ......
    // addr may not be aligned to a block boundary, so round the addr
down
    addr = ROUNDDOWN(addr, PGSIZE);

    // If the block is not in the block cache or is not dirty, does nothing.
    if (va_is_mapped(addr) && va_is_dirty(addr))
    {
        // Flush the contents of the block containing VA out to disk
        if (ide_write(blockno*BLKSECTS, addr, BLKSECTS) < 0)
            panic("flush_block: ide_write error");
        // clear the PTE_D bit using sys_page_map
        if (sys_page_map(0, addr, 0, addr, PTE_SYSCALL) < 0)
            panic("flush_block: sys_page_map error");
    }
}
```

代码实现：

bc_pgfault:

先向下对齐 addr

用 sys_page_alloc 在 addr 处开出一块 page

用 ide_read 从 blockno 这个 sector 读出内容到 addr 位于的 page，此时这个 page 已经 dirty（代表被修改过）

最后用 sys_page_map 把 addr 所在 page 重新置为不 dirty 的

flush_block:

先向下对齐 addr

检查这个 addr 处于的 page 是否已经被 map 或者 dirty（被改动过），如果是的话那么用 ide_write 将 addr 处 page 的内容写到 blockno 这个 sector，用 sys_page_map 重新设置 addr 所在的 page

Exercise 3. Use free_block as a model to implement alloc_block, which should find a free disk block in the bitmap, mark it used, and return the number of that block. When you allocate a block, you should immediately flush the changed bitmap block to disk with flush_block, to help file system consistency.

```
int alloc_block(void)
{
    // The bitmap consists of one or more blocks.  A single bitmap block
    // contains the in-use bits for BLKBITSIZE blocks.  There are
    // super->s_nblocks blocks in the disk altogether.
    uint32_t i;
    for (i = 0; i < super->s_nblocks; i++)
        if (block_is_free(i))
        {
            // uint32_t *bitmap;
            // the num in bitmap is 0 when used, and 1 when free
            bitmap[i/32] &= ~((int)1 << (i % 32));
            flush_block(bitmap);
            return i;
        }
    return -E_NO_DISK;
}
```

代码实现：

从 0 开始逐个检查所有的 block 是否为 free 的，如果是 free 的那就把这个 block 置为不 free 的（修改 bitmap 位），之后用 flush_blcok 把提交 bitmap 的修改，之后返回

Exercise 4. Implement file_block_walk and file_get_block.

```
static int file_block_walk(struct File *f, uint32_t filebno, uint32_t
**ppdiskbno, bool alloc)
{
    // filebno is out of range (it's >= NDIRECT + NINDIRECT).
    if(filebno >= NDIRECT + NINDIRECT) return -E_INVAL;

    // if the filebno is direct
    if (filebno < NDIRECT)
    {
        *ppdiskbno = &(f->f_direct[filebno]);
        return 0;
    }

    // if the filebno is indiredt
    // there is no indirect block, alloc an indriect block index
    if(f->f_indirect == 0)
    {
        // the function needed to allocate an indirect block, but alloc was
0.
        if(alloc == 0) return -E_NOT_FOUND;
```

```
        // alloc a block
        int r = alloc_block();

        // alloc fail, there's no space on the disk for an indirect block.
        if(r < 0) return -E_NO_DISK;

        // initialize the block
        memset(diskaddr(r), 0, BLKSIZE);
        f->f_indirect = r;
        flush_block(diskaddr(r));
    }

    // get the addr
    uint32_t* indirect = diskaddr(f->f_indirect);
    *ppdiskbno = &(indirect[filebno-NDIRECT]);
    return 0;
}
```

```
int file_get_block(struct File *f, uint32_t filebno, char **blk)
{
    uint32_t * ppdiskbno = NULL;

    // the problem can be detected in file_block_walk
    int r = file_block_walk(f, filebno, &ppdiskbno, 1);
    if(r < 0) return r;

    // the block is not alloced, need alloc block point's block
    if(*ppdiskbno == 0)
    {
        r = alloc_block();
        // a block needed to be allocated but the disk is full.
        if(r < 0) return -E_NO_DISK;
        *ppdiskbno = r;
        // initialize the block
        memset(diskaddr(r), 0, BLKSIZE);
        flush_block(diskaddr(r));
    }
    *blk = diskaddr(*ppdiskbno);
    return 0;
}
```

代码实现：

file_block_walk:

首先检查 filebno 是否越界

如果 filebno 为 direct block，那直接取出其地址并返回

如果 filebno 为 indirect block，检查 indrect block index 是否存在

如果不存在且不 alloc，那返回错误信息

如果不存在但是 alloc，则新开一块 block 作为 indirect block index，把这块 block 内容置为 0，并 flush 写入 disk

这样就有 indirect block index 了，最后把 filebno 位于 indirect block index 中的地址取出并返回

file_get_block:

用 file_block_walk 取出 filebno 的地址（错误检查都在 file_block_walk 中完成）

如果该地址为空，那么说明这个 block 没有被用到，还没被创建出来，用 alloc_block 创建出一个新的 block，把这块 block 内容置为 0，并 flush 写入 disk，把地址设为这个 block 的位置。

这样 filebno 的地址就一定不为空。

最后把地址取出并返回。

Exercise 5. Implement serve_read in fs/serv.c and devfile_read in lib/file.c.

```
int serve_read(envid_t envid, union Fsipc *ipc)
{
    struct Fsreq_read *req = &ipc->read;
    struct Fsret_read *ret = &ipc->readRet;

    if (debug)
        cprintf("serve_read %08x %08x %08x\n", envid, req->req_fileid, req->req_n);

    int r;
    struct OpenFile *o;

    // Look up an open file for envied.
    r = openfile_lookup(envid, req->req_fileid, &o);
    if (r < 0) return r;

    // read the file from fd_offset into ret_buf, the max size is PGSIZE, if req_n is larger than PGSIZE, only read PGSIZE
    r = file_read(o->o_file, ret->ret_buf, MIN(req->req_n, PGSIZE), o->o_fd->fd_offset);
```

```
	if (r < 0) return r;

	// refresh the fd_offset
	o->o_fd->fd_offset += r;
	return r;
}

static ssize_t devfile_read(struct Fd *fd, void *buf, size_t n)
{
	int r;
	// init, get file id and n
	fsipcbuf.read.req_fileid = fd->fd_file.id;
	fsipcbuf.read.req_n = n;

	r = fsipc(FSREQ_READ, NULL);
	if (r < 0) return r;

	// get content from fsipcbuf to buf
	memmove(buf, &fsipcbuf, r);
	return r;
}
```

代码实现：

serve_read:

用 req_fileid 代表的文件 id 取出该文件

从该文件中的 fd_offset 处读取 req_n（ req_n 不能比 PGSIZE 大，最大读 PGSIZE）放入 ret_buf

最后更新 fd_offset

devfile_read:

把文件 id 与要读的长度 n 写入 req 中，并发送要求

把读取到的 fsipcbuf 中的内容写到 buf 中

Exercise 6. Implement serve_write in fs/serv.c and devfile_write in lib/file.c.

```
int serve_write(envid_t envid, struct Fsreq_write *req)
{
    if (debug)
        cprintf("serve_write %08x %08x %08x\n", envid, req->req_fileid, req->req_n);

    int r;
    struct OpenFile *o;

    // open the fileid to o
    r = openfile_lookup(envid, req->req_fileid, &o);
    if (r < 0) return r;

    // write req_n from req_buf to the file from fd_offset
    r = file_write(o->o_file, req->req_buf, req->req_n, o->o_fd->fd_offset);
    if (r < 0) return r;

    // refresh the fd_offset
    o->o_fd->fd_offset += r;
```

```
    return r;
}

static ssize_t devfile_write(struct Fd *fd, const void *buf, size_t n)
{
    int r;

    // init, get file id and the pointer of the content
    fsipcbuf.write.req_fileid = fd->fd_file.id;
    void *p = (void*)buf;

    while (n)
    {
        // should not write more than the req_buf
        fsipcbuf.write.req_n = MIN(n, sizeof(fsipcbuf.write.req_buf));
        // move the content in the p(buf+offset) to the req_buf
        memmove(fsipcbuf.write.req_buf, p, fsipcbuf.write.req_n);
        // send wirte message
        r = fsipc(FSREQ_WRITE, NULL);
        if (r < 0) return r;
        n -= r;
        p += r;
```

```
    }

    // return the size writed
    return (ssize_t)(p-buf);
}
```

代码实现：

serve_write:

与 serve_read 相似

用 req_fileid 代表的文件 id 取出该文件

从 ret_buf 中向该文件位于 fd_offset 处写入 req_n 长度的内容

最后更新 fd_offset

devfile_write:

与 devfile_read 类似

把文件 id 与要读的长度 n 写入 req 中，并发送要求

每次写入最大 req_buf 大小的内容到到 req_buf 中并发送写入请求，直至所有 buf 中的内容都被写入

Exercise 7. Implement open.

```
int open(const char *path, int mode)
{
	int r;
	struct Fd *fd;

	// whether the path is too long (>= MAXPATHLEN)
	if (strlen(path) >= MAXPATHLEN) return -E_BAD_PATH;

	// returns an unused fd address.
	r = fd_alloc(&fd);
	if (r < 0) return r;

	// copy path & mode to fs ipc buf
	strcpy(fsipcbuf.open.req_path, path);
	fsipcbuf.open.req_omode = mode;

	// send open message with fd
	r = fsipc(FSREQ_OPEN, fd);
	if (r < 0)
	{
		// close the file when an error occour
```

```
        fd_close(fd, 0);
        return r;
    }

    // return the number of fd(0,1,2.....)
    return fd2num(fd);
}
```

代码实现：

检查 path 长度

用 fd_alloc 找到一个空闲的 fd

把 path 和 mode 放入 fsipcbuf，并发送 open 请求把内容放的 fd 中

返回 fd 的 number

Exercise 8. spawn relies on the new syscall sys_env_set_trapframe to initialize the state of the newly created environment. Implement sys_env_set_trapframe.

```
static int sys_env_set_trapframe(envid_t envid, struct Trapframe *tf)
{
    // Remember to check whether the user has supplied us with a good
    // address!
    int r;
    struct Env* e;
    r = envid2env(envid, &e, 1);
    if (r < 0) return -E_BAD_ENV;

    // FL_IF: interrupt flag,which means the env can be interrupted
    // Set envid's trap frame to 'tf'.
    e->env_tf = *tf;
    // user environments always run at code protection level 3 (CPL 3)
    e->env_tf.tf_cs |= 3;
    // enable interrupt
    e->env_tf.tf_eflags |= FL_IF;
    return 0;
}
```

```
int32_t syscall(uint32_t syscallno, uint32_t a1, uint32_t a2, uint32_t a3,
uint32_t a4, uint32_t a5)
{
    switch (syscallno){
        ……
        case SYS_env_set_trapframe:
            return sys_env_set_trapframe((envid_t)a1,(struct
Trapframe*)a2);
        ……
    }
}
```

代码实现：

在 syscall 中注册

取出 envid 代表的 env 到 e 中

设置 e 的 env_tf 为 tf，tf_cs 有 CPL 3 的权限，且 eflag 设置为可以被中断