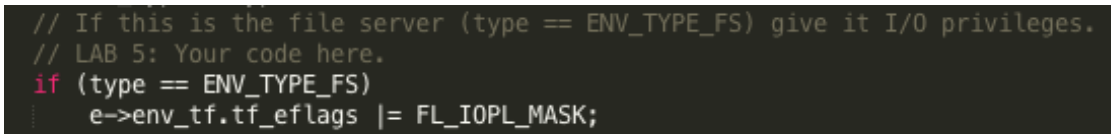
Lab 5 实验报告

姓名：李浩宇

学号：5130809070

Exercise 1. 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.

根据注释，在 env_create 函数中对 type 进行判断并加上 FL_IOPL_MASK 即可。

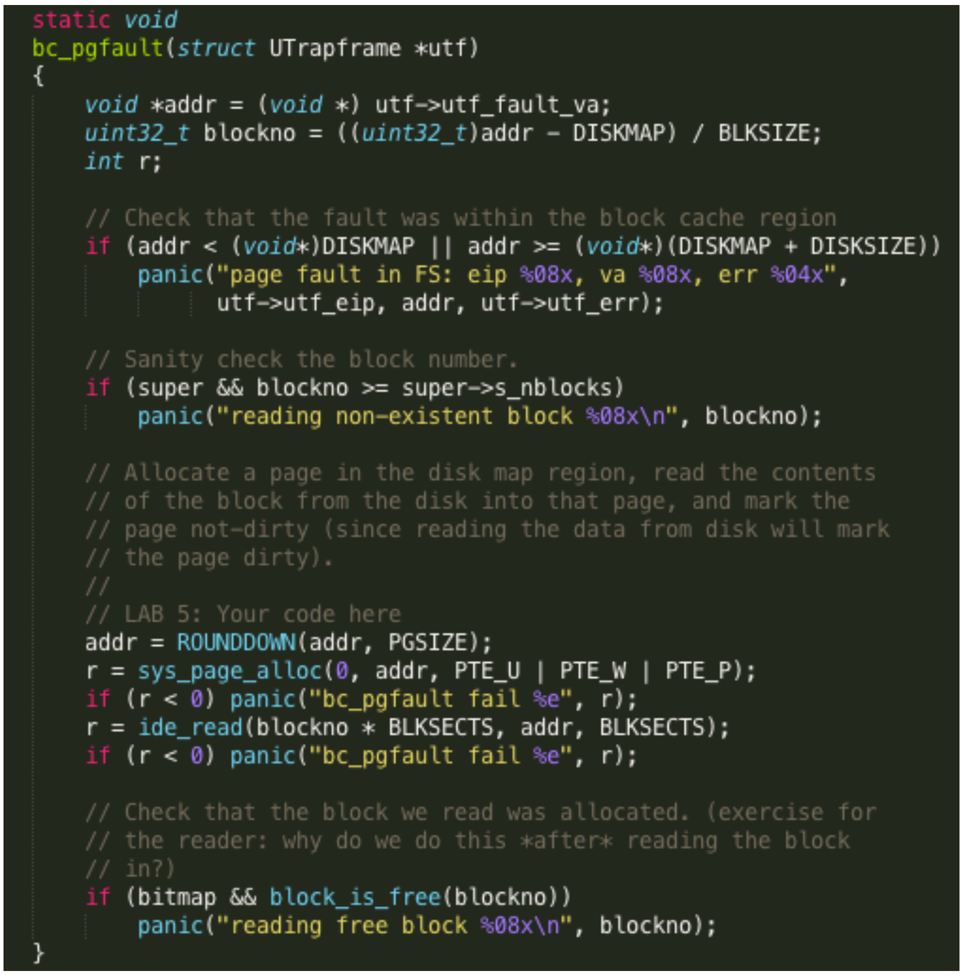


```
// If this is the file server (type == ENV_TYPE_FS) give it I/O privileges.
// LAB 5: Your code here.
if (type == ENV_TYPE_FS)
    e->env_tf.tf_eflags |= FL_IOPL_MASK;
```

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

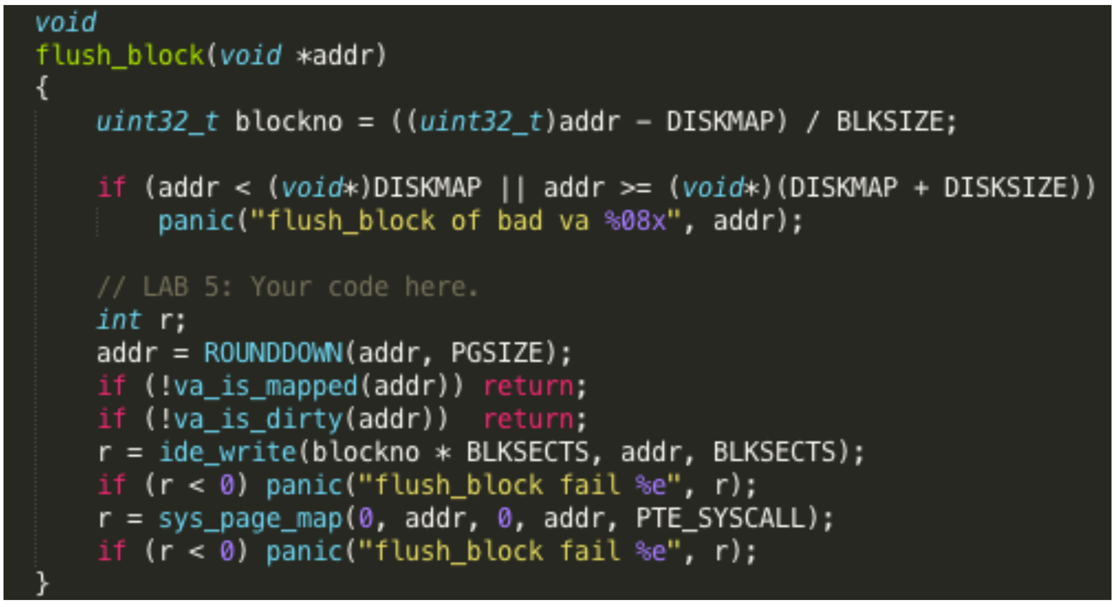
这两个函数的注释十分地详细，需要调用的函数接口也给与了提示，将注释翻译成代码即可。

bc_pgfault:



```
static void
bc_pgfault(struct UTrapframe *utf)
{
    void *addr = (void *) utf->utf_fault_va;
    uint32_t blockno = ((uint32_t)addr - DISKMAP) / BLKSIZE;
    int r;

    // Check that the fault was within the block cache region
    if (addr < (void*)DISKMAP || addr >= (void*)(DISKMAP + DISKSIZE))
        panic("page fault in FS: eip %08x, va %08x, err %04x",
              utf->utf_eip, addr, utf->utf_err);

    // Sanity check the block number.
    if (super && blockno >= super->s_nblocks)
        panic("reading non-existent block %08x\n", blockno);

    // Allocate a page in the disk map region, read the contents
    // of the block from the disk into that page, and mark the
    // page not-dirty (since reading the data from disk will mark
    // the page dirty).
    //
    // LAB 5: Your code here
    addr = ROUNDDOWN(addr, PGSIZE);
    r = sys_page_alloc(0, addr, PTE_U | PTE_W | PTE_P);
    if (r < 0) panic("bc_pgfault fail %e", r);
    r = ide_read(blockno * BLKSECTS, addr, BLKSECTS);
    if (r < 0) panic("bc_pgfault fail %e", r);

    // Check that the block we read was allocated. (exercise for
    // the reader: why do we do this *after* reading the block
    // in?)
    if (bitmap && block_is_free(blockno))
        panic("reading free block %08x\n", blockno);
}
```

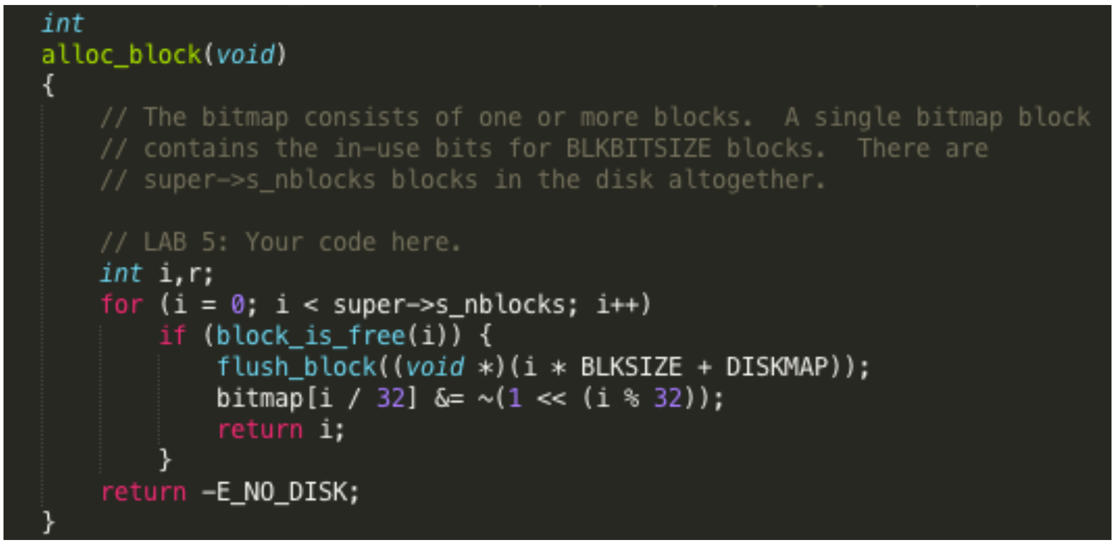
flush_block:



```c
void
flush_block(void *addr)
{
    uint32_t blockno = ((uint32_t)addr - DISKMAP) / BLKSIZE;

    if (addr < (void*)DISKMAP || addr >= (void*)(DISKMAP + DISKSIZE))
        panic("flush_block of bad va %08x", addr);

    // LAB 5: Your code here.
    int r;
    addr = ROUNDDOWN(addr, PGSIZE);
    if (!va_is_mapped(addr)) return;
    if (!va_is_dirty(addr))  return;
    r = ide_write(blockno * BLKSECTS, addr, BLKSECTS);
    if (r < 0) panic("flush_block fail %e", r);
    r = sys_page_map(0, addr, 0, addr, PTE_SYSCALL);
    if (r < 0) panic("flush_block fail %e", r);
}
```

Exercise 3. Use free_block as a model to implement alloc_block.

简单地遍历一遍所有 block，利用 block_is_free 函数找到一块空闲的 block，然后参照 free_block 中操作 bitmap 的方式将其中对应的 bit 置为 1，同时根据注释的提示，对 block 进行 flush 操作。



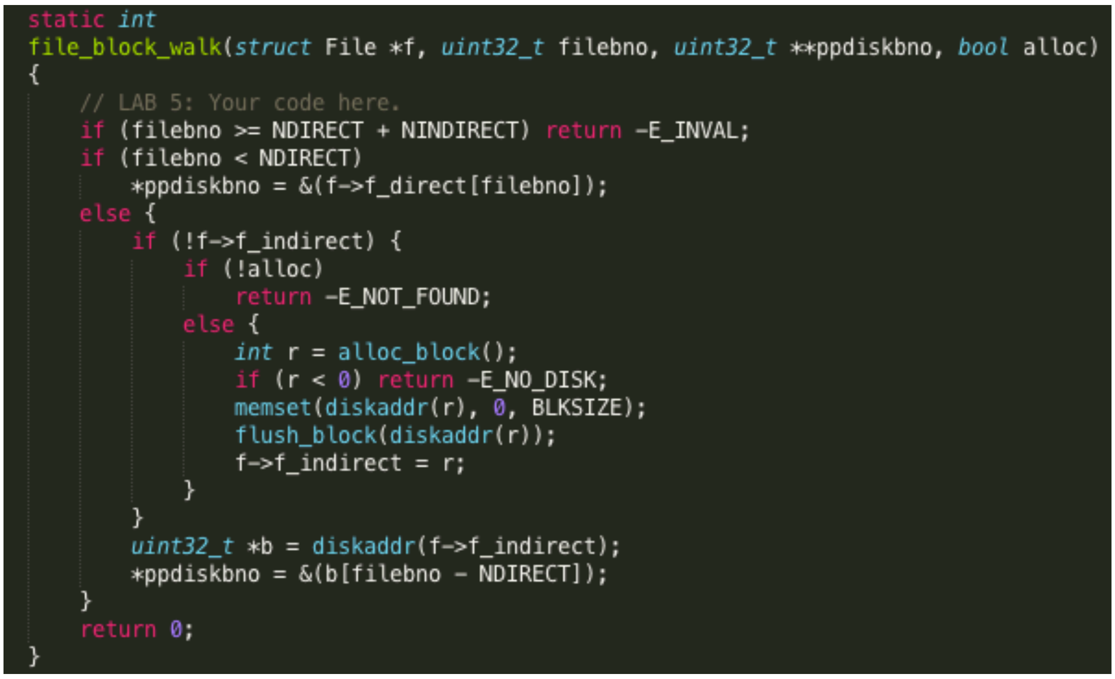
```c
int
alloc_block(void)
{
    // The bitmap consists of one or more blocks.  A single bitmap block
    // contains the in-use bits for BLKBITSIZE blocks.  There are
    // super->s_nblocks blocks in the disk altogether.

    // LAB 5: Your code here.
    int i,r;
    for (i = 0; i < super->s_nblocks; i++)
        if (block_is_free(i)) {
            flush_block((void *)(i * BLKSIZE + DISKMAP));
            bitmap[i / 32] &= ~(1 << (i % 32));
            return i;
        }
    return -E_NO_DISK;
}
```

Exercise 4. Implement file_block_walk and file_get_block.

一开始对 file_block_walk 函数的理解有点偏差，以为是不仅要找到 filebno 对应的 slot 还要完成相应的分配工作，导致浪费了一些时间。后来发现原来只需要找到相应的 slot 即可，不管文件当前有没有分配到这么多的 block（当然如果是 indirect block 的话还是需要

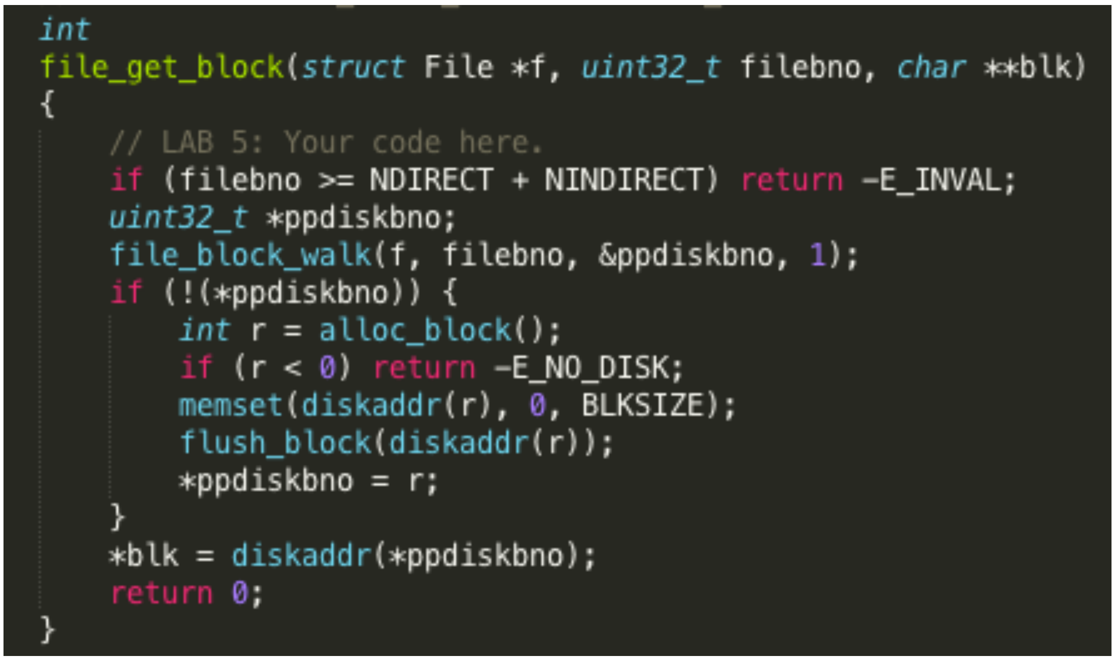
根据 alloc 参数来完成分配），正确理解之后就容易多了，各个需要的函数之前也都实现好或者已有提供。file_get_block 大题逻辑也非常相似，返回 filebno 对应的 slot 即可。

file_block_walk:



```c
static int
file_block_walk(struct File *f, uint32_t filebno, uint32_t **ppdiskbno, bool alloc)
{
    // LAB 5: Your code here.
    if (filebno >= NDIRECT + NINDIRECT) return -E_INVAL;
    if (filebno < NDIRECT)
        *ppdiskbno = &(f->f_direct[filebno]);
    else {
        if (!f->f_indirect) {
            if (!alloc)
                return -E_NOT_FOUND;
            else {
                int r = alloc_block();
                if (r < 0) return -E_NO_DISK;
                memset(diskaddr(r), 0, BLKSIZE);
                flush_block(diskaddr(r));
                f->f_indirect = r;
            }
        }
        uint32_t *b = diskaddr(f->f_indirect);
        *ppdiskbno = &(b[filebno - NDIRECT]);
    }
    return 0;
}
```

file_get_block:



```c
int
file_get_block(struct File *f, uint32_t filebno, char **blk)
{
    // LAB 5: Your code here.
    if (filebno >= NDIRECT + NINDIRECT) return -E_INVAL;
    uint32_t *ppdiskbno;
    file_block_walk(f, filebno, &ppdiskbno, 1);
    if (!(*ppdiskbno)) {
        int r = alloc_block();
        if (r < 0) return -E_NO_DISK;
        memset(diskaddr(r), 0, BLKSIZE);
        flush_block(diskaddr(r));
        *ppdiskbno = r;
    }
    *blk = diskaddr(*ppdiskbno);
    return 0;
}
```

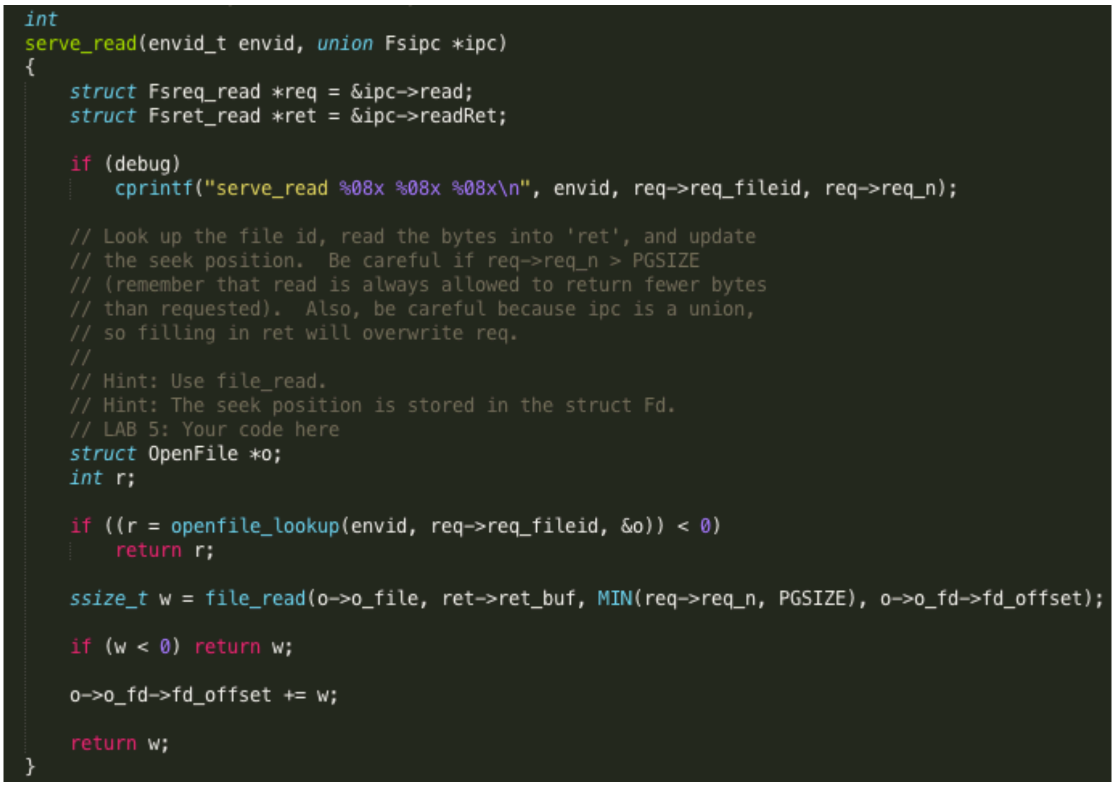
Exercise 5 & 6

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

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

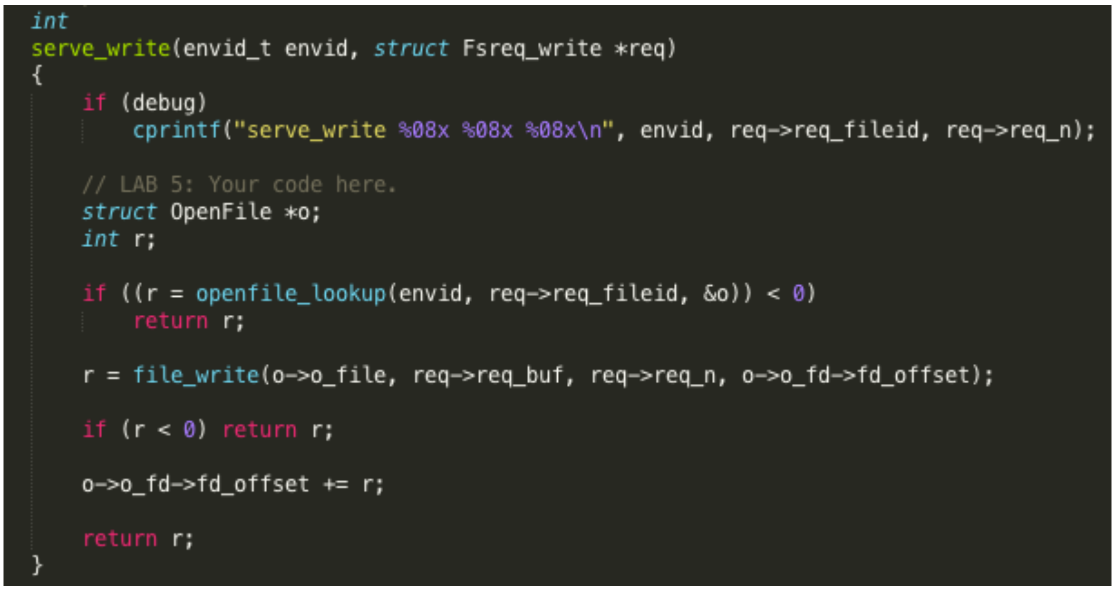
这几个函数实现比较简单，按照注释调用相关接口即可。

serve_read:



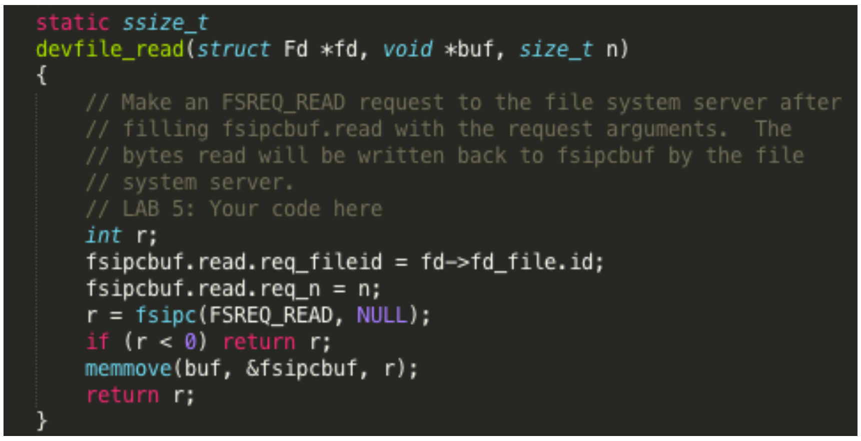
```c
int
serve_read(envid_t envid, union Fsipc *ipc)
{
	struct Fsreq_read *req = &ipc->read;
	struct Fsret_read *ret = &ipc->readRet;

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

	// Look up the file id, read the bytes into 'ret', and update
	// the seek position.  Be careful if req->req_n > PGSIZE
	// (remember that read is always allowed to return fewer bytes
	// than requested).  Also, be careful because ipc is a union,
	// so filling in ret will overwrite req.
	//
	// Hint: Use file_read.
	// Hint: The seek position is stored in the struct Fd.
	// LAB 5: Your code here
	struct OpenFile *o;
	int r;

	if ((r = openfile_lookup(envid, req->req_fileid, &o)) < 0)
		return r;

	ssize_t w = file_read(o->o_file, ret->ret_buf, MIN(req->req_n, PGSIZE), o->o_fd->fd_offset);

	if (w < 0) return w;

	o->o_fd->fd_offset += w;

	return w;
}
```

serve_write:



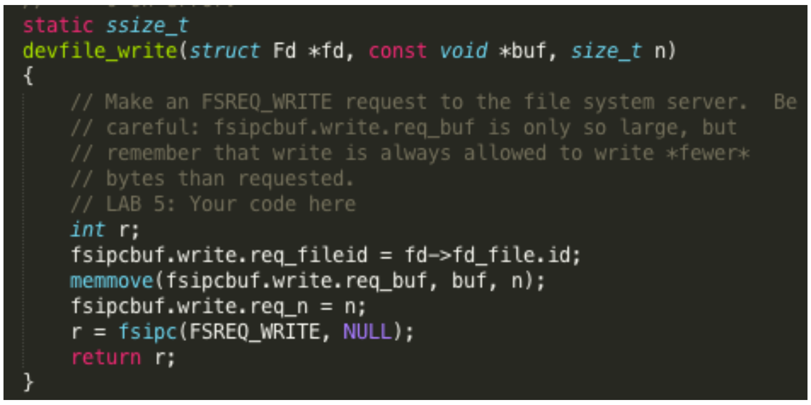
```c
int
serve_write(envid_t envid, struct Fsreq_write *req)
{
	if (debug)
		cprintf("serve_write %08x %08x %08x\n", envid, req->req_fileid, req->req_n);

	// LAB 5: Your code here.
	struct OpenFile *o;
	int r;

	if ((r = openfile_lookup(envid, req->req_fileid, &o)) < 0)
		return r;

	r = file_write(o->o_file, req->req_buf, req->req_n, o->o_fd->fd_offset);

	if (r < 0) return r;

	o->o_fd->fd_offset += r;

	return r;
}
```

devfile_read:



```
static ssize_t
devfile_read(struct Fd *fd, void *buf, size_t n)
{
    // Make an FSREQ_READ request to the file system server after
    // filling fsipcbuf.read with the request arguments.  The
    // bytes read will be written back to fsipcbuf by the file
    // system server.
    // LAB 5: Your code here
    int r;
    fsipcbuf.read.req_fileid = fd->fd_file.id;
    fsipcbuf.read.req_n = n;
    r = fsipc(FSREQ_READ, NULL);
    if (r < 0) return r;
    memmove(buf, &fsipcbuf, r);
    return r;
}
```

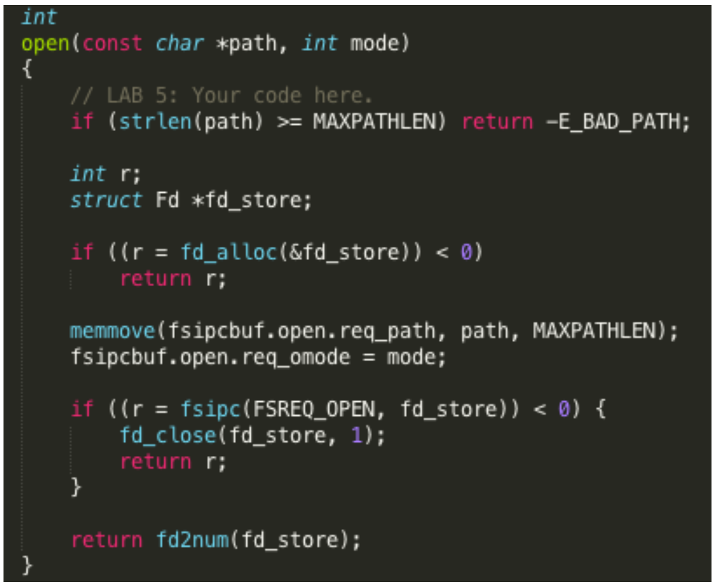
devfile_write:



```
static ssize_t
devfile_write(struct Fd *fd, const void *buf, size_t n)
{
    // Make an FSREQ_WRITE request to the file system server.  Be
    // careful: fsipcbuf.write.req_buf is only so large, but
    // remember that write is always allowed to write *fewer*
    // bytes than requested.
    // LAB 5: Your code here
    int r;
    fsipcbuf.write.req_fileid = fd->fd_file.id;
    memmove(fsipcbuf.write.req_buf, buf, n);
    fsipcbuf.write.req_n = n;
    r = fsipc(FSREQ_WRITE, NULL);
    return r;
}
```

Exercise 7. Implement open.

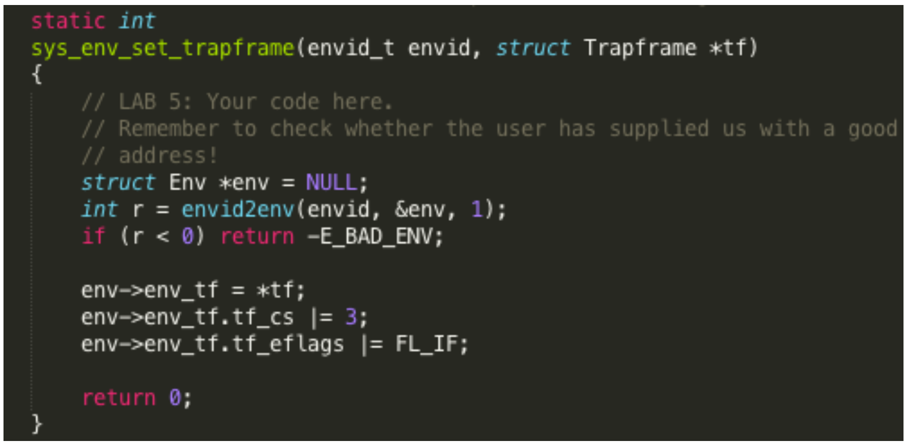
实现 open 函数主要是弄清楚 fd 的数据结构，将里面的属性填好之后，发一个 fsipc 请求就可以了。



```
int
open(const char *path, int mode)
{
    // LAB 5: Your code here.
    if (strlen(path) >= MAXPATHLEN) return -E_BAD_PATH;

    int r;
    struct Fd *fd_store;

    if ((r = fd_alloc(&fd_store)) < 0)
        return r;

    memmove(fsipcbuf.open.req_path, path, MAXPATHLEN);
    fsipcbuf.open.req_omode = mode;

    if ((r = fsipc(FSREQ_OPEN, fd_store)) < 0) {
        fd_close(fd_store, 1);
        return r;
    }

    return fd2num(fd_store);
}
```

Exercise 8. Implement sys_env_set_trapframe.

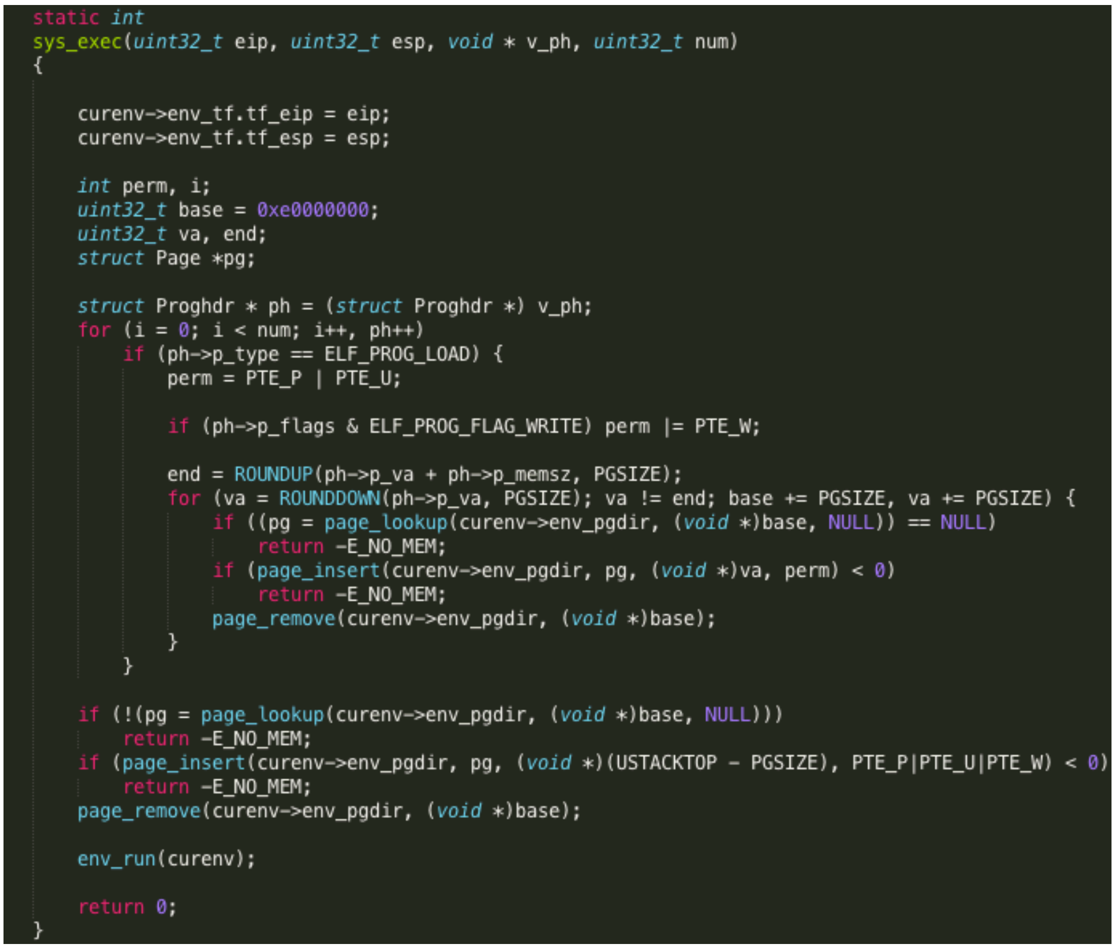
sys_env_set_trapframe 的实现也仍然比较简单，将 protection level 设置为 3，并置上 FL_IF，同时在 syscall 函数中的 switch 语句中添加 SYS_ env_set_trapframe 这一项即可。



```c
static int
sys_env_set_trapframe(envid_t envid, struct Trapframe *tf)
{
	// LAB 5: Your code here.
	// Remember to check whether the user has supplied us with a good
	// address!
	struct Env *env = NULL;
	int r = envid2env(envid, &env, 1);
	if (r < 0) return -E_BAD_ENV;

	env->env_tf = *tf;
	env->env_tf.tf_cs  |= 3;
	env->env_tf.tf_eflags  |= FL_IF;

	return 0;
}
```

Challenge! Implement Unix-style exec.

相比 exercise 而言，这个 lab 中的 challenge 难度都要高出不少，最后我选择了实现 exec 的 challenge。要实现 exec，主要的问题在于不能替换当前正在使用的内存，所以我的做法是在栈上另开一块临时空间，用来载入需要执行的程序，剩下的逻辑则可参考 spawn。



```c
static int
sys_exec(uint32_t eip, uint32_t esp, void * v_ph, uint32_t num)
{

	curenv->env_tf.tf_eip = eip;
	curenv->env_tf.tf_esp = esp;

	int perm, i;
	uint32_t base = 0xe0000000;
	uint32_t va, end;
	struct Page *pg;

	struct Proghdr * ph = (struct Proghdr *) v_ph;
	for (i = 0; i < num; i++, ph++)
		if (ph->p_type == ELF_PROG_LOAD) {
			perm = PTE_P | PTE_U;

			if (ph->p_flags & ELF_PROG_FLAG_WRITE) perm |= PTE_W;

			end = ROUNDUP(ph->p_va + ph->p_memsz, PGSIZE);
			for (va = ROUNDDOWN(ph->p_va, PGSIZE); va != end; base += PGSIZE, va += PGSIZE) {
				if ((pg = page_lookup(curenv->env_pgdir, (void *)base, NULL)) == NULL)
					return -E_NO_MEM;
				if (page_insert(curenv->env_pgdir, pg, (void *)va, perm) < 0)
					return -E_NO_MEM;
				page_remove(curenv->env_pgdir, (void *)base);
			}
		}

	if (!(pg = page_lookup(curenv->env_pgdir, (void *)base, NULL)))
		return -E_NO_MEM;
	if (page_insert(curenv->env_pgdir, pg, (void *)(USTACKTOP - PGSIZE), PTE_P|PTE_U|PTE_W) < 0)
		return -E_NO_MEM;
	page_remove(curenv->env_pgdir, (void *)base);

	env_run(curenv);

	return 0;
}
```