# Jos Lab 5: File System

**Li Xinyu 515030910292**

# The File System

## Disk Access

> **Exercise 1**
> 修改 `env_create()` 的代码，如果 `type == ENV_TYPE_FS` 则给予I/O权限，即 `env->env_tf.tf_eflags |= FL_IOPL_MASK`

## The Block Cache

> **Exercise 2**
> 修改函数 `bc_pgfault()`，首先使用 `ROUNDDOWN` 将 `addr` 对齐到 `PGSIZE` 赋值给 `va`，然后调用 `sys_page_alloc(0, va, PTE_W | PTE_U | PTE_P)` 分配一个页，最后调用 `ide_read(blockno * BLKSECTS, va, BLKSECTS)` 将对应block中的内容读取到 `va`。
> 修改函数 `flush_block()`，首先使用 `ROUNDDOWN` 将 `addr` 对齐到 `PGSIZE` 赋值给 `va`，如果 `!va_is_mapped(va) || !va_is_dirty(va)` 即该地址未被映射或未被写过则直接返回，然后调用 `ide_write(blockno * BLKSECTS, va, BLKSECTS)` 将 `va` 处的页写到对应block中，最后调用 `sys_page_map(0, va, 0, va, PTE_SYSCALL)` 重新映射该页，清除其中的 `PTE_D` 位。

### The Block Bitmap

> **Exercise 3**
> 使用循环遍历[0， `super->s_nblocks`)，调用 `block_is_free(i)` 如果为假则继续遍历，为真则分配这个block，使用 `bitmap[i/32] &= ~(1<<(i%32));` 在 `bitmap` 中将这个block对应的bit设置为1，然后调用 `flush_block(&bitmap[i/32])` 将修改的 `bitmap` 写回到磁盘。

## File Operations

> **Exercise 4**
> 修改 `file_block_walk()`，如果 `filebno` 大于 `NDIRECT + NINDIRECT` 则返回 `-E_INVAL`；若小于 `NDIRECT`，则直接将 `*ppdiskbno` 赋值为 `f->f_direct + filebno`；否则说明是在Indirect Block中，若 `f_indirect` 为空，如 `alloc` 为1则分配一个Indirect

Block初始化为0并调用 `flush_block` 写回磁盘， 否则返回 `-E_NOT_FOUND` 。然后将 `*ppdiskbno` 赋值为Indirect Block中的第 `filebno-NDIRECT` 个块地址。
修改 `file_get_block()` ，调用 `file_block_walk()` ，返回值小于0则返回错误值，如果 `*pdiskbno` 为0则为其分配一个block，最后将 `*blk` 赋值为 `diskaddr(*pdiskbno)` 。

## Client/Server File System Access

**Exercise 5**
修改 `serve_read` ，调用 `openfile_lookup(envid, req->req_fileid, &of)` 得到struct OpenFile的指针 `of` ，将 `req->req_n` 限制在PGSIZE以内得到 `count` ，利用 `of` 调用 `file_read(of->o_file, ret->ret_buf, count, of->o_fd->fd_offset)` 读取文件，最后将 `of->o_fd->fd_offset` 加上读取到的字节数。
修改 `devfile_read` ， 将 `fsipcbuf.read.req_fileid` 赋值为 `fd->fd_file.id` ， `fsipcbuf.read.req_n` 赋值为 `n` 。然后调用 `fsipc(FSREQ_READ, NULL)` 通过IPC发送文件读取请求，最后调用 `memmove` 将读到的内容写到参数 `buf` 中。

-

**Exercise 6**
修改 `serve_write` ，调用 `openfile_lookup(envid, req->req_fileid, &of)` 得到struct OpenFile的指针 `of` ，调用 `file_write(of->o_file, req->req_buf, req->req_n, of->o_fd->fd_offset)` 将内容写到文件中，最后同样将 `of->o_fd->fd_offset` 加上写到文件中的字节数。
修改 `devfile_write` ，将 `fsipcbuf.write.req_fileid` 赋值为 `fd->fd_file.id` ，将 `n` 限制在 `sizeof(fsipcbuf.write.req_buf)` 以内得到 `count` 赋值给 `fsipcbuf.write.req_n` ，调用 `memmove` 将参数 `buf` 中的内容写到 `fsipcbuf.write.req_buf` 中，然后调用 `fsipc(FSREQ_WRITE, NULL)` 发起写文件的请求。

## Client-Side File Operations

**Exercise 7**
修改 `open()` ，调用 `fd_alloc()` 得到一个struct Fd的指针，调用 `strcpy` 将 `path` 复制到 `fsipcbuf.open.req_path` ，将 `fsipcbuf.open.req_omode` 赋值为参数 `mode` ，然后调用 `fsipc(FSREQ_OPEN, fd)` 发送IPC请求打开 `fd` 对应的文件，若返回值小于0则调用 `fd_close(fd, 0)` 关闭 `fd` 否则返回 `fd` 在 `FDTABLE` 中的索引 `fd2num(fd)` 。

## Spawning Processes

**Exercise 8**
修改 `sys_env_set_trapframe()` ，调用 `envid2env(envid, &e, 1)` 得到struct Env的指针 `e` ，将 `e->env_tf` 设置为 `*tf` ，参考 `env_alloc()` 中的代码，将DS/ES/SS设置

为 `GD_UD|3`，将CS设置为 `GD_UT|3`，将 `tf_eflags` 的 `FL_IF` 开启。然后在 `syscall()` 的switch语句中增加一条 `case SYS_env_set_trapframe: ret = sys_env_set_trapframe(a1, (struct Trapframe*)a2);`。

# Challenge

**Implement Unix-style exec:**
参考xv6中exec函数的实现：创建新的地址空间并加载ELF文件，然后将当前的地址空间切换到新的地址空间，释放旧的地址空间。
在Jos中由于涉及到地址空间的操作都需要传入struct Env的指针，因此需要创建一个新的Environment来完成地址空间的创建、初始化及ELF加载。
参考spawn的代码实现，新建函数execl和exec，execl用于将传入的参数字符串转换成参数数组，然后调用exec进行处理。execl的代码与spawnl一致，exec的代码与spawn有些许区别。exec调用 `sys_env_run` 代替spawn中的 `sys_env_set_trapframe` 和 `sys_env_set_status`。
在 `inc/syscall.h` 的enum中添加 `SYS_env_run`。
在 `lib/syscall.c` 中添加函数 `int sys_env_run(envid_t envid, struct Trapframe *tf)`，调用 `syscall(SYS_env_run, 1, envid, (uint32_t) tf, 0, 0, 0);`。
在 `kern/syscall.c` 中添加函数 `static int sys_env_run(envid_t envid, struct Trapframe *tf)`，首先调用 `envid2env(envid, &e, 1)` 得到struct Env的指针 `e`，将 `curenv->env_tf` 赋值为 `*tf`，交换 `e->env_pgdir` 和 `curenv->env_pgdir` 进行地址空间的切换，然后调用 `env_free(e)` 释放旧的地址空间。然后调用 `lcr3(PADDR(curenv->env_pgdir))` 加载新的页表，最后调用 `unlock_kernel();env_pop_tf(&curenv->env_tf);` 运行程序。
在 `syscall()` 的switch语句中增加一条 `case SYS_env_run: ret = sys_env_run(a1, (struct Trapframe*)a2);`。
修改 `user/icode.c` 的函数 `umain()` 如下：

```
#ifdef CHALLENGELAB5
  cprintf("icode: exec /init\n");
  if ((r = execl("/init", "init", "initarg1", "initarg2", (char*)0)) < 0)
    panic("icode: exec /init: %e", r);
#else
  cprintf("icode: spawn /init\n");
  if ((r = spawnl("/init", "init", "initarg1", "initarg2", (char*)0)) < 0)
    panic("icode: spawn /init: %e", r);
#endif
```

在 `inc/lib.h` 中开启宏定义 `#define CHALLENGELAB5`。