Lab #5

陈威宇

Exercise 1

给 file server IO 特权.

```
// 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

给 addr 所在的虚拟地址分配一个页，并从 disk 上把对应的 block 的内容读过来.

```
addr = ROUNDDOWN(addr, PGSIZE);
if ((r = sys_page_alloc(thisenv->env_id, addr, PTE_W | PTE_U | PTE_P)) != 0) {
    panic("bc_pgfault: %e", r);
}
if ((r = ide_read(blockno * BLKSECTS, addr, BLKSECTS)) != 0) {
    panic("bc_pgfault: %e", r);
}
```

把 addr 所在的虚拟页对应的 block 的内容 flush 出去到 disk 上.

```
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)) || (!va_is_dirty(addr)))
    return;
  r = ide_write(blockno * BLKSECTS, addr, BLKSECTS);
  if (r!=0)
    panic("flush_block: %e", r);
  sys_page_map(thisenv->env_id, addr, thisenv->env_id, addr, uvpt[PGNUM(addr)] & PTE_SYSCALL);
}
```

Exercise 3

查询 bitmap 来找到一个空闲的 block 来分配.

```
int
alloc_block(void)
{
  for (int blockno = 0; blockno < (super->s_nblocks); ++blockno)
    if (block_is_free(blockno)){
      bitmap[blockno/32] ^= 1<<(blockno%32);
      flush_block(&bitmap[blockno/32]);
      return blockno;
    }
  return -E_NO_DISK;
}
```

Exercise 4

找到 file f 的第 filebno 个 block 的 block 编号.

```
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]);
    return 0;
  }
  if (!(f->f_indirect) && !alloc)
    return -E_NOT_FOUND;
  if (!(f->f_indirect)){
    int blockno = alloc_block();
    if (blockno < 0)
      return blockno;
    f->f_indirect = blockno;
    memset(diskaddr(blockno), 0, BLKSIZE);
  }
  (*ppdiskbno) = &((uint32_t *)diskaddr(f->f_indirect))[filebno - NDIRECT];
  return 0;
}
```

找到 file f 的第 filebno 个 block 的在内存中被映射的 (虚拟) 地址.

```
int
file_get_block(struct File *f, uint32_t filebno, char **blk)
{
  // LAB 5: Your code here.
  uint32_t * t;
  int r = file_block_walk(f, filebno, &t ,1);
  if (r<0)
    return r;
  int blockno;
  if (!(*t)){
    blockno = alloc_block();
    if (blockno<0)
            return blockno;
        *t = blockno;
        memset(diskaddr(blockno), 0, BLKSIZE);
  }
  (*blk) = diskaddr(*t);
  return 0;
}
```

Exercise 5

进行一个读的服务，具体来讲，找到对应的 openfile，然后让 `file_read` 去完成.

```
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);

  // Lab 5: Your code here:
  struct OpenFile * o;
```

```c
  int r;
  r = openfile_lookup(envid, req->req_fileid, &o);
  if (r<0)
    return r;
  r = file_read(o->o_file, ret->ret_buf, req->req_n, o->o_fd->fd_offset);
  if (r<0)
    return r;
  o->o_fd->fd_offset += r;
  return r;
}
```

Exercise 6

进行一个写的服务，具体来讲，找到对应的 openfile，然后让 `file_write` 去完成.

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

写，具体来讲，构造一个 fsipcbuf，然后让 `fsipc` 去完成.

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

  fsipcbuf.write.req_fileid = fd->fd_file.id;
  fsipcbuf.write.req_n = n;
  memmove(fsipcbuf.write.req_buf, buf, n);
  if ((r = fsipc(FSREQ_WRITE, NULL)) < 0)
    return r;
  assert(r <= n);
  assert(r <= PGSIZE);
  return r;
}

```

Exercise 7

这个 syscall 把一个 env 的 trap frame 设置为 tf。

```c
static int
sys_env_set_trapframe(envid_t envid, struct Trapframe *tf)
{
  struct Env * env;
  int r;
  if ((r=envid2env(envid, &env, 1))!=0)
    return r;
  user_mem_assert(env, tf, sizeof(struct Trapframe), PTE_W);
  tf->tf_cs |= 3;
  tf->tf_ss |= 3;
  tf->tf_eflags |= FL_IF;
  tf->tf_eflags ^= (tf->tf_eflags & FL_IOPL_MASK);
  env->env_tf = (*tf);
  return 0;
}
```

Exercise 8

duppage 把当前 env 的第 pn 个虚拟页 map 到另一个 env 的相同虚拟位置. 如果当前页的 PTE 含有 PTE_SHARE，那么就直接 map 来共享. 如果当前页是 copy on write 或 writable，那么将页映射都设为 copy on write.

```c
static int
duppage(envid_t envid, unsigned pn)
{
  int r;

  // LAB 4: Your code here.
  void * va = (void *)(pn * PGSIZE);
  envid_t id = sys_getenvid();
  if (uvpt[pn]&PTE_SHARE){
    r = sys_page_map(id, va, envid, va, PTE_SYSCALL & uvpt[pn]);
    if (r<0)
      return r;
  }
  else if ((uvpt[pn] & PTE_W) == PTE_W || (uvpt[pn] & PTE_COW) == PTE_COW){
    r = sys_page_map(id, va, envid, va, PTE_COW | PTE_U | PTE_P);
    if (r<0)
      return r;
    r = sys_page_map(id, va, id, va, PTE_COW | PTE_U | PTE_P);
    if (r<0)
      return r;
  }
  else{
    r = sys_page_map(id, va, envid, va, PTE_U | PTE_P);
    if (r<0)
      return r;
  }

  return 0;
}

```

把 parent_env 的所有 shared page 的映射复制给 child.

```
static int
copy_shared_pages(envid_t child)
{
  // LAB 5: Your code here.
  envid_t parent_envid = sys_getenvid();
    uint32_t addr;
    int r;

    for (addr = 0; addr < USTACKTOP; addr += PGSIZE) {
        if ((uvpd[PDX(addr)] & PTE_P) == PTE_P && (uvpt[PGNUM(addr)] & PTE_P) == PTE_P && (uvp
            if ((r = sys_page_map(parent_envid, (void *)addr, child, (void *)addr, uvpt[PGNUM(
                panic("copy_shared_pages: %e", r);
                }
            }
        }
    return 0;
}
```

Exercise 9

在 `trap_dispatch` 中增加对这两个 interrupt 的处理.

```
    if (tf->tf_trapno == IRQ_OFFSET+IRQ_KBD){
      kbd_intr();
      return;
    }
    if (tf->tf_trapno == IRQ_OFFSET+IRQ_SERIAL){
      serial_intr();
      return;
    }
```

Exercise 10

输入重定向, 打开后用下 dup 就行.

```
        if ((fd = open(t, O_RDONLY)) < 0) {
          cprintf("open %s for read: %e", t, fd);
          exit();
        }
        if (fd != 0) {
          dup(fd, 0);
          close(fd);
        }
```