# Lab #2

陈威宇

## Exercise 1

`boot_alloc` 分配 `NENV*sizeof(struct Env)` 字节物理内存给 `envs`. `boot_map_region` 将 `envs` 数组 map 到 `UENVS`. 代码如下.

```
  envs=(struct Env *)boot_alloc(NENV*sizeof(struct Env));
  memset(envs,0,NENV*sizeof(struct Env));

  boot_map_region(kern_pgdir, UENVS, PTSIZE, PADDR(envs), PTE_U | PTE_P);
```

## Exercise 2

`env_init` 用来初始化 `envs` 数组，并将 free 的全部环境都放到链表 `env_free_list`. 代码如下.

```
void
env_init(void)
{
  for (int i=0;i<NENV;++i)
    envs[i].env_id = 0, envs[i].env_status = ENV_FREE;
  env_free_list = NULL;
  for (int i=NENV-1;i>=0;--i){
    envs[i].env_link = env_free_list;
    env_free_list = &(envs[i]);
  }
  env_init_percpu();
}
```

`env_setup_vm` 用来给一个环境 `e` 初始化虚拟内存. 用 `page_alloc` 分配一个物理页给 `e` 当 page directory, 并将其内容初始化. 因为 UTOP 以上的映射都是一样的，所以直接从 *kern_pgdir* 复制过来就行，除了 UVPT 映射到环境 `e` 自己的 page directory.

```
static int
env_setup_vm(struct Env *e)
{
  int i;
  struct PageInfo *p = NULL;
  if (!(p = page_alloc(ALLOC_ZERO)))
    return -E_NO_MEM;
  (e->env_pgdir) = (pde_t *)page2kva(p);
```

```
  memcpy((void *)(e->env_pgdir), (void *)kern_pgdir, PGSIZE);
  ++(p->pp_ref);
  e->env_pgdir[PDX(UVPT)] = PADDR(e->env_pgdir) | PTE_P | PTE_U;
  return 0;
}
```

region_alloc() 的作用是给环境 e 的分配 len 字节的物理内存, 并映射到虚拟地址 va 开始的地方. 调用 lab2 时候写的 page_insert 函数即可.

```
static void
region_alloc(struct Env *e, void *va, size_t len)
{
  len = ROUNDUP((uint32_t)(va+len), PGSIZE);
  va = ROUNDDOWN(va, PGSIZE);
  len -= (uint32_t) va;
  for (int i=0;i<len/PGSIZE;++i){
    void * x = va + i*PGSIZE;
    struct PageInfo * pp = page_alloc(0);
    if (pp == NULL) panic("out of free memory!!!");
    int o = page_insert(e->env_pgdir, pp, x, PTE_P | PTE_W | PTE_U);
    if (o<0) panic("page_insert no mem!!!");
  }
}
```

load_icode 的作用是把从 binary 开始的 ELF 格式的内容读取并将要 load 的内容按照指定的位置 load 到内存.

```
static void
load_icode(struct Env *e, uint8_t *binary)
{
  lcr3(PADDR(e->env_pgdir));

  struct Proghdr *ph, *eph;
  struct Elf * elfhdr = (struct Elf *)binary;
  if (elfhdr->e_magic != ELF_MAGIC)
    panic("ELF_MAGIC wrong!!!");
  // load each program segment (ignores ph flags)
  ph = (struct Proghdr *) ((uint8_t *) elfhdr + (elfhdr->e_phoff));
  eph = ph + elfhdr->e_phnum;
  for (; ph < eph; ph++)
    if (ph->p_type == ELF_PROG_LOAD){
      // p_pa is the load address of this segment (as well
```

```
16        // as the physical address)
17        if (!(ph->p_filesz <= ph->p_memsz)) panic("ph->p_filesz <= ph->p_memsz wrong!!!");
18        region_alloc(e, (void *)(ph->p_va), ph->p_memsz);
19        memcpy((void *)(ph->p_va), (ph->p_offset) + binary, ph->p_filesz);
20        memset((void *)(ph->p_va) + ph->p_filesz, 0, ph->p_memsz - ph->p_filesz);
21      }
22    e -> env_tf.tf_eip = elfhdr -> e_entry;
23    region_alloc(e, (void *)(USTACKTOP - PGSIZE), PGSIZE);
24
25    lcr3(PADDR(kern_pgdir));
26  }
```

env_create 的用处是用 env_alloc 分配一个环境 e，然后调用上面的 load_icode 函数从 binary 开始的 ELF 格式二进制 load 给环境 e.

```
1  void
2  env_create(uint8_t *binary, enum EnvType type)
3  {
4    // LAB 3: Your code here.
5    struct Env * e;
6    env_alloc(&e, 0);
7    load_icode(e,binary);
8    (e->env_type) = ENV_TYPE_USER;
9  }
```

env_run 的用处是进行一个上下文切换，然后开始运行环境 e.

```
1  void
2  env_run(struct Env *e)
3  {
4    if (curenv!=NULL){
5      if ((curenv->env_status)==ENV_RUNNING){
6        (curenv->env_status) = ENV_RUNNABLE;
7      }
8    }
9    curenv = e;
10   (curenv -> env_status) = ENV_RUNNING;
11   ++(curenv -> env_runs);
12   lcr3(PADDR(curenv->env_pgdir));
13   env_pop_tf(&(curenv->env_tf));
14 }
```

## Exercise 4

在 `trapentry.S` 里生成那些 trap 的 entry points.

```
TRAPHANDLER_NOEC(DIVIDE, T_DIVIDE)
TRAPHANDLER_NOEC(DEBUG, T_DEBUG)
TRAPHANDLER_NOEC(NMI, T_NMI)
TRAPHANDLER_NOEC(BRKPT, T_BRKPT)
TRAPHANDLER_NOEC(OFLOW, T_OFLOW)
TRAPHANDLER_NOEC(BOUND, T_BOUND)
TRAPHANDLER_NOEC(ILLOP, T_ILLOP)
TRAPHANDLER_NOEC(DEVICE, T_DEVICE)
TRAPHANDLER(DBLFLT, T_DBLFLT)
TRAPHANDLER(TSS, T_TSS)
TRAPHANDLER(SEGNP, T_SEGNP)
TRAPHANDLER(STACK, T_STACK)
TRAPHANDLER(GPFLT, T_GPFLT)
TRAPHANDLER(PGFLT, T_PGFLT)
TRAPHANDLER_NOEC(FPERR, T_FPERR)
TRAPHANDLER(ALIGN, T_ALIGN)
TRAPHANDLER_NOEC(MCHK, T_MCHK)
TRAPHANDLER_NOEC(SIMDERR, T_SIMDERR)
TRAPHANDLER_NOEC(SYSCALL, T_SYSCALL)
```

`trap_init` 的用处是初始化 interrupt descriptor table.

```
void
trap_init(void)
{
  extern struct Segdesc gdt[];

  // LAB 3: Your code here.
  void DIVIDE();
  SETGATE(idt[0], 0, GD_KT, DIVIDE, 0);
  void DEBUG();
  SETGATE(idt[1], 0, GD_KT, DEBUG, 0);
  void NMI();
  SETGATE(idt[2], 0, GD_KT, NMI, 0);
  void BRKPT();
  SETGATE(idt[3], 1, GD_KT, BRKPT, 3);
  void OFLOW();
  SETGATE(idt[4], 1, GD_KT, OFLOW, 0);
```

```
17    void BOUND();
18    SETGATE(idt[5], 0, GD_KT, BOUND, 0);
19    void ILLOP();
20    SETGATE(idt[6], 0, GD_KT, ILLOP, 0);
21    void DEVICE();
22    SETGATE(idt[7], 0, GD_KT, DEVICE, 0);
23    void DBLFLT();
24    SETGATE(idt[8], 0, GD_KT, DBLFLT, 0);
25    void TSS();
26    SETGATE(idt[10], 0, GD_KT, TSS, 0);
27    void SEGNP();
28    SETGATE(idt[11], 0, GD_KT, SEGNP, 0);
29    void STACK();
30    SETGATE(idt[12], 0, GD_KT, STACK, 0);
31    void GPFLT();
32    SETGATE(idt[13], 0, GD_KT, GPFLT, 0);
33    void PGFLT();
34    SETGATE(idt[14], 0, GD_KT, PGFLT, 0);
35    void FPERR();
36    SETGATE(idt[16], 0, GD_KT, FPERR, 0);
37    void ALIGN();
38    SETGATE(idt[17], 0, GD_KT, ALIGN, 0);
39    void MCHK();
40    SETGATE(idt[18], 0, GD_KT, MCHK, 0);
41    void SIMDERR();
42    SETGATE(idt[19], 0, GD_KT, SIMDERR, 0);
43    void SYSCALL();
44    SETGATE(idt[48], 1, GD_KT, SYSCALL, 3);
45
46    // Per-CPU setup
47    trap_init_percpu();
48  }
```

_alltraps 的用处是,trap_handler 在进入 trap 函数之前,在栈上做出一个完整的 struct Trapframe 来给 trap 及后续的函数使用. 另外, 将%ds 和%es 设置成 GD_KD, 即 kernel data 段.

```
1  _alltraps:
2    pushl %ds
3    pushl %es
4    pushal
5    pushl $(GD_KD)
```

```
  popl %ds
  pushl $(GD_KD)
  popl %es
  pushl %esp
  call trap
```

## Questions after Exercise 4

### 1

这样使得我们，对每种不同的 trap，我们可以给他设置权限，有些是用户有权限的.

### 2

用户程序没有权限执行这条 `int` 指令, 所以就 General Protection Exception 了. 这样起到保护作用.

## Exercise 5 与 Exercise 6

`trap_dispatch` 的作用是对一个 `struct Trapframe` 看其具体的 `trapno` 来决定将这个 trap 交给哪个函数处理.

```
static void
trap_dispatch(struct Trapframe *tf)
{
  switch (tf->tf_trapno){
  case T_PGFLT:
    page_fault_handler(tf);
    return;
  case T_BRKPT:
    monitor(tf);
    return;
  case T_SYSCALL:
      tf->tf_regs.reg_eax = syscall(tf->tf_regs.reg_eax, tf->tf_regs.reg_edx,
      tf->tf_regs.reg_ecx, tf->tf_regs.reg_ebx, tf->tf_regs.reg_edi, tf->tf_regs.reg_esi);
    if (tf->tf_regs.reg_eax < 0)
      panic("syscall invalid");
    return;
  default:
    break;
  }
  // Unexpected trap: The user process or the kernel has a bug.
  print_trapframe(tf);
```

```
  if (tf->tf_cs == GD_KT)
    panic("unhandled trap in kernel");
  else {
    env_destroy(curenv);
    return;
  }
}
```

## Questions after Exercise 6

### 3

这个取决于我 SETGATE 时候的 dpl 的设置，dpl=3 就是用户有权限的，dpl=0 就是没的.

### 4

这些机制很好地起到保护作用.

## Exercise 7

kern 文件夹下的 syscall.c 中的 `syscall` 的用处是，根据 syscall 的 syscallno 和参数来调用对应的函数来处理 syscall.

```
int32_t
syscall(uint32_t syscallno, uint32_t a1, uint32_t a2, uint32_t a3, uint32_t a4, uint32_t a5)
{
  int t;
  switch (syscallno) {
  case ((int)SYS_cputs):
    sys_cputs((char *)a1, a2);
    return 0;
  case ((int)SYS_cgetc):
    return sys_cgetc();
  case ((int)SYS_env_destroy):
    return sys_env_destroy(a1);
  case ((int)SYS_getenvid):
    return sys_getenvid();
  default:
    return -E_INVAL;
  }
}
```

## Exercise 8

将 thisenv 指向当前的环境.

```
thisenv = (&envs[ENVX(sys_getenvid())]);
```

## Exercise 9 与 Exercise 10

user_mem_check 的用处是判断环境 env 是否允许访问 va 开始的 len 字节. 做法是对每个页都 pgdir_walk 看下 PTE 上的权限.

```
int
user_mem_check(struct Env *env, const void *va, size_t len, int perm)
{
  // LAB 3: Your code here.
  len = ROUNDUP((uint32_t)(va+len), PGSIZE);
  void * vva = (void *)ROUNDDOWN(va, PGSIZE);
  len -= (uint32_t) vva;
  int gg=0;
  uint32_t x;
  for (int i=0;i<len/PGSIZE;++i){
    x = (uint32_t) vva + i*PGSIZE;
    if ((uint32_t) x >= ULIM){
      gg=1;
      break;
    }
    pte_t * t = pgdir_walk(env->env_pgdir, (const void *)x, 0);
    if (t==NULL){
      gg=1;
      break;
    }
    if ( ((*t) & (PTE_P | perm)) != (PTE_P | perm)){
      gg=1;
      break;
    }
  }
  if (gg==1){
    user_mem_check_addr = (uintptr_t)MAX((uint32_t)x, (uint32_t)va);
    return -E_FAULT;
  }
  return 0;
}
```

在 debuginfo_eip 函数里增加确认内存合法的判断.

```
    if (user_mem_check(curenv, (void *)usd, sizeof(struct UserStabData), PTE_U | PTE_P)<0)
      return -1;

    stabs = usd->stabs;
    stab_end = usd->stab_end;
    stabstr = usd->stabstr;
    stabstr_end = usd->stabstr_end;

    if (user_mem_check(curenv, (void *)stabs, (uint32_t)stab_end - (uint32_t)stabs, PTE_U | PT
      return -1;
    if (user_mem_check(curenv, (void *)stabstr, (uint32_t)stabstr_end - (uint32_t)stabstr, PTE
      return -1;
```