Lab3

516030910101 罗宇辰

Part A

Exercise 1

- mem_init()

```
// 先申请`ENVSIZE`大小的物理空间
uint32_t ENVSIZE = NENV * sizeof(struct Env);
envs = (struct Env*)boot_alloc(ENVSIZE);
memset(envs, 0, ENVSIZE);

// 再映射到虚拟地址中的`UENVS`处
boot_map_region(kern_pgdir, UENVS, PTSIZE, PADDR(envs), PTE_U);
```

Exercise 2.

- env_init()

```
void
env_init(void)
{
    env_free_list = NULL;

    for (int i=NENV-1; i>=0; i--){  //为了保证env_free_list中的顺序和env中相同，要逆向遍历
        envs[i].env_id = 0;
        envs[i].env_status = ENV_FREE;
        envs[i].env_link = env_free_list;

        env_free_list = &envs[i];
    }
    // Per-CPU part of the initialization
    env_init_percpu();
}
```

- env_setup_vm()

```
static int
env_setup_vm(struct Env *e)
{
    int i;
    struct PageInfo *p = NULL;

    // Allocate a page for the page directory
    if (!(p = page_alloc(ALLOC_ZERO)))
        return -E_NO_MEM;
    // 设置新分配的物理页为env的页表
    e->env_pgdir = page2kva(p);
    p->pp_ref += 1;
    // 由于初始情况下UTOP以上的虚拟空间是各进程相同的，所以把kern_pgdir中的这部分内容拷贝到env
pgdir中
```

```
    int begin = PDX(UTOP);
    int size = (NPDENTRIES-begin) * sizeof(pde_t);
    memcpy(&e->env_pgdir[begin],&kern_pgdir[begin], size);
    ...
}
```

- region_alloc()

```
static void
region_alloc(struct Env *e, void *va, size_t len)
{
    va = ROUNDDOWN(va, PGSIZE);
    int n = (ROUNDUP(va+len, PGSIZE) - va) / PGSIZE;
    // align之后，依次alloc需要的空间并插入env_pgdir
    for(int i=0;i<n;i++){
        struct PageInfo* pp = page_alloc(ALLOC_ZERO);
        if(!pp)
            panic("region_alloc: pp is NULL!");
        page_insert(e->env_pgdir, pp, va, PTE_U|PTE_W);
        va += PGSIZE;
    }
}
```

- load_icode()

```
static void
load_icode(struct Env *e, uint8_t *binary)
{
    // 对照boot/main中的main.c来load elf file
    struct Proghdr *ph, *eph;
    struct Elf* ELFHDR = (struct Elf*)binary;

    // is this a valid ELF?
    if (ELFHDR->e_magic != ELF_MAGIC)
        panic("load icode:\tinvalid binary file!");

    // load each program segment
    ph = (struct Proghdr *) (binary + ELFHDR->e_phoff);
    eph = ph + ELFHDR->e_phnum;

    for (; ph < eph; ph++){
        // 判断ph的类型是否是ELF_PROG_LOAD
        if(ph->p_type != ELF_PROG_LOAD)
            continue;

        void* va = (void*)(ph->p_va);
        void* pa = (void*)(binary+ph->p_offset);

        // 先把alloc到的虚拟地址清空，然后将程序load到对应的虚拟地址处，这样超过filesz的部分都是0
        region_alloc(e, va, ph->p_memsz);
        memset(va, 0, ph->p_memsz);
        memmove(va, pa, ph->p_filesz);
    }
    // 设置eip为prog的entry
    e->env_tf.tf_eip = ELFHDR->e_entry;

    ...
}
```

- env_create()

```
void
env_create(uint8_t *binary, enum EnvType type)
{
    struct Env *e;

    // Allocates a new env
    int r = env_alloc(&e, 0);
    if(r != 0)
        panic("env_create: %e", r);

    // sets its env_type
    e->env_type = type;

    // 由于elf file是env空间下的，所以要进行地址空间的切换
    lcr3(PADDR(e->env_pgdir));
    load_icode(e, binary);
    lcr3(PADDR(kern_pgdir));
}
```

- env_run()

```
void
env_run(struct Env *e)
{
    // Step 1:如果存在env switch就要更改当前env的状态
    if(e != curenv) {
        if(curenv && curenv->env_status == ENV_RUNNING)
            curenv->env_status = ENV_RUNNABLE;

        curenv = e;
        e->env_status = ENV_RUNNING;
        e->env_runs += 1;
        lcr3(PADDR(e->env_pgdir));
    }

    // Step 2:运行
    env_pop_tf(&(e->env_tf));
}
```

Exercise 4.

1. trapentry.S

   首先，利用MACRO为每个exception生成entry:

   ```
   TRAPHANDLER_NOEC( ENTRY_DIVIDE   , T_DIVIDE )  /*  0 divide error*/
   TRAPHANDLER_NOEC( ENTRY_DEBUG    , T_DEBUG  )  /*  1 debug exception*/
   TRAPHANDLER_NOEC( ENTRY_NMI      , T_NMI    )  /*  2 non-maskable interrupt*/
   TRAPHANDLER_NOEC( ENTRY_BRKPT    , T_BRKPT  )  /*  3 breakpoint*/
   ...
   ```

   然后，对照ppt和trap_frame的结构，定义_alltraps:

   ```
   .globl _alltraps
   .type _alltraps, @function
   .align 2
   _alltraps:
       # Build trap frame
       pushl %ds    # tf_ds
       pushl %es    # tf_es
       pushal
   ```

```
    # Set up data segments
    movl $GD_KD, %eax
    movw %ax, %ds
    movw %ax, %es

    # Call trap(tf), where tf=%esp
    pushl %esp
    call trap
    addl $4, %esp

    # Return
    popal
    popl %es
    popl %ds
    addl $0x8, %esp # trapno and errcode
    iret
```

2. trap.c

   先声明entry函数

```
extern void ENTRY_DIVIDE ();/*  0 divide error*/
extern void ENTRY_DEBUG  ();/*  1 debug exception*/
extern void ENTRY_NMI    ();/*  2 non-maskable interrupt*/
extern void ENTRY_BRKPT  ();/*  3 breakpoint*/
...
```

然后， 在trap_init中SETGATE

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

    // 注意不同的exception的istrap属性和dpl权限不同
    SETGATE(idt[T_DIVIDE ],1,GD_KT,ENTRY_DIVIDE ,0);
    SETGATE(idt[T_DEBUG  ],1,GD_KT,ENTRY_DEBUG  ,0);
    SETGATE(idt[T_NMI    ],0,GD_KT,ENTRY_NMI    ,0);
    SETGATE(idt[T_BRKPT  ],1,GD_KT,ENTRY_BRKPT  ,3);
    ...

    // Per-CPU setup
    trap_init_percpu();
}
```

Exercise 5&6.

- trap_dispatch

```
    if(tf->tf_trapno == T_PGFLT){ // 特殊处理page_fault
        page_fault_handler(tf);
        return;
    }
    else if(tf->tf_trapno == T_BRKPT){  // 特殊处理break
        monitor(tf);
        return;
    }
```

Exercise 7.

1. trapentry.S

   为SYSCALL生成entry:

   ```
   TRAPHANDLER_NOEC( ENTRY_SYSCALL , T_SYSCALL)  /* 48 system call*/
   ```

2. trap.c

   先声明SYSCALL entry函数:

   ```
   extern void ENTRY_SYSCALL();/* 48 system call*/
   ```

然后，在trap_init中SETGATE:

```
SETGATE(idt[T_SYSCALL],1,GD_KT,ENTRY_SYSCALL,3);
```

在trap_dispatch中特殊处理SYSCALL:

```
// CALLNUM和不同参数存储在对应寄存器中，返回值存在%eax中
else if(tf->tf_trapno == 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);
   return;
}
```

3. kern/syscall.c

   根据syscallno调用不同的处理函数:

```
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_cputs:{
            sys_cputs((char*)a1, (size_t)a2);
            return -SYS_cputs;
        }
        case SYS_cgetc:
            return sys_cgetc();
        case SYS_env_destroy:
            return sys_env_destroy((envid_t)a1);
        case SYS_getenvid:
            return sys_getenvid();
        case SYS_map_kernel_page:
            return sys_map_kernel_page((void*)a1, (void*)a2);
        case SYS_sbrk:
            return sys_sbrk((uint32_t)a1);
        default:
            return -E_INVAL;
    }
}
```

Exercise 8.

1. trapentry.S

   定义sysenter_handler，按顺序把参数push入栈，调用syscall，并将return pc/esp 保存，调用sysexit

   ```
   # syscall handler
   .global sysenter_handler
   .type sysenter_handler, @function
   .align 2
   sysenter_handler:
       pushl %esi
       pushl %edi
       pushl %ebx
       pushl %ecx
       pushl %edx
       pushl %eax
       call syscall
       movl %esi, %edx
       movl %ebp, %ecx
       sysexit
   ```

2. inc/x86.h

   加入MSR定义

   ```
   #define rdmsr(msr,val1,val2) \
       __asm__ __volatile__("rdmsr" \
       : "=a" (val1), "=d" (val2) \
       : "c" (msr))

   #define wrmsr(msr,val1,val2) \
       __asm__ __volatile__("wrmsr" \
       : /* no outputs */ \
       : "c" (msr), "a" (val1), "d" (val2))
   ```

3. trap.c

   声明sysenter_handler:

   ```
   extern void sysenter_handler();
   ```

   在trap_init中设置MSRs来启用sysenter/sysexit:

   ```
       wrmsr(0x174, GD_KT, 0);           /* SYSENTER_CS_MSR */
       wrmsr(0x175, KSTACKTOP, 0);       /* SYSENTER_ESP_MSR */
       wrmsr(0x176, sysenter_handler, 0);/* SYSENTER_EIP_MSR */
   ```

4. lib/syscall.c

   更改汇编，实现对pc和esp的保存，并调用sysenter

   ```
       asm volatile(
           // Store return %esp to %ebp, store return pc to %esi
           "pushl %%esp\n\t"
           "popl %%ebp\n\t"
           "leal after_sysenter_label%=, %%esi\n\t" // Use "%=" to generate a unique label
   number.
           "sysenter\n\t"
           "after_sysenter_label%=:\n\t"
           : "=a" (ret)
           : "a" (num),
   ```

```
              "d" (a1),
              "c" (a2),
              "b" (a3),
              "D" (a4),
              "S" (a5)
            : "cc", "memory");
```

Exercise 9.

使用sys_getenvid来查找当前env:

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

Exercise 10.

首先更改struct Env的结构，加入属性env_brk来记录prog的break:

```
    uintptr_t env_brk;
```

使用region_alloc实现kern/syscall.c中的sys_sbrk:

```
static int
sys_sbrk(uint32_t inc)
{
    region_alloc(curenv, (void*)curenv->env_brk-inc, inc);
    curenv->env_brk = (uintptr_t)ROUNDDOWN(curenv->env_brk-inc, PGSIZE);
    // 注意是返回新的brk，不是旧的
    return curenv->env_brk;
}
```

Exercise 11.

1. trap.c

   在page_fault_handler中检察是否是kernel发生page fault，是的话就panic:

   ```
       if (!(tf->tf_cs & 3)){
           panic("page_fault_handler:\tkernel-mode page faults!");
       }
   ```

2. pmap.c

   检查memory的访问权限:

   ```
   int
   user_mem_check(struct Env *env, const void *va, size_t len, int perm)
   {
       void* va_ptr = (void*)va;
       void* end  = (void*)ROUNDUP((uintptr_t) va + len, PGSIZE);
       perm |= PTE_P;
       // 遍历检测
       for (;va_ptr < end; va_ptr = ROUNDDOWN(va_ptr+PGSIZE, PGSIZE)) {
           // 是否高于ULIM
           if ((uintptr_t) va_ptr >= ULIM) {
               user_mem_check_addr = (uintptr_t) va_ptr;
               return -E_FAULT;
           }
           pte_t * pte = pgdir_walk (env->env_pgdir, va_ptr, 0);
           // page是否存在，user是否有权限
           if ( !pte || !(*pte & perm)) {
   ```

```
                user_mem_check_addr = (uintptr_t) va_ptr;
                return -E_FAULT;
            }
        }
        return 0;
    }
```

3. kern/syscall.c

   检查用到的指针的权限

```
static void
sys_cputs(const char *s, size_t len)
{
    // 检查是否可以访问s处的memory
    user_mem_assert(curenv, (void*)s, len, PTE_U);

    // Print the string supplied by the user.
    cprintf("%.*s", len, s);
}
```

4. kern/kdebug.c

   在debuginfo_eip中对usd, stabs, stabstr进行权限检查:

```
        if(user_mem_check(curenv, (void*)usd, sizeof(struct UserStabData), PTE_U<0))
            return -1;
        ...
        if(user_mem_check(curenv, (void*)stabs, stab_end-stabs, PTE_U)<0)
            return -1;
        if(user_mem_check(curenv, (void*)stabstr, stabstr_end-stabstr, PTE_U)<0)
            return -1;
```

Exercise 12.

```
void ring0_call(void (*fun_ptr)(void)) {
    // 1.使用sgdt将GDT存储到r_get中
    struct Pseudodesc r_gdt;
    sgdt(&r_gdt);

    // 2.将GDT中的page映射到vaddr中
    int t = sys_map_kernel_page((void* )r_gdt.pd_base, (void* )vaddr);
    if (t < 0) {
        cprintf("ring0_call: sys_map_kernel_page failed, %e\n", t);
        return;
    }
    // 找到gdt的位置，并选一个设置callgate的入口
    uint32_t base = (uint32_t)(PGNUM(vaddr) << PTXSHIFT);
    uint32_t index = GD_UD >> 3;
    uint32_t offset = PGOFF(r_gdt.pd_base);

    gdt = (struct Segdesc*)(base+offset);
    entry = gdt + index;
    // 保存gdt的entry中的数据
    old= *entry;

    // 3.将call_fun_ptr放在get中的[entry]处,dpl是3(用户可调用),segment是GD_KT(kern text)
    SETCALLGATE(*((struct Gatedesc*)entry), GD_KT, call_fun_ptr, 3);

    // 4\5.使用lcall进入ring0，调用call_fun_ptr
    asm volatile(
```

```
        "lcall %0, $0"
        :
        :"i"(GD_UD)
        );
}
void call_fun_ptr()
{
    evil();
    // 6.恢复gdt的entry中的数据
    *entry = old;
    // 7.使用lret离开ring0，回到ring3
    asm volatile("leave");
    asm volatile("lret");
}
```