JOS Lab3 实验报告

李晟 5130379017

本次试验目标是搭建 env，设置 trap 和 system call。

PART1.

首先是第一部分搭建 user-mode environment。

Exercise1
调整 mem_init()来分配 env 结构体数组的内存，这和 lab2 分配 pages 数组是类似的，主要就是加了以下两句：

```
envs=boot_alloc(NENV*sizeof(struct Env));
boot_map_region(kern_pgdir,UENVS,ROUNDUP(NENV*sizeof(struct
Env),PGSIZE),PADDR(envs),PTE_P | PTE_U);
```

不过要注意的是在 page_init()中要修改空闲 page 链表防止 env 数组所占内存被分配出去。

Exercise2
初始化 env 数组中的内容。
env_init()：
构建起空闲的 env 链表，还有将结构体的部分值初始化 0
env_setup_vm(struct Env *)：
用来初始化目标 env 的页表，并将 kernel 部分做一下映射
region_alloc()：
分配一块连续内存
load_icode()：
加载用户代码，并分配一个 page 作为栈，这个函数比较的麻烦，参考了 boot/main.c 还是很难写。要注意的是在将 binary 加载到用户内存区域之前要切换 cr3，在分配完栈之后要把 cr3 切回 kernel。
env_create()：
根据参数中的 type 创建对应类型的 env，然后调用一下 load_icode()。
env_run()：
运行指定的 env，先将 curenv 设置为 RUNNABLE,再将指定的 env 设置为 RUNNING。然后不知道为什么要把 env->env_runs + 1.然后切换一下 cr3 再 pop 出寄存器就好了。

虽然还是 part 1，但是接着就是设置 IDT 了，我觉得这个 lab 可以分成 3 个 part。

Exercise4

首先是在 trapentry.S 中生成 entry point，对应 19 个 trap 号

```
TRAPHANDLER_NOEC(divide_trap,T_DIVIDE);
TRAPHANDLER_NOEC(debug_trap,T_DEBUG);
TRAPHANDLER_NOEC(nmi_trap,T_NMI);
TRAPHANDLER_NOEC(brkpt_trap,T_BRKPT);
TRAPHANDLER_NOEC(oflow_trap,T_OFLOW);
TRAPHANDLER_NOEC(bound_trap,T_BOUND);
TRAPHANDLER_NOEC(illop_trap,T_ILLOP);
TRAPHANDLER_NOEC(device_trap,T_DEVICE);
TRAPHANDLER(dblflt_trap,T_DBLFLT);
TRAPHANDLER(tss_trap,T_TSS);
TRAPHANDLER(segnp_trap,T_SEGNP);
TRAPHANDLER(stack_trap,T_STACK);
TRAPHANDLER(gpflt_trap,T_GPFLT);
TRAPHANDLER(pgflt_trap,T_PGFLT);
TRAPHANDLER_NOEC(fperr_trap,T_FPERR);
TRAPHANDLER_NOEC(align_trap,T_ALIGN);
TRAPHANDLER_NOEC(mchk_trap,T_MCHK);
TRAPHANDLER_NOEC(simderr_trap,T_SIMDERR);
```

在_alltrap 中 push trapframe 然后跳转到 trap

```
pushw $0
pushw %ds
pushw $0
pushw %es
pushal
pushl %esp
movw $GD_KD,%ax
movw %ax,%ds
movw %ax,%es
call trap
```

然后在 trap_init()中初始化 idt，代码太长不贴了。我一开始没分清楚 GD_KD 和 GD_KT 导致 init 中的 idt 到了 data 段。

PART2.

Exercise5

补完 trap_dispatch()，在 exercise5 中只要处理 T_PGFLT，报告是在做完之后再写的，所以包含了 T_BRKPT 和 T_DEBUG。

```
static void
trap_dispatch(struct Trapframe *tf)
{
	switch(tf->tf_trapno){
		case T_PGFLT:
		page_fault_handler(tf);
		break;
		case T_BRKPT:
		case T_DEBUG:
		monitor(tf);
		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;
	}
}
```

Exercise6

完成 syscall 的调用过程

首先是在 lib/syscall.c 的 syscall()函数中写完 sysenter 以及设置返回的地址。

将 esp 放在 ebp 中，把 pc 放在 esi 中。

```
static inline int32_t
syscall(int num, int check, uint32_t a1, uint32_t a2, uint32_t a3, uint32_t a4, uint32_t a5)
{
…
		"movl %%esp,%%ebp\n\t"
		"leal 1f,%%esi\n\t"
		"sysenter\n\t"
		"1:"
…
}
```

不知道为什么我设置其他的 lable 都会报错，后来看了网上的攻略后发现这样写不会报错。

然后在 trapentry.S 中完成 syscall_handler
参数就按照教程中的顺序 push，第 5 个参数反正目前没用到我就瞎 push 了个 0.
然后按照_alltrap 中那样设置 ds 和 es 就好了。

```
sysenter_handler:
/*
 * Lab 3: Your code here for system call handling
 */

    pushl $0
    pushl %edi
    pushl %ebx
    pushl %ecx
    pushl %edx
    pushl %eax
    movw $GD_KD, %ax
    movw %ax, %ds
    movw %ax, %es
    call syscall
    movw $GD_UD, %cx
    movw %cx, %ds
    movw %cx, %es
    movl %ebp,%ecx
    movl %esi,%edx
    sysexit
```

然后完成 kern/syscall.c 中的 syscall

```
int32_t
syscall(uint32_t syscallno, uint32_t a1, uint32_t a2, uint32_t a3, uint32_t a4, uint32_t a5)
{
    // Call the function corresponding to the 'syscallno' parameter.
    // Return any appropriate return value.
    // LAB 3: Your code here.
    switch(syscallno){
        case SYS_getenvid:
            return sys_getenvid();
        case SYS_cputs:
            sys_cputs((const char*)a1, a2);
            return 0;
        case SYS_cgetc:
            return sys_cgetc();
        case SYS_env_destroy:
            return sys_env_destroy(a1);
        case SYS_map_kernel_page:
```

```
                return sys_map_kernel_page((void*)a1,(void*)a2);
        case SYS_sbrk:
                return sys_sbrk(a1);
        default:
                return -E_INVAL;
    }
    //panic("syscall not implemented");
}
```

最后要设置一下 MSRs

在 inc/x86.h 中定义一下 wrmsr

```
static __inline void
wrmsr(uint32_t msr,uint32_t val1,uint32_t val2){
    __asm __volatile("wrmsr" : : "c"(msr),"a"(val1),"d"(val2));
}
```

在 trap_init 后设置一下，于是我直接放在了 trap_init 中，这个是参考学长的

```
void
trap_init(void)
{
…
    extern void sysenter_handler();
    wrmsr(0x174, (uint32_t)GD_KT, 0);
    wrmsr(0x175, (uint32_t)KSTACKTOP, 0);
    wrmsr(0x176, (uint32_t)sysenter_handler, 0);
…
}
```

Exercise7

完成 lib/libmain.c，就 1 行代码

```
void
libmain(int argc, char **argv)
{
    // set thisenv to point at our Env structure in envs[].
    // LAB 3: Your code here.
    thisenv = envs + ENVX(sys_getenvid());
}
```

Exercise8

完成 sys_sbrk

要实现这个函数首先要在 Env 中加入一个 32 位的值来记录目前的堆顶地址

```
struct Env {
…
	// LAB3: might need code here for implementation of sbrk
	uint32_t env_brk;
…
};
```

在 load_icode()中要设置一下这个值

然后完成 sys_sbrk(),这里起始地址和终止地址无论选择是 ROUNDUP 还是 ROUNDDOWN 都不会出错，非常的神奇

```
static int
sys_sbrk(uint32_t inc)
{
	// LAB3: your code sbrk here...
	struct Page *p = NULL;
	char *vaddr = ROUNDUP((char *) curenv->env_brk, PGSIZE);
	char *end = ROUNDUP((char *) (curenv->env_brk + inc -1 ), PGSIZE);
	int r;

	if (inc == 0)
		return curenv->env_brk;
	for(; vaddr <= end; vaddr += PGSIZE) {
		if (!(p = page_alloc(0)))
			panic("sys_sbrk failed!");
		if ((r = page_insert(curenv->env_pgdir, p, vaddr, PTE_P | PTE_U |
					PTE_W)) < 0)
			panic("sys_sbrk: %e", r);
	}
	curenv->env_brk += inc;
	return curenv->env_brk;
}
```

Exercise9

完成 debug command，首先要在 trap_dispatch 中获取这个中断并开启 monitor。

然后再 kern/monitor.c 中加入这 3 条指令，并实现。

这里 si 和 c 根本不知道怎么写，又是看了学长代码 orz 才知道设置一下 tf->tf_eflags 就好了。

```
static struct Command commands[] = {
	{ "help", "Display this list of commands", mon_help },
	{ "kerninfo", "Display information about the kernel", mon_kerninfo },
	{ "backtrace","Display stack backtrace",mon_backtrace },
	{ "x", "Display the memory",mon_x},
	{ "si", "Step by step",mon_si},
	{ "c", "Continue",mon_c}
};

int mon_x(int argc, char **argv, struct Trapframe *tf){
	if(argc != 2){
		cprintf("please enter addr");
		return 0;
	}
	uint32_t addr = strtol(argv[1],NULL,16);
	uint32_t val = *(uint32_t*)addr;
	cprintf("%d\n",val);
	return 0;
}
int mon_si(int argc, char **argv, struct Trapframe *tf){
	tf->tf_eflags = tf->tf_eflags|FL_TF;
	cprintf("tf_eip=0x%x\n",tf->tf_eip);
	env_run(curenv);
	return 0;
}
int mon_c(int argc, char **argv, struct Trapframe *tf){
	tf->tf_eflags&=~FL_TF;
	env_run(curenv);
	return 0;
}
```

Exercise 10

完成 page fault handler

首先在 kern/pmap.c 中完成 user_mem_check()

```
int
user_mem_check(struct Env *env, const void *va, size_t len, int perm)
{
	// LAB 3: Your code here.
	uintptr_t low = (uintptr_t) va;
	uintptr_t high = (uintptr_t) va + len -1;
	uintptr_t idx = low;
	pte_t *pte;
	perm |= PTE_U |PTE_P;
	for(; idx <= high; idx = ROUNDDOWN(idx + PGSIZE,PGSIZE)){
		if(idx >= ULIM){
			user_mem_check_addr = idx;
			return -E_FAULT;
		}
		pte = pgdir_walk(env->env_pgdir,(void*)idx, 0);
		if(pte == NULL ||(*pte &perm)!=perm){
			user_mem_check_addr = idx;
			return -E_FAULT;
		}
	}
	return 0;
}
```

然后完成 page_fault_handler,如果 page_fault 发生在用户态就销毁这个 env，如果在 kernel 态就 panic kernel

最后在 kern/kdebug.c 中的 debuginfo_eip 中插入 user_mem_check

Exercise12

要在用户态进入 kernel 态执行一个用户定义的函数。

这个一开始果断不知道怎么写啊，然后我就只能去参考学长代码了。

主要使用 sgdt 这个非特权指令以及 kernel 暴露在外的 sys_map_kernel_page 函数。

首先用 sgdt 获取 gdt，然后把 kernel 的一级页表映射到用户自己预留的一块内存空间。然后获取到 gdt 自身所在地址。然后把用 SETCALLGATE 设置 gdt 某个段的为 UD_KT，并将其内容设置为要执行的函数，不能设置 gdt[1]和 gdt[2]因为这两个分别是 UD_KT 和 UD_KD，设置了使 ring0 态执行时出错，然后 gdt[3]和 gdt[4]也不能用，因为 gdt[3]修改后跳转回来会找不到正确的用户代码，gdt[4]修改后会丢失用户的数据段（因为之前的 gdt 相应项的内容存在了用户栈上）。最后就只能用 gdt[5]，在回来的时候再把 gdt[5]改回去。