操作系统 第三次实验报告

周子锐 2100011032

2023 年 10 月 27 日

完成的 Challenge 为:

Challenge! Modify the JOS kernel monitor so that you can 'continue' execution from the current location (e.g., after the int3, if the kernel monitor was invoked via the breakpoint exception), and so that you can single-step one instruction at a time. You will need to understand certain bits of the EFLAGS register in order to implement single-stepping.

Exercise 1

与 lab2 类似, 根据注释完成代码.

```
//////////////////////////////////////////////////////////////////////
// Make 'envs' point to an array of size 'NENV' of 'struct Env'.
// LAB 3: Your code here.
envs = (struct Env*) boot_alloc(NENV * sizeof(struct Env));
...
//////////////////////////////////////////////////////////////////////
// Map the 'envs' array read-only by the user at linear address UENVS
// (ie. perm = PTE_U | PTE_P).
// Permissions:
// - the new image at UENVS -- kernel R, user R
// - envs itself -- kernel RW, user NONE
// LAB 3: Your code here.
boot_map_region(kern_pgdir, UENVS, PTSIZE, PADDR(envs), PTE_U | PTE_P);
```

Exercise 2

根据注释完成以下函数.

void env_init(void)

初始化 envs 数组中的元素, 注意因为要让链表头为 0 所以应倒序遍历.

```
void
env_init(void)
{
    // Set up envs array
    for (int i = NENV - 1; i >= 0; i--) {
        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();
}
```

static int env_setup_vm(struct Env *e)

对每一个 env 新建一个页目录, 可以直接使用 Kernel 的页目录作为模板.

```
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;

    p->pp_ref++;
    e->env_pgdir = (pde_t *)page2kva(p);
    memcpy(e->env_pgdir, kern_pgdir, PGSIZE);

    // UVPT maps the env's own page table read-only.
    // Permissions: kernel R, user R
    e->env_pgdir[PDX(UVPT)] = PADDR(e->env_pgdir) | PTE_P | PTE_U;

```

```c
    return 0;
}
```

static void region_alloc(struct Env *e, void *va, size_t len)

为 env 申请一段连续的虚拟地址空间, 并映射到物理地址空间. 逐页申请并映射即可.

```c
static void
region_alloc(struct Env *e, void *va, size_t len)
{
    if (!len) {
        return;
    }
    uintptr_t vstart = ROUNDDOWN((uintptr_t)va, PGSIZE);
    uintptr_t vend = ROUNDUP((uintptr_t)va + len, PGSIZE);
    if (vstart > vend) {
        panic("region_alloc: invalid va and len");
    }
    for (; vstart < vend; vstart += PGSIZE) {
        struct PageInfo *pp = page_alloc(ALLOC_ZERO);
        if (!pp) {
            panic("region_alloc: page_alloc failed");
        }
        if (page_insert(e->env_pgdir, pp, (void *)vstart, PTE_U | PTE_W) < 0) {
            panic("region_alloc: page_insert failed");
        }
    }
}
```

static void load_icode(struct Env *e, uint8_t *binary)

将一个 ELF 格式的二进制文件加载到 env 中, 参考 boot/main.c 实现. 这里可以预先切换成新环境的页目录, 以便于 memcpy 复制连续的 pages.

```c
static void
load_icode(struct Env *e, uint8_t *binary)
{
    struct Elf *elfhdr;
    struct Proghdr *ph, *eph;
    elfhdr = (struct Elf *)binary;
```

```
    if (elfhdr->e_magic != ELF_MAGIC) {
        panic("load_icode: invalid ELF header");
    }
    ph = (struct Proghdr *)((uintptr_t)binary + elfhdr->e_phoff);
    eph = ph + elfhdr->e_phnum;
    lcr3(PADDR(e->env_pgdir));
    for (; ph < eph; ph++) {
        if (ph->p_type == ELF_PROG_LOAD) {
            region_alloc(e, (void *)ph->p_va, ph->p_memsz);
            memset((void *)ph->p_va, 0, ph->p_memsz);
            memcpy((void *)ph->p_va, (void *)(binary + ph->p_offset), ph->p_filesz);
        }
    }
    e->env_tf.tf_eip = elfhdr->e_entry;
    lcr3(PADDR(kern_pgdir));

    // Now map one page for the program's initial stack
    // at virtual address USTACKTOP - PGSIZE.
    region_alloc(e, (void *)(USTACKTOP - PGSIZE), PGSIZE);
}
```

void env_create(uint8_t *binary, enum EnvType type)

分配一个可用的 Env 来加载一个二进制文件.

```
void
env_create(uint8_t *binary, enum EnvType type)
{
    struct Env *e;
    int ret = env_alloc(&e, 0);
    if (ret < 0) {
        panic("env_create: %e", ret);
    }
    load_icode(e, binary);
    e->env_type = type;
}
```

void env_run(struct Env *e)

切换到给定环境中运行. 根据注释完成即可.

```c
void
env_run(struct Env *e)
{
    if ((curenv) && (curenv->env_status == ENV_RUNNING)) {
        curenv->env_status = ENV_RUNNABLE;
    }
    curenv = e;
    curenv->env_status = ENV_RUNNING;
    curenv->env_runs++;
    lcr3(PADDR(curenv->env_pgdir));

    env_pop_tf(&(curenv->env_tf));
}
```

在实现完上述函数后运行 qemu, 可以发现确实触发了一个 triple fault.

```
npages = 32768
check_page_free_list() succeeded!
check_page_alloc() succeeded!
check_page() succeeded!
check_kern_pgdir() succeeded!
check_page_free_list() succeeded!
check_page_installed_pgdir() succeeded!
[00000000] new env 00001000
EAX=00000000 EBX=00000000 ECX=0000000d EDX=eebfde88
ESI=00000000 EDI=00000000 EBP=eebfde60 ESP=eebfde54
EIP=00800b52 EFL=00000092 [--S-A--] CPL=3 II=0 A20=1 SMM=0 HLT=0
ES =0023 00000000 ffffffff 00cff300 DPL=3 DS   [-WA]
CS =001b 00000000 ffffffff 00cffa00 DPL=3 CS32 [-R-]
SS =0023 00000000 ffffffff 00cff300 DPL=3 DS   [-WA]
DS =0023 00000000 ffffffff 00cff300 DPL=3 DS   [-WA]
FS =0023 00000000 ffffffff 00cff300 DPL=3 DS   [-WA]
GS =0023 00000000 ffffffff 00cff300 DPL=3 DS   [-WA]
LDT=0000 00000000 00000000 00008200 DPL=0 LDT
TR =0028 f0181bc0 00000067 00408900 DPL=0 TSS32-avl
GDT=     f011c300 0000002f
IDT=     f01813a0 000007ff
CR0=80050033 CR2=00000000 CR3=003bc000 CR4=00000000
DR0=00000000 DR1=00000000 DR2=00000000 DR3=00000000
DR6=ffff0ff0 DR7=00000400
EFER=0000000000000000
Triple fault.  Halting for inspection via QEMU monitor.
QEMU: Terminated
```

图 1: Exercise 2 完成后的 qemu 运行信息

同时使用 gdb 调试发现这的确是由 int 指令引起的.

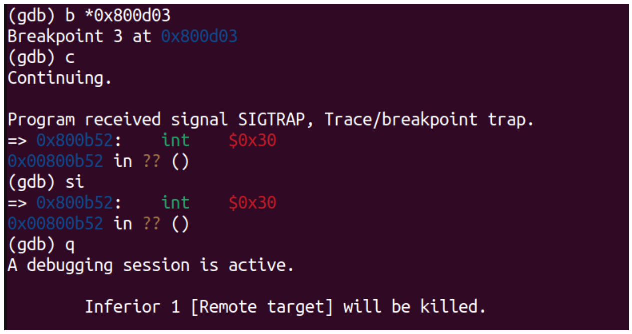

图 2: Exercise 2 完成后的 gdb 调试信息

Exercise 4

在 `kern/trap.c` 中将生成的中断处理函数用 SETGATE 宏加载到 IDT 中, 需要指出的是, 这里为 Breakpoint 和 Syscall 两个中断设置了特殊的 DPL, 以便于在用户态下使用.

```
void handler_divide();
void handler_debug();
void handler_nmi();
void handler_brkpt();
void handler_oflow();
void handler_bound();
void handler_illop();
void handler_device();
void handler_dblflt();
void handler_tss();
void handler_segnp();
void handler_stack();
void handler_gpflt();
void handler_pgflt();
void handler_fperr();
void handler_align();
void handler_mchk();
void handler_simderr();
void handler_syscall();

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

```
    SETGATE(idt[T_DIVIDE], 0, GD_KT, handler_divide, 0);
    SETGATE(idt[T_DEBUG], 0, GD_KT, handler_debug, 0);
    SETGATE(idt[T_NMI], 0, GD_KT, handler_nmi, 0);
    SETGATE(idt[T_BRKPT], 0, GD_KT, handler_brkpt, 3);
    SETGATE(idt[T_OFLOW], 0, GD_KT, handler_oflow, 0);
    SETGATE(idt[T_BOUND], 0, GD_KT, handler_bound, 0);
    SETGATE(idt[T_ILLOP], 0, GD_KT, handler_illop, 0);
    SETGATE(idt[T_DEVICE], 0, GD_KT, handler_device, 0);
    SETGATE(idt[T_DBLFLT], 0, GD_KT, handler_dblflt, 0);
    SETGATE(idt[T_TSS], 0, GD_KT, handler_tss, 0);
    SETGATE(idt[T_SEGNP], 0, GD_KT, handler_segnp, 0);
    SETGATE(idt[T_STACK], 0, GD_KT, handler_stack, 0);
    SETGATE(idt[T_GPFLT], 0, GD_KT, handler_gpflt, 0);
    SETGATE(idt[T_PGFLT], 0, GD_KT, handler_pgflt, 0);
    SETGATE(idt[T_FPERR], 0, GD_KT, handler_fperr, 0);
    SETGATE(idt[T_ALIGN], 0, GD_KT, handler_align, 0);
    SETGATE(idt[T_MCHK], 0, GD_KT, handler_mchk, 0);
    SETGATE(idt[T_SIMDERR], 0, GD_KT, handler_simderr, 0);
    SETGATE(idt[T_SYSCALL], 0, GD_KT, handler_syscall, 3);

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

接下来是 `kern/trapentry.S`, 首先使用给出的宏为各类中断生成入口:

```
TRAPHANDLER_NOEC(handler_divide, T_DIVIDE)
TRAPHANDLER_NOEC(handler_debug, T_DEBUG)
TRAPHANDLER_NOEC(handler_nmi, T_NMI)
TRAPHANDLER_NOEC(handler_brkpt, T_BRKPT)
TRAPHANDLER_NOEC(handler_oflow, T_OFLOW)
TRAPHANDLER_NOEC(handler_bound, T_BOUND)
TRAPHANDLER_NOEC(handler_illop, T_ILLOP)
TRAPHANDLER_NOEC(handler_device, T_DEVICE)
TRAPHANDLER(handler_dblflt, T_DBLFLT)
TRAPHANDLER(handler_tss, T_TSS)
TRAPHANDLER(handler_segnp, T_SEGNP)
TRAPHANDLER(handler_stack, T_STACK)
TRAPHANDLER(handler_gpflt, T_GPFLT)
TRAPHANDLER(handler_pgflt, T_PGFLT)
```

```
TRAPHANDLER_NOEC(handler_fperr, T_FPERR)
TRAPHANDLER(handler_align, T_ALIGN)
TRAPHANDLER_NOEC(handler_mchk, T_MCHK)
TRAPHANDLER_NOEC(handler_simderr, T_SIMDERR)
TRAPHANDLER_NOEC(handler_syscall, T_SYSCALL)
```

接着是 `_alltraps` 函数，这是为 `TRAPHANDLER` 提供的跳转函数. 对于第一条要求，我们注意到 `kern/trap.h` 中的定义

```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

同时注意到在执行 `_alltraps` 之前已经将 `tf_trapno` 及其之后的信息倒序压入到栈中了，所以只需要补充其上方的信息即可.

```
.global _alltraps
.type _alltraps, @function
.align 2
_alltraps:
    pushw $0
    pushw %ds
    pushw $0
    pushw %es
    pushal

    movl $GD_KD, %eax
```

```
    movw %ax, %ds
    movw %ax, %es
    pushl %esp
    call trap
```

Question 1

注意到在运行这些处理程序之前, 部分中断还会额外将错误码压入栈中, 这是使用统一的中断处理程序所不能实现的.

Question 2

根据上面对 IDT 的设置, `int 14` 这一异常在用户态无法申请, 因此会触发第 13 号异常即 general protection exception. 如果操作系统允许用户处罚这一缺页异常, 会使得用户能更加方便地操作虚拟内存, 对程序的安全性造成影响.

Exercise 5&6

直接加上对应条件判断即可.

```
static void
trap_dispatch(struct Trapframe *tf)
{
    // Handle processor exceptions.
    if (tf->tf_trapno == T_PGFLT) {
        page_fault_handler(tf);
        return;
    } else if (tf->tf_trapno == T_BRKPT) {
        monitor(tf);
        return;
    }

    // Unexpected trap: The user process or the kernel has a bug.
    ...
}
```

Challenge

这个 Challenge 需要我们在接受到 breakpoint 异常的时候能够继续执行, 并且能够单步执行.

通过查阅这里可以知道 `EFLGAS` 的 TF 位可以控制程序的单步执行. 因此可以根据如下的代码在 `kern/monitor.c` 中添加对 `continue` 和 `stepi` 命令的支持.

```c
int
mon_continue(int argc, char **argv, struct Trapframe *tf) {
    if (!(tf && (tf->tf_trapno == T_DEBUG || tf->tf_trapno == T_BRKPT) &&
          ((tf->tf_cs & 3) == 3)))
        return 0;
    tf->tf_eflags &= ~FL_TF;
    return -1;
}


int
mon_stepi(int argc, char **argv, struct Trapframe *tf) {
    if (!(tf && (tf->tf_trapno == T_DEBUG || tf->tf_trapno == T_BRKPT) &&
          ((tf->tf_cs & 3) == 3)))
        return 0;
    tf->tf_eflags |= FL_TF;
    return -1;
}
```

在 `kern/init.c` 中修改执行程序为 breakpoint 进行测试得到如下结果.

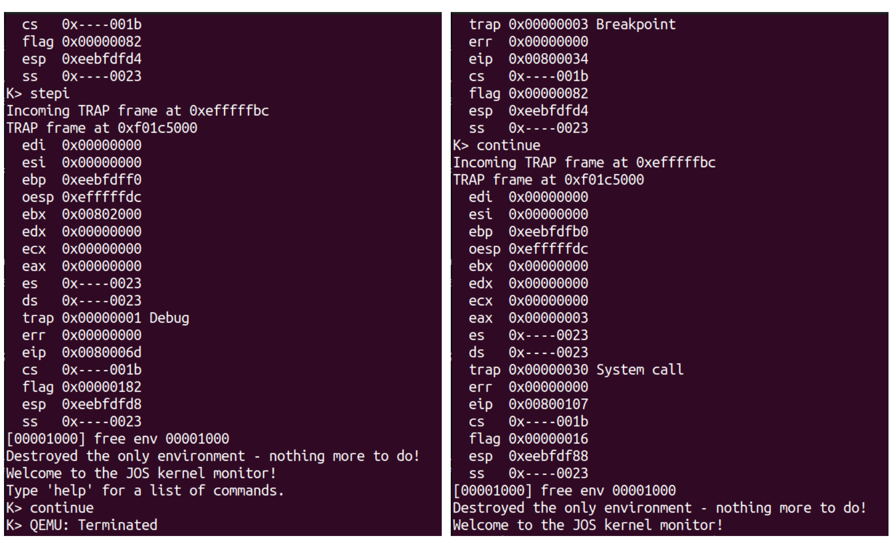

图 3: 运行 `stepi` 命令和 `continue` 命令的结果

Question 3

因为与其他常规的中断或异常不同，breakpoint 异常是由用户发起的，当 IDT 中设置其的 DPL 为 0 时，用户态下若尝试触发此异常则会触发 general protection fault. 因此为了成功触发 breakpoint 需要在加载 IDT 设置其对应项的 DPL 为 3.

Question 4

这样做是为了防止用户能够随意操作只有操作系统才能操作的资源，从而实现对 kernel 的攻击.

Exercise 7

在 `kern/trap.c/trap_dispatch()` 中添加对 `syscall` 的处理.

```
...
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;
}
...
```

在 `kern/syscall.c` 中实现对不同类别的 syscall 的 dispatch，各个具体 syscall 的 API 参见 `lib/syscall.c`.

```
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.
```

```
    //panic("syscall not implemented");

    switch (syscallno) {
    case SYS_cputs:
        sys_cputs((const char *)a1, (size_t)a2);
        return 0;
    case SYS_cgetc:
        return sys_cgetc();
    case SYS_getenvid:
        return sys_getenvid();
    case SYS_env_destroy:
        return sys_env_destroy((envid_t)a1);
    default:
        return -E_INVAL;
    }
}
```

Exercise 8

在 `lib/libmain.c` 中加上这一句话即可

```
    thisenv = envs + ENVX(sys_getenvid());
```

Exercise 9

对 `kern/pmap.c` 中的 `user_mem_check()` 函数, 可以仿照 Exercise 2 中的 `region_alloc()` 函数, 逐页判断即可.

```
int
user_mem_check(struct Env *env, const void *va, size_t len, int perm)
{
    if (len == 0) {
        return 0;
    }
    uintptr_t vstart = ROUNDDOWN((uintptr_t)va, PGSIZE);
    uintptr_t vend = ROUNDUP((uintptr_t)va + len, PGSIZE);
    if (vend > ULIM) {
        user_mem_check_addr = MAX(vstart, ULIM);
        return -E_FAULT;
```

```
    }
    perm |= PTE_P;
    for (; vstart < vend; vstart += PGSIZE) {
        pte_t *pte = pgdir_walk(env->env_pgdir, (void *)vstart, 0);
        if ((!pte) || (((*pte) & perm) != perm)) {
            user_mem_check_addr = MAX(vstart, (uintptr_t)va);
            return -E_FAULT;
        }
    }
    return 0;
}
```

在 kern/syscall.c 中, 目前只有 SYSY_puts 涉及到访存相关的内容, 为其补充检查即可.

```
static void
sys_cputs(const char *s, size_t len)
{
    // Check that the user has permission to read memory [s, s+len).
    // Destroy the environment if not.
    user_mem_assert(curenv, s, len, PTE_U);

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

在 kern/kdebug.c 中, 补充对 usd, stabs, stabstr 的检查即可.

```
int
debuginfo_eip(uintptr_t addr, struct Eipdebuginfo *info)
{
        ...
        // Make sure this memory is valid.
        // Return -1 if it is not. Hint: Call user_mem_check.
        if (user_mem_check(curenv, usd, sizeof(struct UserStabData), PTE_U) < 0)
            return -1;
        ...
        // Make sure the STABS and string table memory is valid.
        if (user_mem_check(curenv, stabs, (uintptr_t)stab_end - (uintptr_t)stabs, PTE_U) < 0)
            return -1;
        if (user_mem_check(curenv, stabstr, stabstr_end - stabstr, PTE_U) < 0)
            return -1;
        ...
```

```
}
```

测试结果

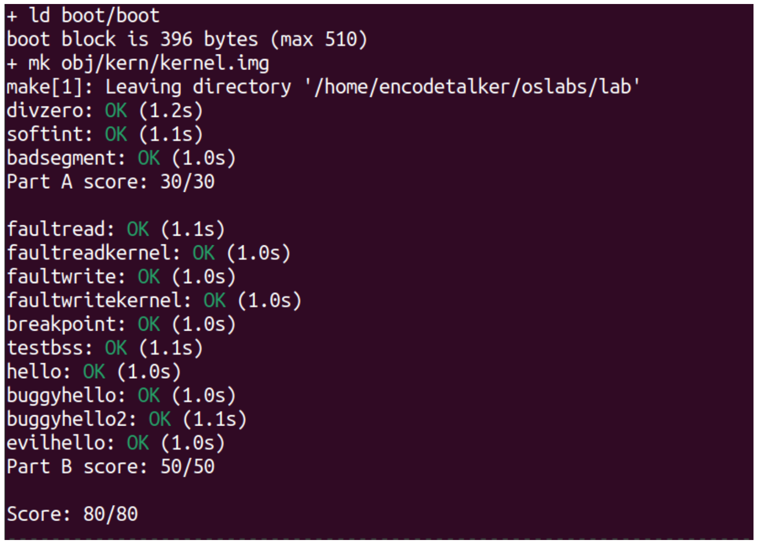

图 4: 评测结果

This completes the lab.