5137169011　陈伟业

Question1: What is the purpose of having an individual handler function for each exception/interrupt? (i.e., if all exceptions/interrupts were delivered to the same handler, what feature that exists in the current implementation could not be provided?)

因为不同的 exception 和 interrupt 需要不同的处理函数，他们有不同的参数和不同的内容，使用不同的 handler 可以直接调用他们对应的处理函数。而使用同一个 handler 则无法知道 exception 或 interrupt 的类型，也就不能调用其对应的处理函数。

Question2: Did you have to do anything to make the `user/softint` program behave correctly? The grade script expects it to produce a general protection fault (trap 13), but `softint`'s code says `int $14`. *Why* should this produce interrupt vector 13? What happens if the kernel actually allows `softint`'s `int $14` instruction to invoke the kernel's page fault handler (which is interrupt vector 14)?

我赋予了 int 14 内核级别的 DPL，故用户访问时会造成 protection fault。如果允许用户使用 int 14，则不会向栈压入错误码，导致栈错位。

Question3: The break point test case will either generate a break point exception or a general protection fault depending on how you initialized the break point entry in the IDT (i.e., your call to `SETGATE` from `trap_init`).

设置 breakpoint 的 DPL 为 3 即可。

Question4 What do you think is the point of these mechanisms, particularly in light of what the `user/softint` test program does?

不允许用户接触内核的保护机制，使得整个系统更加安全。

## 1. Environment

env_init():遍历envs 数组，并设置status为ENV_FREE，设置env_id为0，设置link为其后一项。

env_setup_vm(): 设置e->env_pgdir 为p 的虚拟地址，将p 的pp_ref 加1，复制kern_pgdir到PDX(UTOP)到NPTENTRIES的区域。同时，PDX[UVPT]项置为e的env_pgdir

region_alloc(): 从va到va + len的内存分配新的大小为PGSIZE的页
load_icode():遍历ph 。当读到ELF_PROG_LOAD 项时，根据filesz 与memsz 分配相应的页。同时用lcr3切换页表。更新eip，并为用户栈分配空间。
env_create(): 调用load_icode()初始化一个用户进程
env_run():设置status为ENV_RUNNABLE,将e 的状态置为ENV_RUNNING，调用lcr3切换到e 的页表,并调用env_pop_tf()将e 的Trapframe。

## 2. Trap

调用TRAPHANDLER_NOEC()和TRAPHANDLER()初始化各trap 的处理函数。在_alltraps中，将段寄存器%ds 和%es 切换到GD_KD，调用trap 函数。同时，将breakpoint 的调用级别设置为ring3。 trap_init()首先用extern 声明trapentry.S中的处理函数，并调用SETGATE 设置各项trap 的处理函数。在trap_dispatch()中，对pagefault，调用page_fault_handler() 函数处理，对breakpoint 和debug，调用monitor 函数，

## 3. System Call

在trap_init()，使用wrmsr 注册系统调用函数，段设置为GD_KT，栈指针设置为KSTASKTOP. 保存栈指针%esp 到%ebp，并调用sysenter_handler 函数。在lib/trapentry.S 中sysenter_handler 函数：按照%edi, %ebx, %ecx, %edx, %eax的顺序依次压栈，将GD_KD 更新到段寄存器%ds 和%es 后，调用kern/syscall.c 中的syscall 函数。函数返回后，复原GD_UD，并将返回地址和栈指针分别存入%edx与%ecx。对于sys_sbrk(), 在struct Env 中新增属性brk 记录堆顶。在sys_sbrk 中根据inc 分配相应的页。

## 4. breakpoint

将trap_init()和breakpoint 的调用级别设置为ring3。mon_x()从argv[1]中取出地址，并输出其中的内容。mon_si()将tf_tf_eflags 置为tf->tf_eflags |= FL_TF, 并且输出tf_eip 的值。然后调用env_run。mon_c()将tf->tf_eflags 的FL_TF 还原，并调用env_run.

## 5. evilhello2

调用sys_map_kernel_page 将gdtd.pd_base 映射到va在gdt 表中找到用户数据段，并保存入old_entry，调用SETCALLGATE 重置调用函数。调用lcall $20, $0 调用wrappered_call()函数。wrappered_call()中，调用evil()函数， 并在调用完成后，将entry 还原成old_entry，使用popl %ebp 和lret 指令返回。