Design

Part A: User Environments and Exception Handling

1. Exercise 1: /kern/pmap.c: mem_init()
   Just allocate space for envs and map it to [UENVS, UENVS + PTSIZE], set permission PTE_U | PTE_P

2. Exercise 2: implement functions in /kern/env.c
   Implements functions by following the hints. It take a bit more time to make load_icode correct, following the stop in /boot/main.c

Part B: Page Faults, Breakpoints Exceptions, and System Calls

3. Exercise 4, 5: implement trap entry in /kern/trap.c and /kern/trapentry.S
   _alltraps: push required information onto stack, by the reverse order defined in Trapframe struct then call trap
   Some work for the challenge:
   Inspired by the implements in xv6
   In trapentry.S firstly, map the interrupt number defined in inc/trap.h with and handler(named "handler"+interrupt_number) using the marco TRAPHANDLER and TRAPHANDLER_NOEC. Then, make a data section in trapentry.S name vectors, each element in vectors refer to a handler above.
   Trap.c: trap_init(): using the vectors defined above. Map idt elements to vectors elements using SETGATE. Except T_BRKPT, set the dpl to 0x3(user level), others all set to 0x0(super level)
   Trap.c: trap_dispatch: Just two kind of trap number need special dispatch. One for T_BRKPT => monitor(), one for T_PGFLT => page_fault_handler()
   Trap.c: page_fault_handler(): should panic when kernel mode page fault happens, which is conditioned by (tf->tf_cs & 0x3) == 0

4. Exercise 6:
   Sysenter_handler: push %edi, %ebx, %ecx, %edx, %eax in order, call syscall, then move %ebp -> %ecx, %esi -> %edx, sysexit
   MSRs: add a function wrmsr in inc/x86.h then enable sep cpu by adding lines in trap_init_percpu. Reference: https://lwn.net/Articles/18414/
   lib/syscall.c: syscall() :
      1). first push %esp, pop %ebp to modify %ebp indirectly

2). Then make a label just after sysenter, so before sysenter get the address after Sysenter by leal. Because gcc compile will fail when try to leal a label defined After, add %= after a label to let gcc generate unique label for you.

5. Exercise 7:

Thisenv is not inited. Just add one line `thisenv = envs + ENVX(sys_getenvid());` in lib/libmain.c

6. Exercise 8: sys_sbrk()

First add a field called env_break in struct Env

In kern/syscall.c: sys_sbrk(): just increase curenv->env_break ROUNDUP(inc, PGSIZE)

And map the new allocated space with region_alloc

7. Exercise 9: breakpoint

Trap_dispatch is described in exercise 4, 5

Here need implement three command: si, x and c

Register the three new command by adding them into commands in kern/monitor.c

Then define three functions: mon_si, mon_x and mon_c

1). Mon_x: print the value of address va(in hex). So things to do is just parse the string Of the address to int and get the value by dereference(*)

2), mon_c: continue executing. Set tf->tf_eflags to tf_efalgs & (~FL_TF) (remove the FL_TF bits) and return -1

3). Mon_si: execute one instruction. Add the FL_TF bit to tf->tf_efalgs, then print the Info required just like mon_backtrace

8. Exercise 10

User_mem_check:

Create a function user_mem_check_page to check permission of the page where va resides. It will fail (return -1) when va >= ULIM or permission not match

Then in user_mem_check() just check permission of pages from va to va+len

9. Exercise 12: evilhello2

Global variables:

Struct Gatedesc *gd

Char gdt_pgs[2*PGSIZE]: mapped GDT in user space

Void (*evil_func)(): the evil function

Implements the ring0_call:

1). Store GDT into dp using sgdt

2). Map dp.pd_base to ROUNDUP(gdt_pgs, PGSIZE)

3). Setup call gate with SETCALLGATE: gdt entry gd => call_evil_func

4). Lcall GD_TSS0, $0