CS 444/544 OS II
Lab Session #5

User Environments and Exception Handling

(Lab3 – Part A)

Before Start

- Lab 3 is for creating User environment
  - Will run your code in Ring 3!

- You will mainly edit kern/env.c
- Also,
  - kern/pmap.c
  - kern/trapentry.S
  - kern/syscall.c
  - lib/syscall.c
  - inc/syscall.h

Before Start

- If you are suffering an infinite loop of JOS, then try to check your path

```
[jangye@os2 ~$] echo $PATH
/usr/local/bin:/nfs/stak/users/jangye/bin:/usr/lib64/qt-3.3/bin:/nfs/stak/users/jangye/perl5/bin:/usr/local/bin:/usr/bin:/usr/l
ocal/sbin:/usr/sbin:/opt/dell/srvadmin/bin
```

- It must have ~/bin as the path
  - Otherwise, it will execute OS's default QEMU
- We have to use a modified version of QEMU
  - Which is at under ~/bin/qemu-system-i386

- Easy solution: use BASH! (my dotfiles includes that for you)

Exercise 1: ENVs

Exercise 1. Modify `mem_init()` in `kern/pmap.c` to allocate and map the `envs` array. This array consists of exactly `NENV` instances of the `Env` structure allocated much like how you allocated the `pages` array. Also like the `pages` array, the memory backing `envs` should also be mapped user read-only at `UENVS` (defined in `inc/memlayout.h`) so user processes can read from this array.

You should run your code and make sure `check_kern_pgdir()` succeeds.

- Use boot_alloc to allocate ENVS (as we do for pages)
- Use boot_map_region to make
  - envs RW for kernel,
  - UENV R for both kernel and user

One tip

- Fix the line below if you have a weird memory error

```
if (!nextfree) {
    extern char end[];
    nextfree = ROUNDUP((char *) end + 1, PGSIZE);
}
```

- In boot_alloc(),
  - Add **+1** to the end…

Exercise 2: Implement funcs for ENV

- env_init()

```
// Mark all environments in 'envs' as free, set their env_ids to 0,
// and insert them into the env_free_list.
// Make sure the environments are in the free list in the same order
// they are in the envs array (i.e., so that the first call to
// env_alloc() returns envs[0]).
//
void
env_init(void)
```

- Building linked-list (similar to page_free_list)
  - But, we need to keep the **order** (envs[0] is the first free one)

Exercise 2: Implement funcs for ENV

- env_setup_vm()

- Create a new page directory for an ENV
- Copy all kernel mappings above UTOP, and set UVPT

- Check pp_ref..

```
for(int i=PDX(UTOP); i<NPDENTRIES; ++i) {
    e->env_pgdir[i] = kern_pgdir[i];
}

// 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;
```

Exercise 2: Implement funcs for ENV

- region_alloc()
- Similar to boot_map_region, but it is only virtually contiguous
  - boot_map_region allocates both physically and virtually contiguous memory

- Use functions wisely
  - Page_lookup()
  - Page_alloc()
  - Page_insert()

Exercise 2: Implement funcs for ENV

- load_icode()
- Get ELF Header:

```
struct Elf *elf = (struct Elf *) binary;
```

- Understand how ELF file is formatted…
  - https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
  - Refer to how bootmain() in boot/main.c read the code
  - Use virtual address (from the header) for mapping
  - Set the entry point as

```
e->env_tf.tf_eip = elf->e_entry;
```

- Use
  - memset, memcpy
  - region_alloc

Exercise 2: Implement funcs for ENV

- load_icode()
- Get ELF Header:

```
struct Elf *elf = (struct Elf *) binary;
```

- Understand how ELF file is formatted…
  - https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
  - Refer to how bootmain() in boot/main.c read the code
  - Use virtual address (from the header) for mapping
  - Set the entry point as

```
e->env_tf.tf_eip = elf->e_entry;
```

- Use
  - memset, memcpy
  - region_alloc

Exercise 2: Implement funcs for ENV

- env_create()
  - Allocate a new env, set type, and load binary

- Use
  - env_alloc()
  - load_icode()

Exercise 2: Implement funcs for ENV

- env_run()
  - Follow the comment…

```
// Step 1: If this is a context switch (a new environment is running):
//     1. Set the current environment (if any) back to
//        ENV_RUNNABLE if it is ENV_RUNNING (think about
//        what other states it can be in),
//     2. Set 'curenv' to the new environment,
//     3. Set its status to ENV_RUNNING,
//     4. Update its 'env_runs' counter,
//     5. Use lcr3() to switch to its address space.
// Step 2: Use env_pop_tf() to restore the environment's
//     registers and drop into user mode in the
//     environment.
```

Exercise 3: Read Intel Manual Chapter 6

- Please read, and focus on Error Code
  - https://os.unexploitable.systems/r/ia32/IA32-3A.pdf

Table 6-1. Protected-Mode Exceptions and Interrupts

Vector	Mne-monic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	Debug Exception	Fault/ Trap	No	Instruction, data, and I/O breakpoints; single-step; and others.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT 3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD2 instruction or reserved opcode.[1]
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction.[2]
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.

Exercise 4: Implement Trap Handlers

- kern/trapentry.S
  - Use MACROs to define handlers, depending on their error code existence

- If error code does not exists:

```
TRAPHANDLER_NOEC(t_divide, T_DIVIDE);
```

- If error code exists:

```
TRAPHANDLER(t_dblflt, T_DBLFLT);
```

Exercise 4: Implement Trap Handlers

- kern/trapentry.S

- Implement _alltraps:
  - Both TRAPHANDER_EC and TRAPHANDER_NOEC runs _alltraps
  - Push
    - ds
    - es
    - All general purpose registers
  - Change DS and ES to kernel DS ($GD_KD)
  - Push esp
  - Call trap() (kern/trap.c)

Exercise 4: Implement Trap Handlers

- Please think about how we store trap context and use
  - struct TrapFrame

```
struct PushRegs {
    /* registers as pushed by pusha */
    uint32_t reg_edi;
    uint32_t reg_esi;
    uint32_t reg_ebp;
    uint32_t reg_oesp;      /* Useless */
    uint32_t reg_ebx;
    uint32_t reg_edx;
    uint32_t reg_ecx;
    uint32_t reg_eax;
} __attribute__((packed));
```

```
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));
```

Exercise 4: Implement Trap Handlers

- kern/trap.c
  - Implement trap_init()
  - Use reference in comment, e.g.,

```
SETGATE(idt[T_DIVIDE], 0, GD_KT, t_divide, 0);
```

  - You must define t_divide (for the above case) in trap.c

```
void t_divide();
```

  - Do this for all traps that you would like to handle…

Exercise 4: Implement Trap Handlers

- kern/trap.c
  - Implement trap_init()

  - T_BRKPT and T_SYSCALL must be available to Ring 3
  - E.g.,

```
SETGATE(idt[T_SYSCALL], 0, GD_KT, t_syscall, 3);
```

Tips

- How to get the current pgdir?
  - `physaddr_t pgdir_addr = rcr3()`
- How to set CR3?
  - `lcr3(PADDR(e->env_pgdir)`