CS 444/544 OS II
Lab Tutorial #5

User Environments and Exception Handling

Prof. Sibin Mohan | Spring 2022

Adapted from content originally created by: Prof. Yeongjin Jang

Before Starting Lab 3

- 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 $HOME/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 $HOME/bin/qemu-system-i386

- Easy solution: use BASH! (CS444/544 dotfiles includes that for you)

Checkout & Merge Lab3

- If you are currently on lab2 branch, then let's switch to lab3
  - $ git status
  - … please commit all changes..

  - $ git checkout lab3
  - $ git merge lab2
  - …
  - Start!

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…
  - This is for making the area for ‘envs’ array in-use

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..
  - p->pp_ref += 1

```
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(struct Env *e, void *va, size_t len)
- 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()
  - Load an application program to memory space
  - Load to the environment's memory space
- Now we are using kern_pgdir in running kernel
  - If we load the code in current memory space, it will be loaded to kernel
- We want to load the program to the memory space of ENV
  - We need to switch the page directory
  - How? Updating CR3.

Exercise 2: Implement funcs for ENV

- Change page directory from kern_pgdir to env's pgdir

```
// LAB 3: Your code here.
uint32_t prev_cr3 = rcr3();
lcr3(PADDR(e->env_pgdir));
```

- CR3 points to the current page directory
  - store previous cr3 (kern_pgdir) to prev_cr3
  - Load the page directory of the environment to CR3
    - cr3 = PADDR(e->env_pgdir)
- After that, we can access virtual memory space of the ENV

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

- Don’t forget to restore the cr3 when returning from load_icode

```
// LAB 3: Your code here.
uint32_t prev_cr3 = rcr3();
lcr3(PADDR(e->env_pgdir));
```

- Restore the cr3 to the previous value before returning from load_icode

```
// change cr3 to previous one
lcr3(prev_cr3);
```

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

- Do not have to read all but focus on Error Code and Interrupt numbers
  - 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)

```
_alltraps:
    pushl %ds
    pushl %es
    pushal
```

```
movl $GD_KD, %eax
movw %ax, %ds
movw %ax, %es
```

1. push values to make the stack look like a struct Trapframe
2. load `GD_KD` into `%ds` and `%es`
3. `pushl %esp` to pass a pointer to the Trapframe as an argument to trap()
4. `call trap` (can `trap` ever return?)

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()`
- Virtual address of the current pgdir?
  - `KADDR(rcr3())`
- How to set the page directory to CR3?
  - `lcr3(PADDR(e->env_pgdir)`

How to Run USER Program?

```
[blue9057@blue9057-vm-jos (lab3) ~/jos$] ls obj/user/ | grep -v \\.
badsegment
breakpoint
buggyhello
buggyhello2
divzero
evilhello
faultread
faultreadkernel
faultwrite
faultwritekernel
hello
softint
testbss
```

make run-[NAME]-nox

Run the program, e.g., make run-divzero-nox

make run-[NAME]-nox-gdb

Run the program with gdb, e.g., make run-divzero-nox-gdb

make run-divzero-nox

```
[blue9057@blue9057-vm-jos (lab3) ~/jos$] make run-divzero-nox
make[1]: Entering directory '/home/blue9057/jos'
+ cc kern/init.c
+ ld obj/kern/kernel
ld: warning: section `.bss' type changed to PROGBITS
+ mk obj/kern/kernel.img
make[1]: Leaving directory '/home/blue9057/jos'
qemu-system-i386 -nographic -drive file=obj/kern/kernel.img,index=0,media=disk,format=raw -serial mon:stdio -gdb tcp::26000 -D qemu.log
6828 decimal is 15254 octal!
Physical memory: 131072K available, base = 640K, extended = 130432K
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
Incoming TRAP frame at 0xeffffbc
Incoming TRAP frame at 0xeffffbc
TRAP frame at 0xf01c0000
  edi  0x00000000
  esi  0x00000000
  ebp  0xeebfdfd0
  oesp 0xefffffdc
  ebx  0x00000000
  edx  0x00000000
  ecx  0x00000000
  eax  0x00000001
  es   0x----0023
  ds   0x----0023
  trap 0x00000000 Divide error
  err  0x00000000
  eip  0x0080004e
  cs   0x----001b
  flag 0x0000001a
  esp  0xeebfdfb8
  ss   0x----0023
[00001000] free env 00001000
Destroyed the only environment - nothing more to do!
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
```

How to debug USER program?

- Set a breakpoint at env_pop_tf
  - b env_pop_tf
  - c
- Then, trace it with ‘si’ upto iret
  - After iret, user execution starts!
- How to know about the semantics of the user program?
  - Open obj/user/program_name.asm