Os lab3: User Environment

陈炜栋 11300240057

November 24, 2013

Contents

1 Introduction

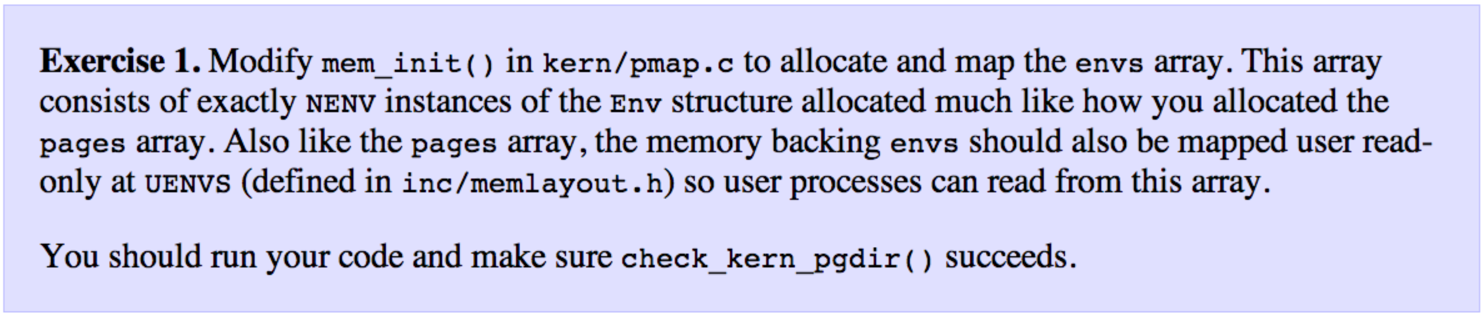
Lab3的任务是建立基本的用户态进程环境，并时间简单的进程调度。与此同时，也提供一些简单的系统函数。

2 User Environments and Exception Handling

2.1 Environment State

在in/env.h下有struct Env的定义。

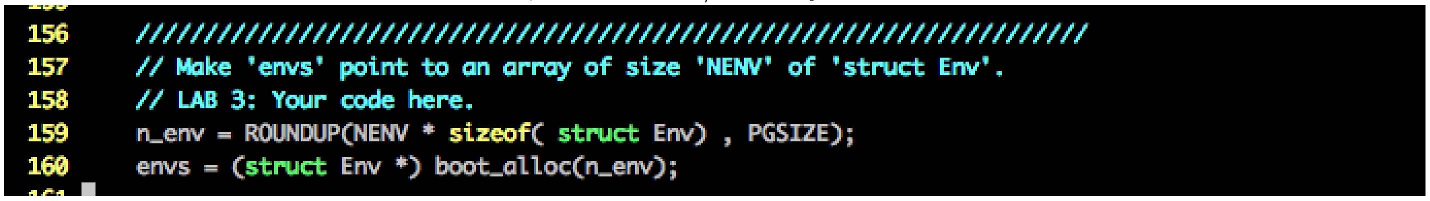
2.2 Allocating the Environment Array



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.

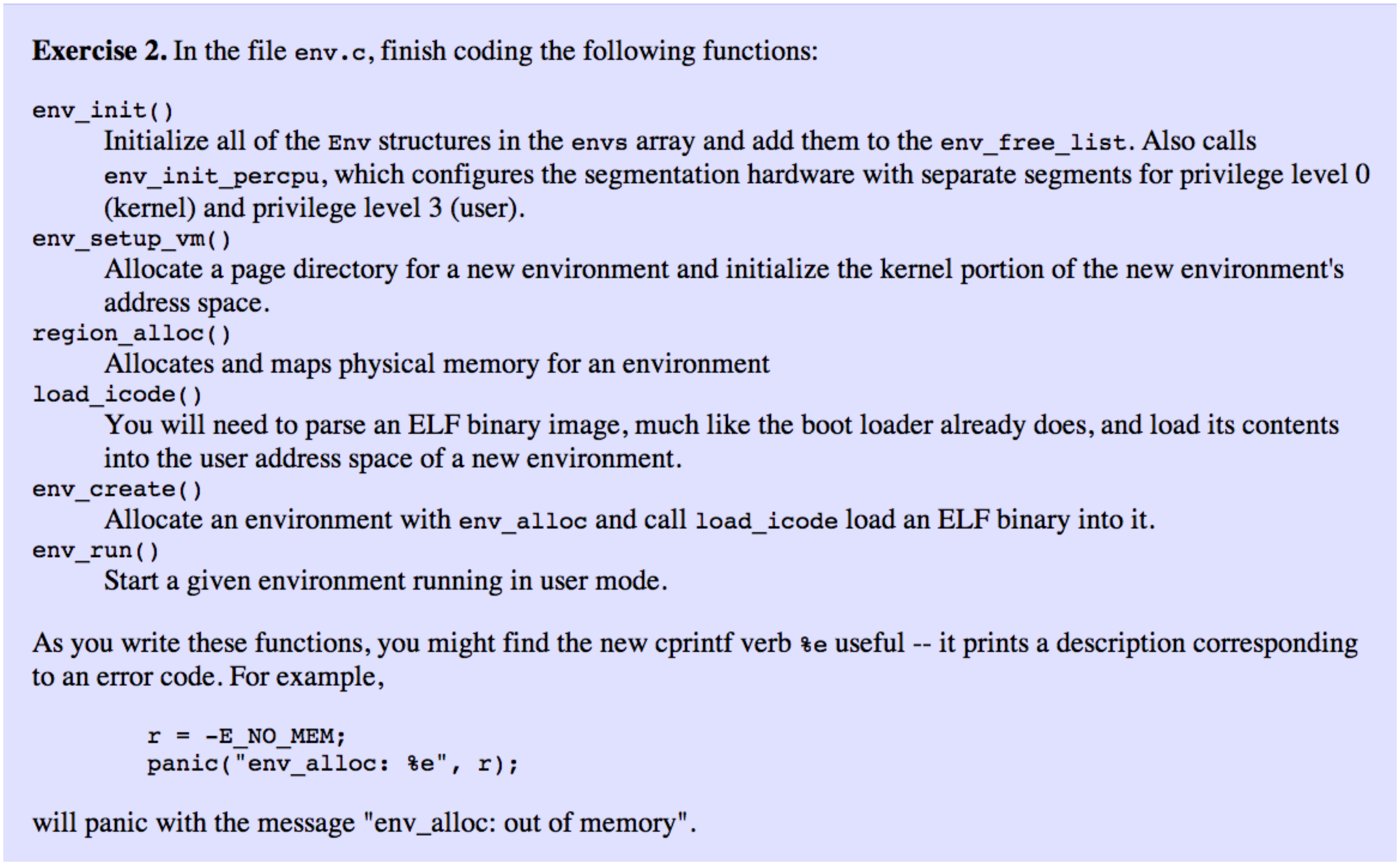
You should run your code and make sure `check_kern_pgdir()` succeeds.

- 补全mem_init()函数，在其中分配对应的页给envs。
- 在页表中为其作映射，映射到UENVS,这个对应inc/memlayout.h中的注释中。



```
//////////////////////////////////////////////////////////////////////
// Make 'envs' point to an array of size 'NENV' of 'struct Env'.
// LAB 3: Your code here.
n_env = ROUNDUP(NENV * sizeof( struct Env) , PGSIZE);
envs = (struct Env *) boot_alloc(n_env);
```

2.3 Creating and Running Environments



Exercise 2. In the file `env.c`, finish coding the following functions:

`env_init()`
: Initialize all of the `Env` structures in the `envs` array and add them to the `env_free_list`. Also calls `env_init_percpu`, which configures the segmentation hardware with separate segments for privilege level 0 (kernel) and privilege level 3 (user).

`env_setup_vm()`
: Allocate a page directory for a new environment and initialize the kernel portion of the new environment's address space.

`region_alloc()`
: Allocates and maps physical memory for an environment

`load_icode()`
: You will need to parse an ELF binary image, much like the boot loader already does, and load its contents into the user address space of a new environment.

`env_create()`
: Allocate an environment with `env_alloc` and call `load_icode` load an ELF binary into it.

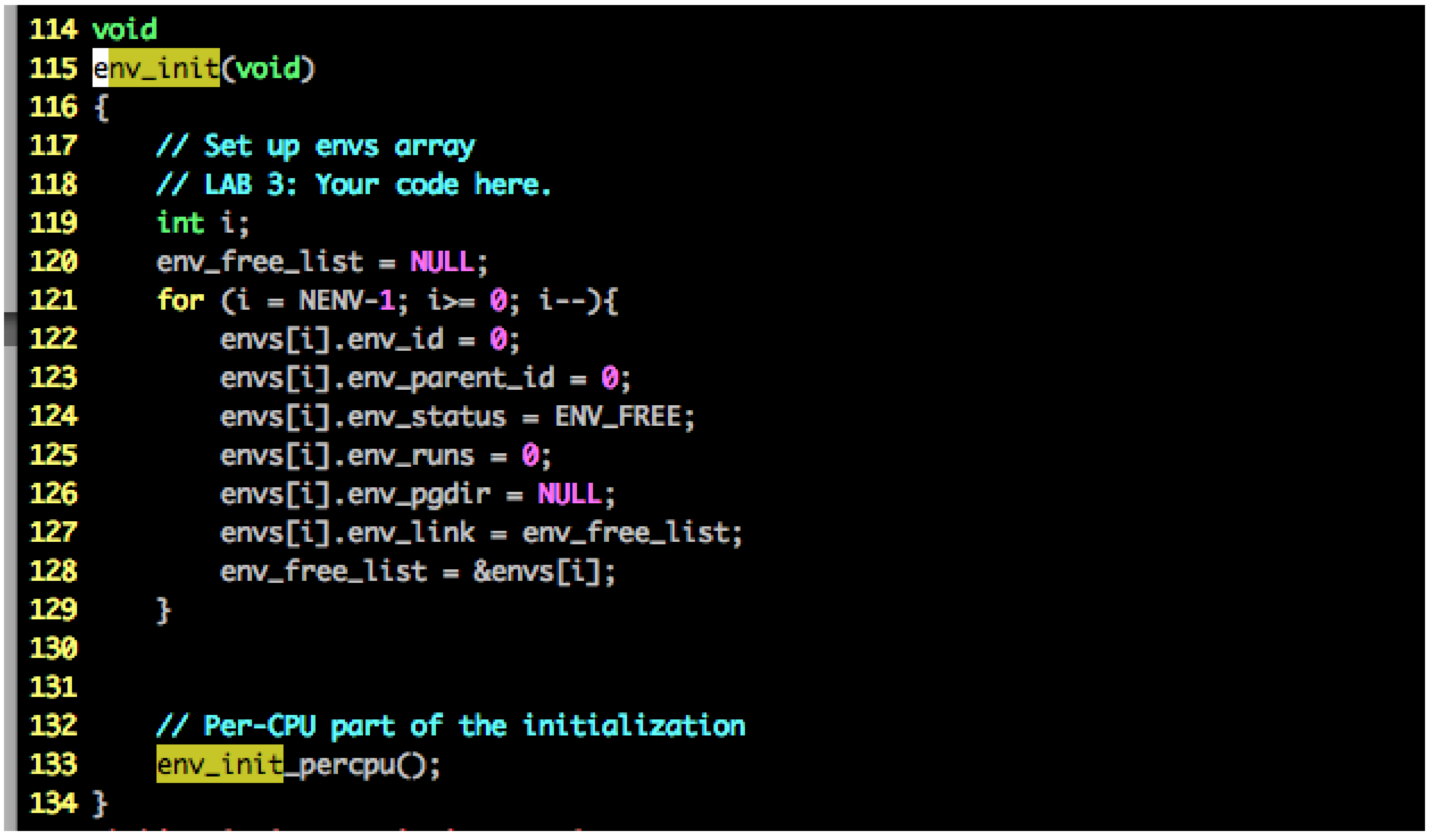
`env_run()`
: Start a given environment running in user mode.

As you write these functions, you might find the new cprintf verb `%e` useful -- it prints a description corresponding to an error code. For example,

```
	r = -E_NO_MEM;
	panic("env_alloc: %e", r);
```

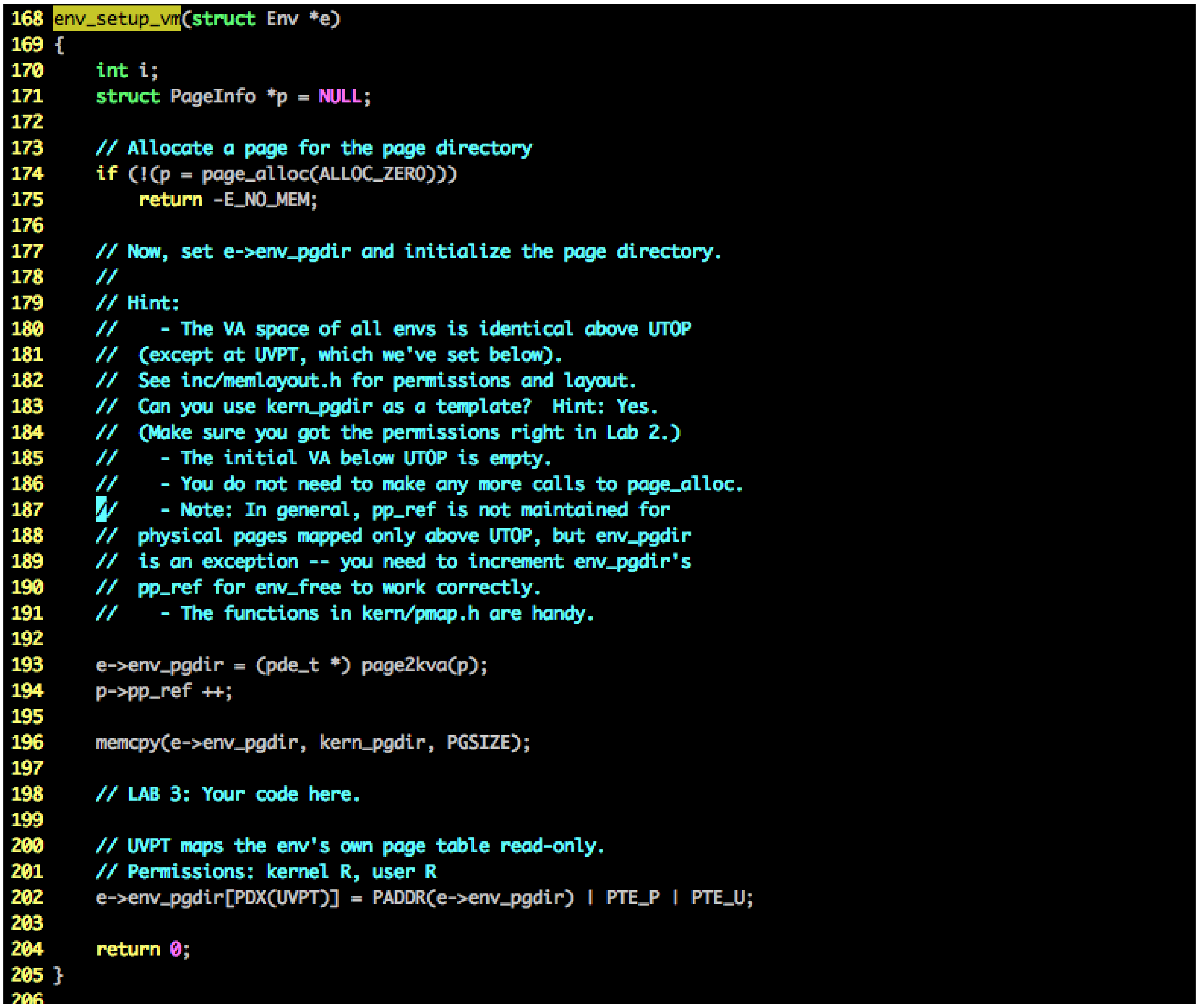
will panic with the message "env_alloc: out of memory".

- 接下来就是完成env_init()和env_create(),env_run()等函数，使得JOS中内嵌的二进制镜像得以运行在虚拟环境中。
- env_init()初始化free的enviorment链表，这里我把所有的属性都初始化了一遍。按照注释，为了保证分配的地址和数组的地址是顺序的，i逆序枚举。



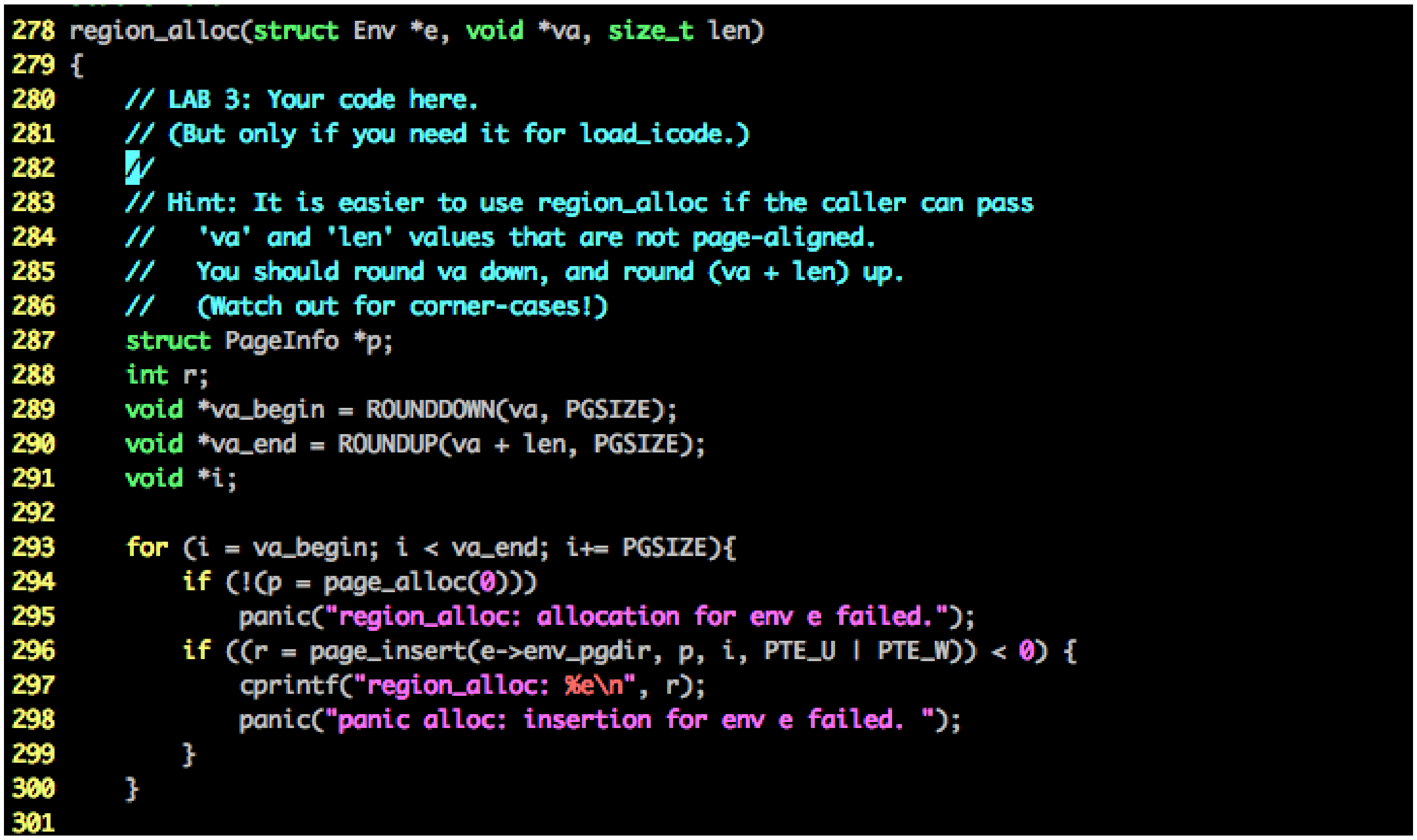
```c
void
env_init(void)
{
	// Set up envs array
	// LAB 3: Your code here.
	int i;
	env_free_list = NULL;
	for (i = NENV-1; i>= 0; i--){
		envs[i].env_id = 0;
		envs[i].env_parent_id = 0;
		envs[i].env_status = ENV_FREE;
		envs[i].env_runs = 0;
		envs[i].env_pgdir = NULL;
		envs[i].env_link = env_free_list;
		env_free_list = &envs[i];
	}


	// Per-CPU part of the initialization
	env_init_percpu();
}
```

- env_setup_vm简单的为新建的struct Env对象进行初始化，分配一个物理页给其页目录e-¿env_pgdir,同时复制kern_pgdir在 UTOP以上的内容。实际操作起来其实就是按照Hint所说，以kern_pgdir作为模板复制一份。



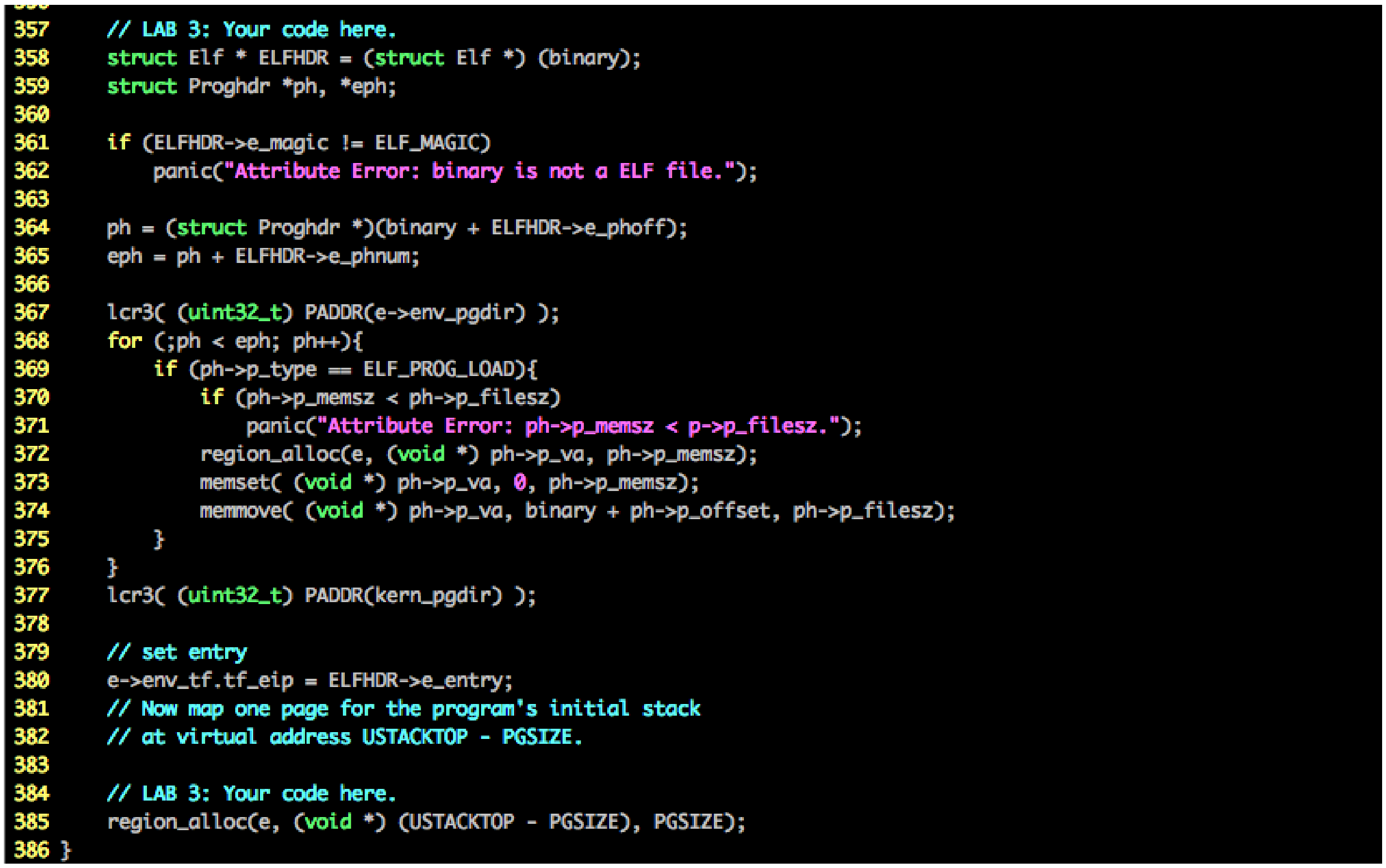
```c
env_setup_vm(struct Env *e)
{
    int i;
    struct PageInfo *p = NULL;

    // Allocate a page for the page directory
    if (!(p = page_alloc(ALLOC_ZERO)))
        return -E_NO_MEM;

    // Now, set e->env_pgdir and initialize the page directory.
    //
    // Hint:
    //    - The VA space of all envs is identical above UTOP
    //  (except at UVPT, which we've set below).
    //  See inc/memlayout.h for permissions and layout.
    //  Can you use kern_pgdir as a template?  Hint: Yes.
    //  (Make sure you got the permissions right in Lab 2.)
    //    - The initial VA below UTOP is empty.
    //    - You do not need to make any more calls to page_alloc.
    //    - Note: In general, pp_ref is not maintained for
    //  physical pages mapped only above UTOP, but env_pgdir
    //  is an exception -- you need to increment env_pgdir's
    //  pp_ref for env_free to work correctly.
    //    - The functions in kern/pmap.h are handy.

    e->env_pgdir = (pde_t *) page2kva(p);
    p->pp_ref ++;

    memcpy(e->env_pgdir, kern_pgdir, PGSIZE);

    // LAB 3: Your code here.

    // 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;

    return 0;
}
```

- region_alloc 给Env 的[va,va+len]分配页。和上个lab的kern_pgdir 类似.不过我们现在是对一个环境来说。



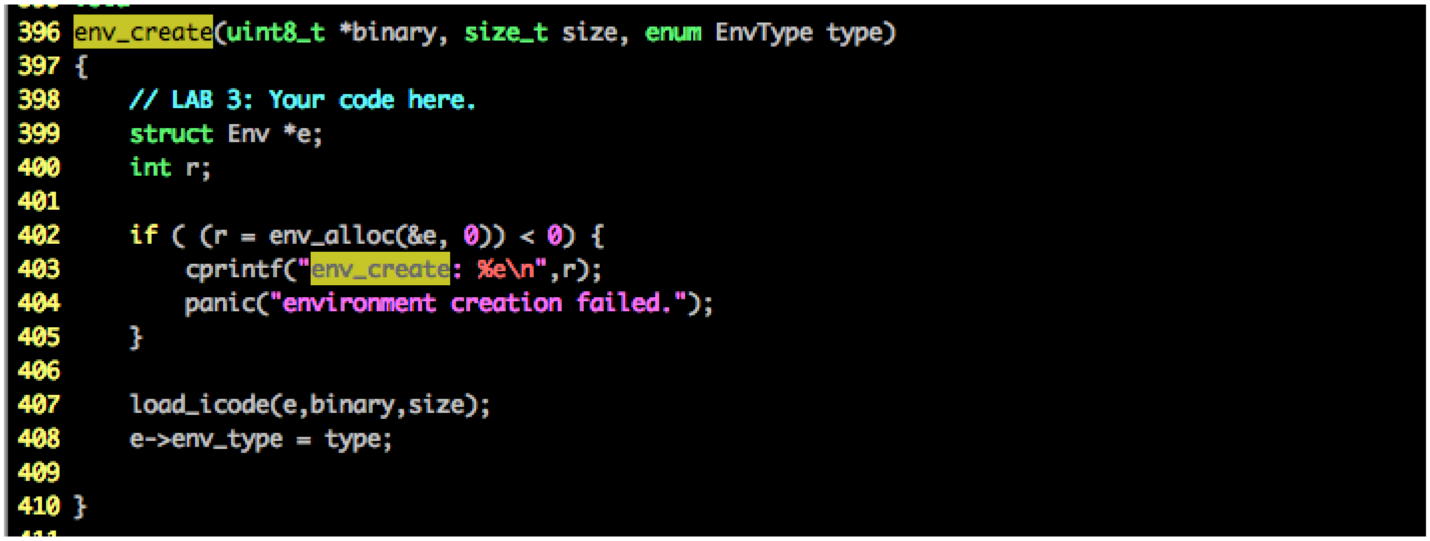
```
region_alloc(struct Env *e, void *va, size_t len)
{
	// LAB 3: Your code here.
	// (But only if you need it for load_icode.)
	//
	// Hint: It is easier to use region_alloc if the caller can pass
	//   'va' and 'len' values that are not page-aligned.
	//   You should round va down, and round (va + len) up.
	//   (Watch out for corner-cases!)
	struct PageInfo *p;
	int r;
	void *va_begin = ROUNDDOWN(va, PGSIZE);
	void *va_end = ROUNDUP(va + len, PGSIZE);
	void *i;

	for (i = va_begin; i < va_end; i+= PGSIZE){
		if (!(p = page_alloc(0)))
			panic("region_alloc: allocation for env e failed.");
		if ((r = page_insert(e->env_pgdir, p, i, PTE_U | PTE_W)) < 0) {
			cprintf("region_alloc: %e\n", r);
			panic("panic alloc: insertion for env e failed. ");
		}
	}

```

- load_icode() 和 boot/main.c 里的bootmain() 差不多，对二进制文件ELF头进行解析，将其load到内存中。为了地址转化，cr3需要先转成 env的页目录，载入完了。再转化回来。



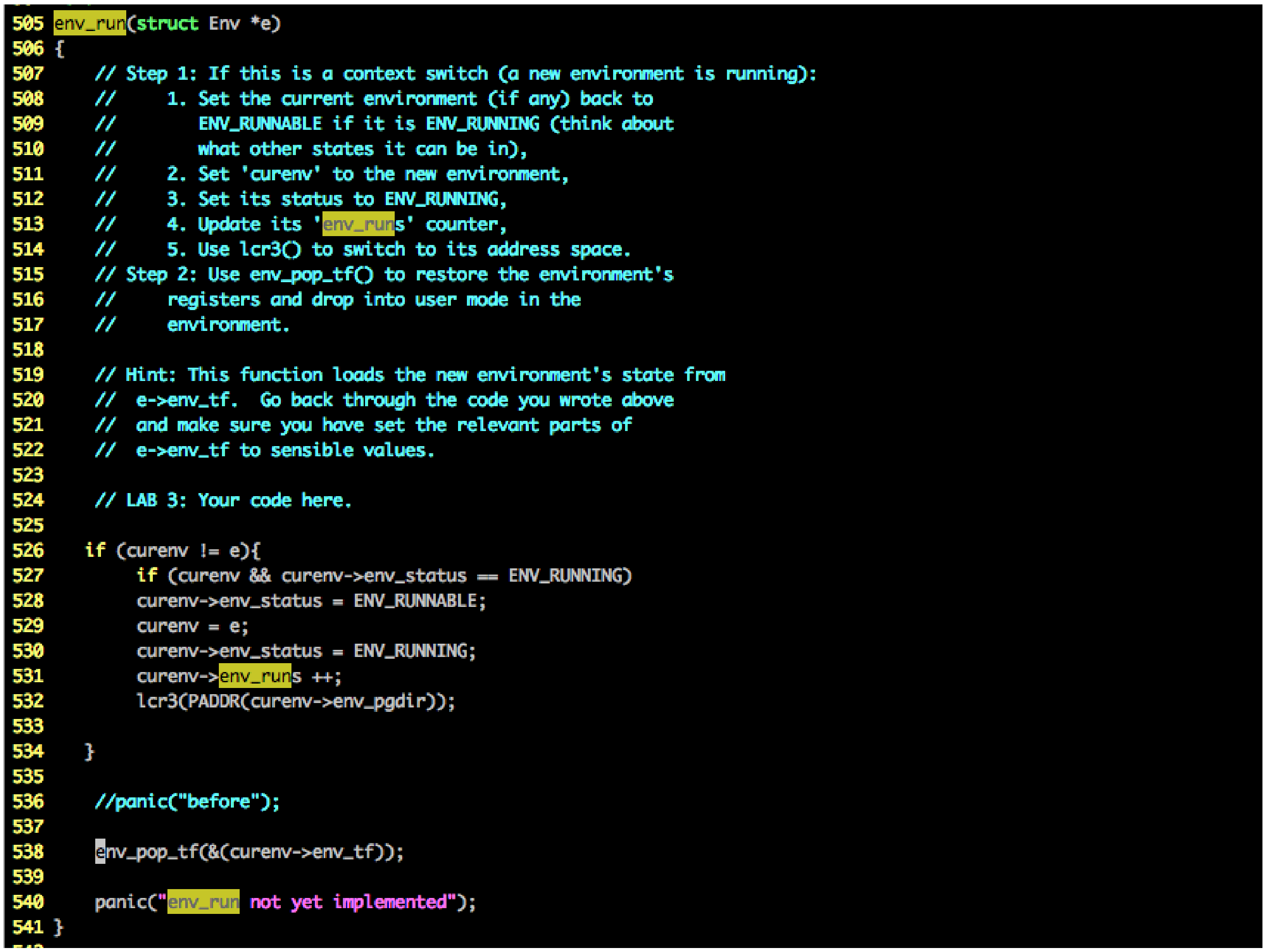
```
    // LAB 3: Your code here.
    struct Elf * ELFHDR = (struct Elf *) (binary);
    struct Proghdr *ph, *eph;

    if (ELFHDR->e_magic != ELF_MAGIC)
        panic("Attribute Error: binary is not a ELF file.");

    ph = (struct Proghdr *)(binary + ELFHDR->e_phoff);
    eph = ph + ELFHDR->e_phnum;

    lcr3( (uint32_t) PADDR(e->env_pgdir) );
    for (;ph < eph; ph++){
        if (ph->p_type == ELF_PROG_LOAD){
            if (ph->p_memsz < ph->p_filesz)
                panic("Attribute Error: ph->p_memsz < p->p_filesz.");
            region_alloc(e, (void *) ph->p_va, ph->p_memsz);
            memset( (void *) ph->p_va, 0, ph->p_memsz);
            memmove( (void *) ph->p_va, binary + ph->p_offset, ph->p_filesz);
        }
    }
    lcr3( (uint32_t) PADDR(kern_pgdir) );

    // set entry
    e->env_tf.tf_eip = ELFHDR->e_entry;
    // Now map one page for the program's initial stack
    // at virtual address USTACKTOP - PGSIZE.

    // LAB 3: Your code here.
    region_alloc(e, (void *) (USTACKTOP - PGSIZE), PGSIZE);
}
```

- env_create调用已经存在的env_alloc穿件并且初始化一个新的 env,然后调用load_icode将二进制镜像载入。



```
env_create(uint8_t *binary, size_t size, enum EnvType type)
{
    // LAB 3: Your code here.
    struct Env *e;
    int r;

    if ( (r = env_alloc(&e, 0)) < 0) {
        cprintf("env_create: %e\n",r);
        panic("environment creation failed.");
    }

    load_icode(e,binary,size);
    e->env_type = type;

}
```

- env_run()用于启动一个进程，如果当前已经有一个环境在运行，则需改变状态。



```
505 env_run(struct Env *e)
506 {
507     // Step 1: If this is a context switch (a new environment is running):
508     //     1. Set the current environment (if any) back to
509     //        ENV_RUNNABLE if it is ENV_RUNNING (think about
510     //        what other states it can be in),
511     //     2. Set 'curenv' to the new environment,
512     //     3. Set its status to ENV_RUNNING,
513     //     4. Update its 'env_runs' counter,
514     //     5. Use lcr3() to switch to its address space.
515     // Step 2: Use env_pop_tf() to restore the environment's
516     //     registers and drop into user mode in the
517     //     environment.
518
519     // Hint: This function loads the new environment's state from
520     //  e->env_tf.  Go back through the code you wrote above
521     //  and make sure you have set the relevant parts of
522     //  e->env_tf to sensible values.
523
524     // LAB 3: Your code here.
525
526    if (curenv != e){
527         if (curenv && curenv->env_status == ENV_RUNNING)
528         curenv->env_status = ENV_RUNNABLE;
529         curenv = e;
530         curenv->env_status = ENV_RUNNING;
531         curenv->env_runs ++;
532         lcr3(PADDR(curenv->env_pgdir));
533
534    }
535
536     //panic("before");
537
538     env_pop_tf(&(curenv->env_tf));
539
540     panic("env_run not yet implemented");
541 }
```

至此exercise 2 就完成了。使用make qemu，应该会看到Triple fault而出现的register dump.

但是还需证明我们是正确流程导致的该错误。为了验证的确是正确流程产生改错误。在上面env_run的代码536 行插入一个panic 证明了我们是在执行env_pop_tf的时候才产生的Triple Fault

2.4 Handling Interrupts and Exceptions

Exercise 3. Read Chapter 9, Exceptions and Interrupts in the 80386 Programmer's Manual (or Chapter 5 of the IA-32 Developer's Manual), if you haven't already.

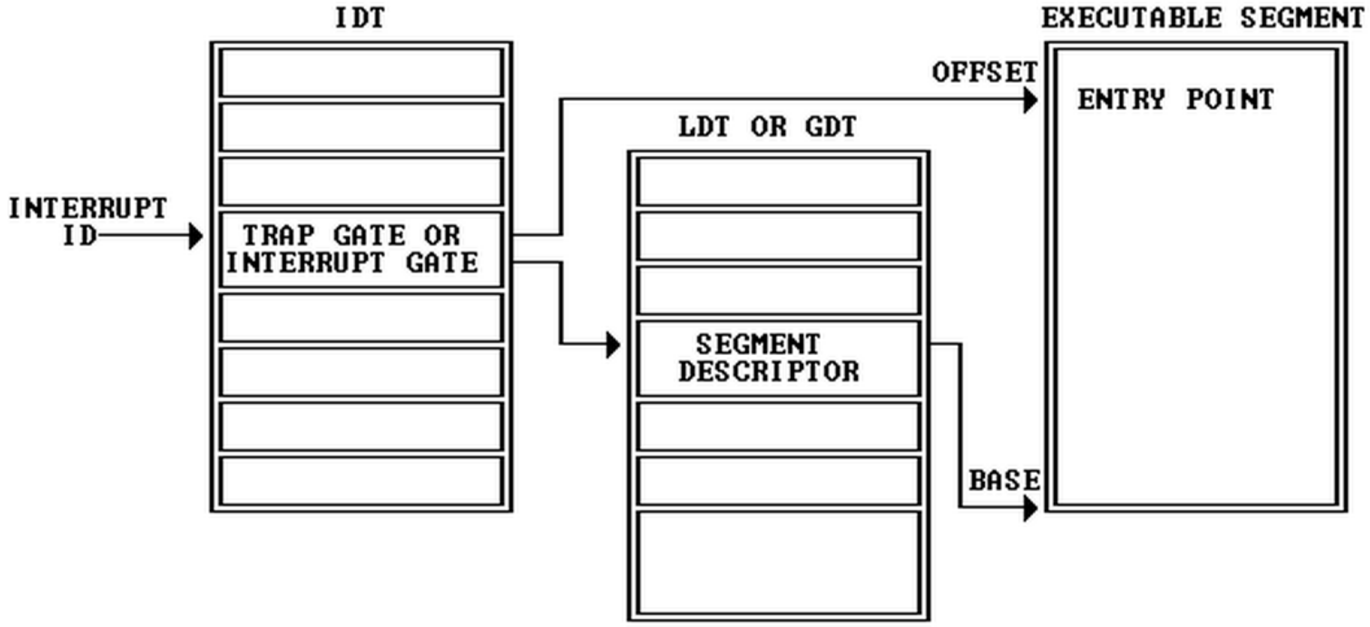
看了前面的文档，后面的手册太长了Orz。

下面是我的一点阅读记录:

- interrupt 和 exception 是特殊的流程空格那只转化，可以想象成另外一种call，改变正常的程序流程来处理外部时间或者报错，处理异常情况。

- 两者的区别在于 Interrupt用于处理异步的硬件终端，Exception 用于处理指令执行中的异常，如除零错误。
- Exception 又分为faults , traps, aborts. 三者都是软件终端，但是一次严重程度递增。fault是可恢复的。abort则是直接终端
- IDT(Interrupt Descriptor Table) GDT LDT 三者都是8字节的数组。
- IDT又分为三类Task gates, Interrupt gates, Trap gates.其中的区别比较复杂。。Interrupt gates 和Trap的对于IF的区别接下来描述。
- TF(Trap Flag)， interrupt h和 trap 均会导致重置TF，从而抑制挑食，IRET执行后恢复。
- IF(Interrupt-enable Flag) 使得 interrupt 不被打扰，IRET恢复。其中通过interrupt gates的会重置IF，从而达到屏蔽终端的效果。Trap Gate则不然。

Figure 9-4. Interrupt Vectoring for Procedures



2.5 Basics of Protected Control Transfer

描述了IDT和TSS的具体构造。大致他们都是为了抛出给你interrupt/exception的正确处理，通过user mode与 Kernel mode 的切换及环境状态的保存到栈中以隔开不同的环境。

2.6 Types of Exceptions and Interrupts

0-31用于处理器抛出exception。之后32之后用户外部设备或者用户程序通过int指令抛出，其中48

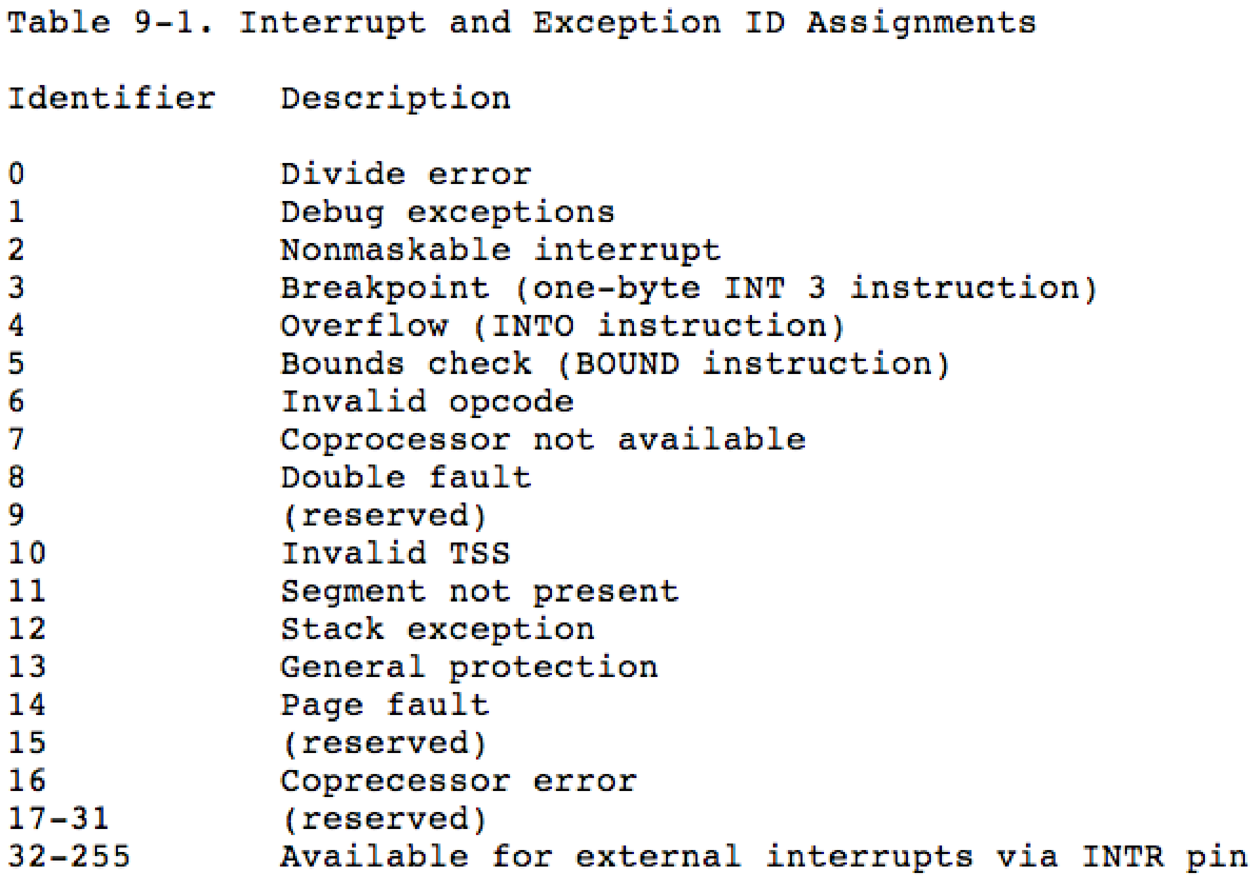


Table 9-1. Interrupt and Exception ID Assignments

Identifier	Description
0	Divide error
1	Debug exceptions
2	Nonmaskable interrupt
3	Breakpoint (one-byte INT 3 instruction)
4	Overflow (INTO instruction)
5	Bounds check (BOUND instruction)
6	Invalid opcode
7	Coprocessor not available
8	Double fault
9	(reserved)
10	Invalid TSS
11	Segment not present
12	Stack exception
13	General protection
14	Page fault
15	(reserved)
16	Coprecessor error
17-31	(reserved)
32-255	Available for external interrupts via INTR pin

是system_call

2.7 An Example

讲了一个除零异常和一个缺页异常的处理，栈操作。有助于理解张弛报告上的代码。

2.8 Nested exceptions and Interrupts

当紧车工已经在Kernel mode底下，又引发了Exception/interrupt,那么就算是一个nested Excetion/interrupts.这个时候不需要切换的内核栈，页不需要保存SS和ESP寄存器

2.9 Setting Up the IDT

Exercise 4. Edit `trapentry.S` and `trap.c` and implement the features described above. The macros `TRAPHANDLER` and `TRAPHANDLER_NOEC` in `trapentry.S` should help you, as well as the T_* defines in `inc/trap.h`. You will need to add an entry point in `trapentry.S` (using those macros) for each trap defined in `inc/trap.h`, and you'll have to provide `_alltraps` which the `TRAPHANDLER` macros refer to. You will also need to modify `trap_init()` to initialize the `idt` to point to each of these entry points defined in `trapentry.S`; the `SETGATE` macro will be helpful here.

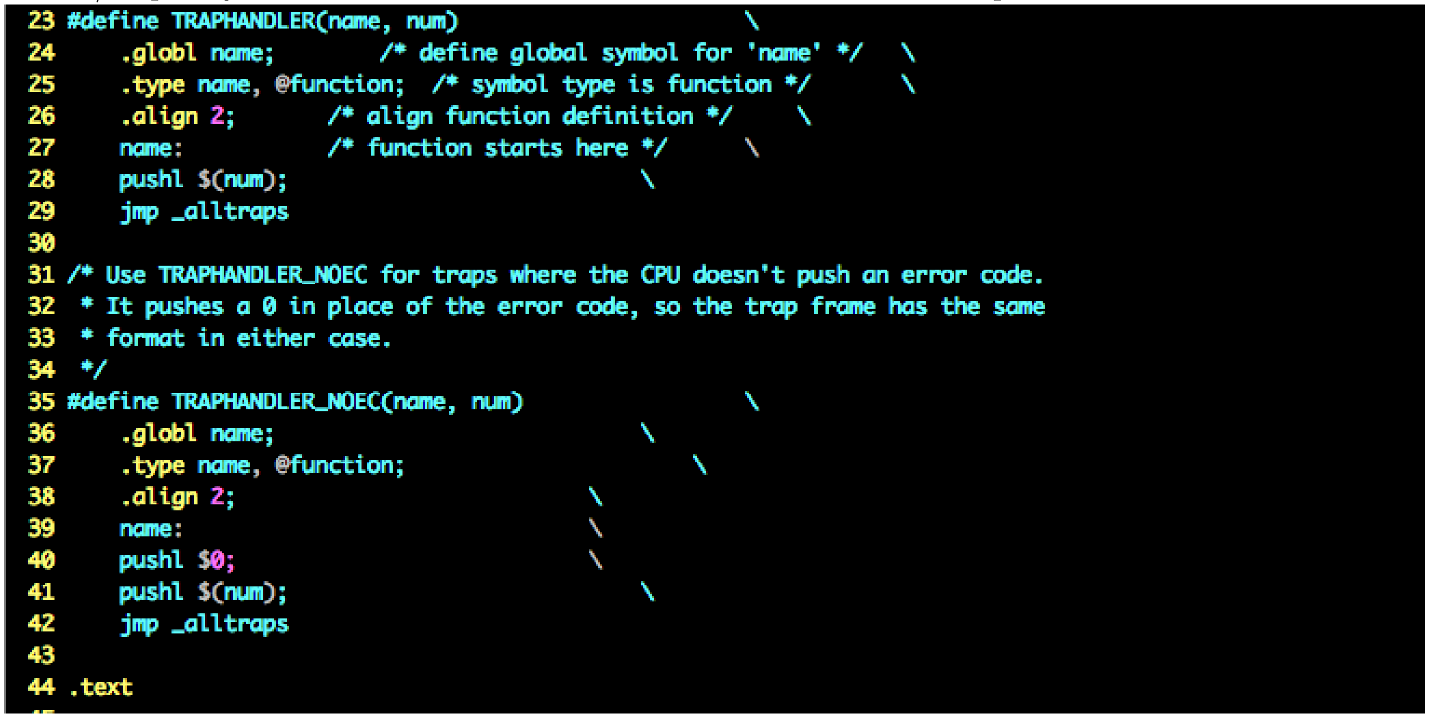
Your `_alltraps` should:

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?)

Consider using the `pushal` instruction; it fits nicely with the layout of the `struct Trapframe`.

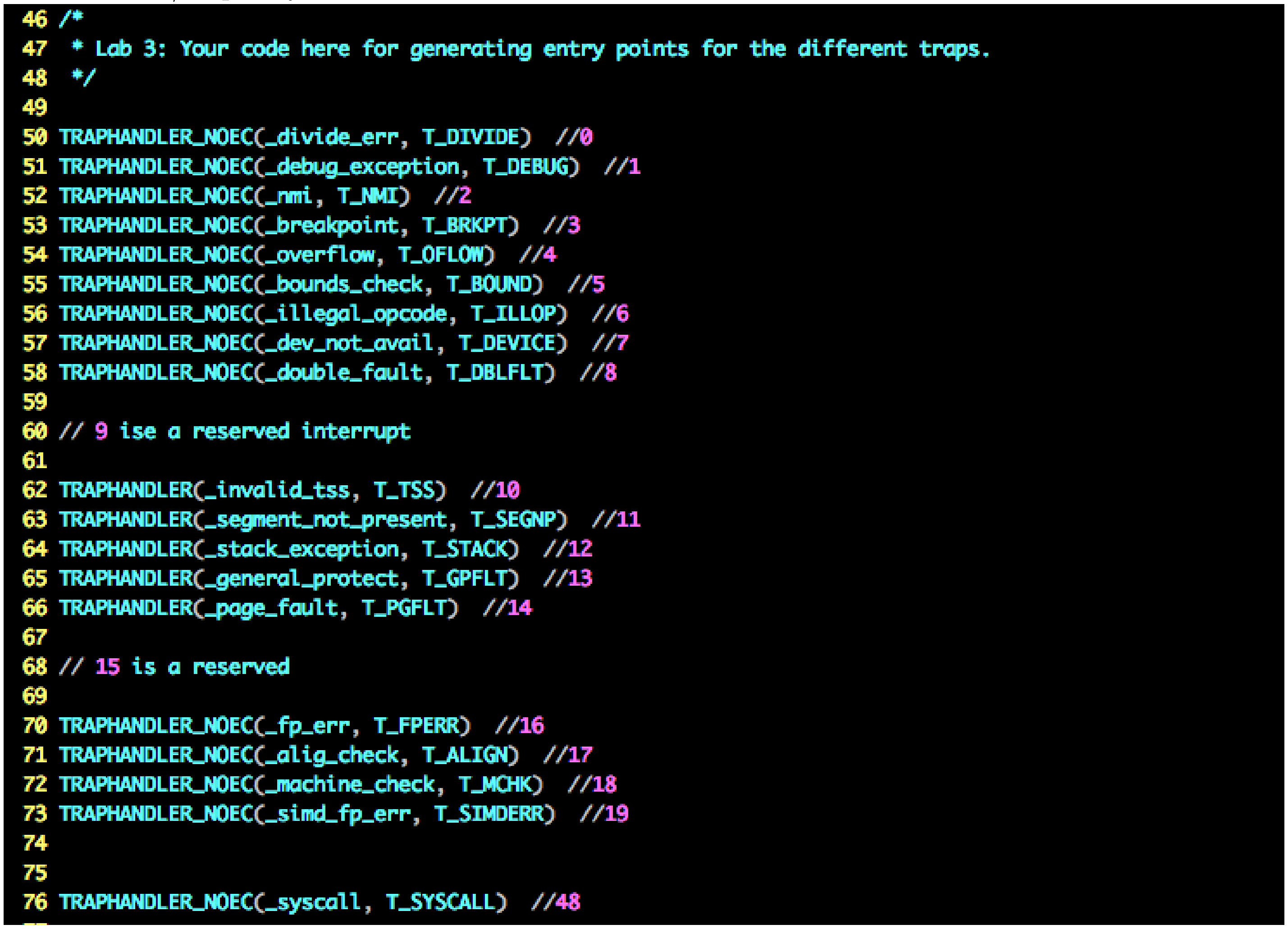
Test your trap handling code using some of the test programs in the `user` directory that cause exceptions before making any system calls, such as `user/divzero`. You should be able to get **`make grade`** to succeed on the `divzero`, `softint`, and `badsegment` tests at this point.

- 在kern/trapentry.S中已经定义好的两个Handler的宏，可以用于定义Trap函数



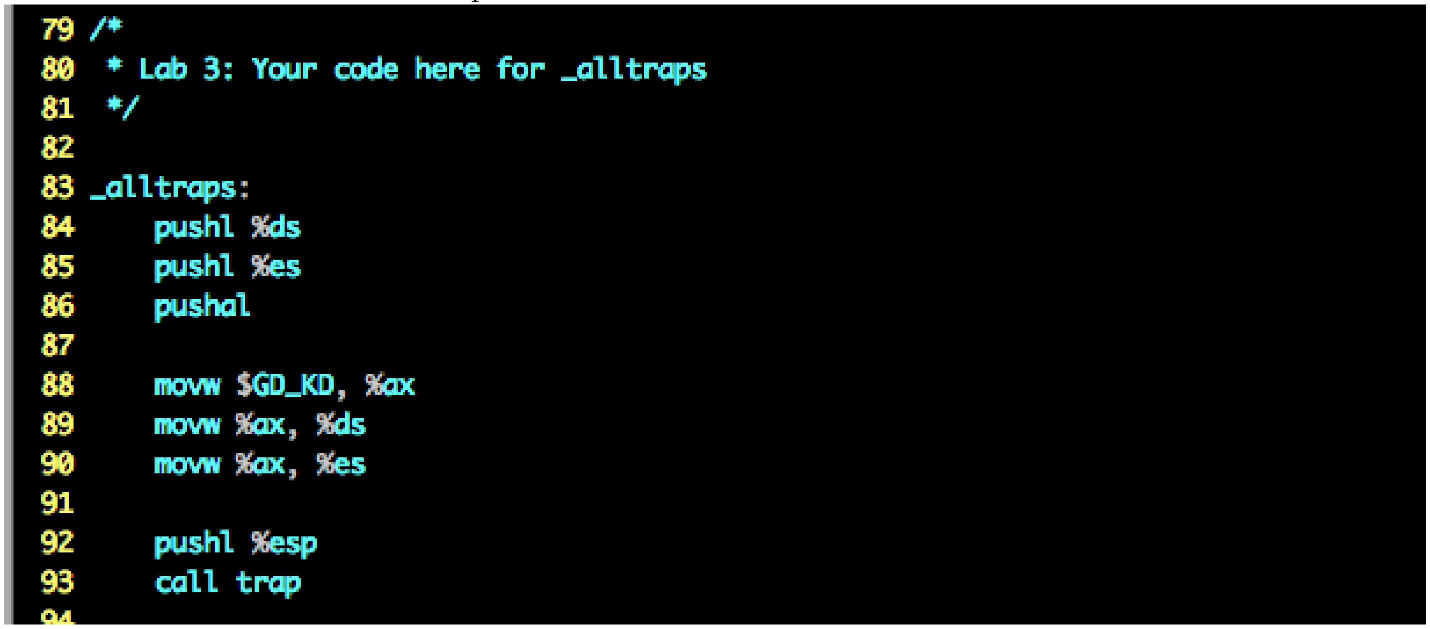
```
#define TRAPHANDLER(name, num)                                          \
    .globl name;        /* define global symbol for 'name' */   \
    .type name, @function;  /* symbol type is function */       \
    .align 2;       /* align function definition */     \
    name:           /* function starts here */      \
    pushl $(num);                                   \
    jmp _alltraps

/* Use TRAPHANDLER_NOEC for traps where the CPU doesn't push an error code.
 * It pushes a 0 in place of the error code, so the trap frame has the same
 * format in either case.
 */
#define TRAPHANDLER_NOEC(name, num)                                     \
    .globl name;                                    \
    .type name, @function;                              \
    .align 2;                               \
    name:                                   \
    pushl $0;                               \
    pushl $(num);                                   \
    jmp _alltraps

.text
```

- 于是，在kern/trapentry.S 底下，我们利用上面两条宏定义了对应的中断函数。



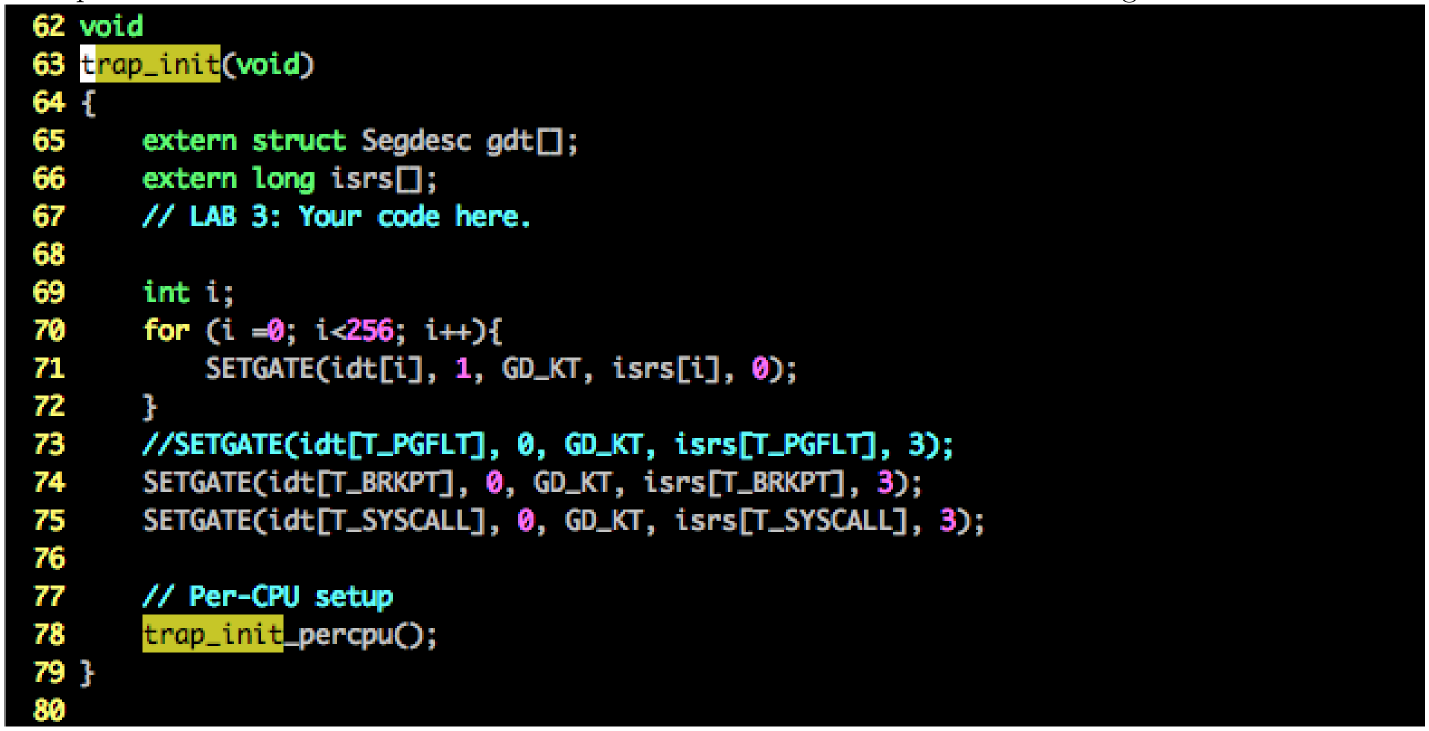
```
/*
 * Lab 3: Your code here for generating entry points for the different traps.
 */

TRAPHANDLER_NOEC(_divide_err, T_DIVIDE)  //0
TRAPHANDLER_NOEC(_debug_exception, T_DEBUG)  //1
TRAPHANDLER_NOEC(_nmi, T_NMI)  //2
TRAPHANDLER_NOEC(_breakpoint, T_BRKPT)  //3
TRAPHANDLER_NOEC(_overflow, T_OFLOW)  //4
TRAPHANDLER_NOEC(_bounds_check, T_BOUND)  //5
TRAPHANDLER_NOEC(_illegal_opcode, T_ILLOP)  //6
TRAPHANDLER_NOEC(_dev_not_avail, T_DEVICE)  //7
TRAPHANDLER_NOEC(_double_fault, T_DBLFLT)  //8

// 9 ise a reserved interrupt

TRAPHANDLER(_invalid_tss, T_TSS)  //10
TRAPHANDLER(_segment_not_present, T_SEGNP)  //11
TRAPHANDLER(_stack_exception, T_STACK)  //12
TRAPHANDLER(_general_protect, T_GPFLT)  //13
TRAPHANDLER(_page_fault, T_PGFLT)  //14

// 15 is a reserved

TRAPHANDLER_NOEC(_fp_err, T_FPERR)  //16
TRAPHANDLER_NOEC(_alig_check, T_ALIGN)  //17
TRAPHANDLER_NOEC(_machine_check, T_MCHK)  //18
TRAPHANDLER_NOEC(_simd_fp_err, T_SIMDERR)  //19


TRAPHANDLER_NOEC(_syscall, T_SYSCALL)  //48
```

- 以及在宏中出现的未实现的_alltraps.非常底层的东西了:



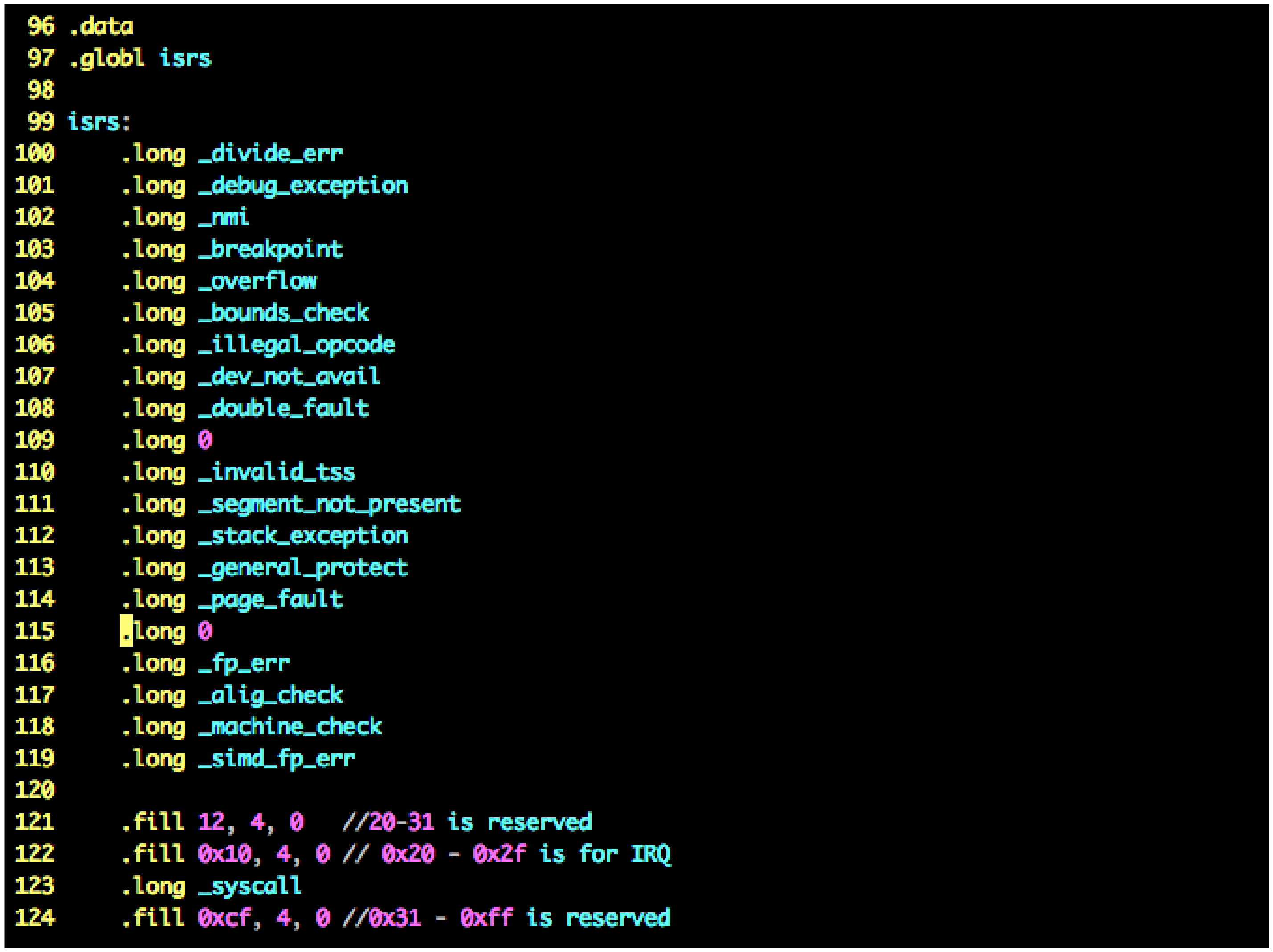
```
/*
 * Lab 3: Your code here for _alltraps
 */

_alltraps:
    pushl %ds
    pushl %es
    pushal

    movw $GD_KD, %ax
    movw %ax, %ds
    movw %ax, %es

    pushl %esp
    call trap
```

- 在trap_inti中将，IDT表与handler函数联系起来。这里我定义了isrs，是把challange一起做了。



```
void
trap_init(void)
{
    extern struct Segdesc gdt[];
    extern long isrs[];
    // LAB 3: Your code here.

    int i;
    for (i =0; i<256; i++){
        SETGATE(idt[i], 1, GD_KT, isrs[i], 0);
    }
    //SETGATE(idt[T_PGFLT], 0, GD_KT, isrs[T_PGFLT], 3);
    SETGATE(idt[T_BRKPT], 0, GD_KT, isrs[T_BRKPT], 3);
    SETGATE(idt[T_SYSCALL], 0, GD_KT, isrs[T_SYSCALL], 3);

    // Per-CPU setup
    trap_init_percpu();
}

```

- isrs定义在trapentry.S的最后面。



```
.data
.globl isrs

isrs:
    .long _divide_err
    .long _debug_exception
    .long _nmi
    .long _breakpoint
    .long _overflow
    .long _bounds_check
    .long _illegal_opcode
    .long _dev_not_avail
    .long _double_fault
    .long 0
    .long _invalid_tss
    .long _segment_not_present
    .long _stack_exception
    .long _general_protect
    .long _page_fault
    .long 0
    .long _fp_err
    .long _alig_check
    .long _machine_check
    .long _simd_fp_err

    .fill 12, 4, 0   //20-31 is reserved
    .fill 0x10, 4, 0 // 0x20 - 0x2f is for IRQ
    .long _syscall
    .fill 0xcf, 4, 0 //0x31 - 0xff is reserved
```

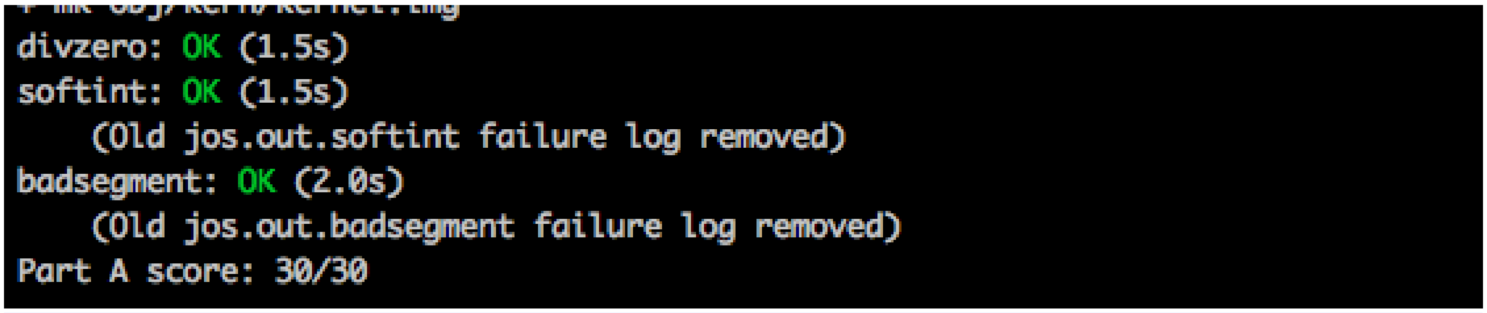
Challenge! You probably have a lot of very similar code right now, between the lists of `TRAPHANDLER` in `trapentry.S` and their installations in `trap.c`. Clean this up. Change the macros in `trapentry.S` to automatically generate a table for `trap.c` to use. Note that you can switch between laying down code and data in the assembler by using the directives `.text` and `.data`.

至此我们完成了Challenge1所给的要求。

整个流程是这样的：

建立中断流程响应机制后，一个中断被送到_alltraps,然后调用kern/trap.c中的trap(),在trap_dispatch()中把当前环境destroy掉，再通过env_run来把当前的新环境运行起来。

至此可以通过Part A 的测试：



```
divzero: OK (1.5s)
softint: OK (1.5s)
    (Old jos.out.softint failure log removed)
badsegment: OK (2.0s)
    (Old jos.out.badsegment failure log removed)
Part A score: 30/30
```

Questions

Answer the following questions in your `answers-lab3.txt`:

1. 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?)
2. 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)?

1. 现在JOS的终端处理程序在真正处理之前要将中断号放入内核栈以组织成Trapframe的结构，但是如果所有终端都跳到同一个处理程序，那么就无法区分哪个中断调用进来的，也就无法正确设置他们的中断号了。

2. 现在我们在中断向量里设置的14号Page fault的调用权限是0，也就是说只能内核抛出，所以在softint中用int指令调用肯定产生GP Fault权限错误。

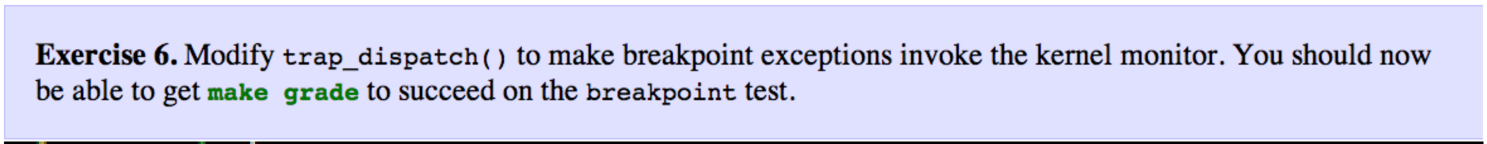
3 Page Faults, Breakpoints Exception, and System Calls

3.1 Handling Page Faults

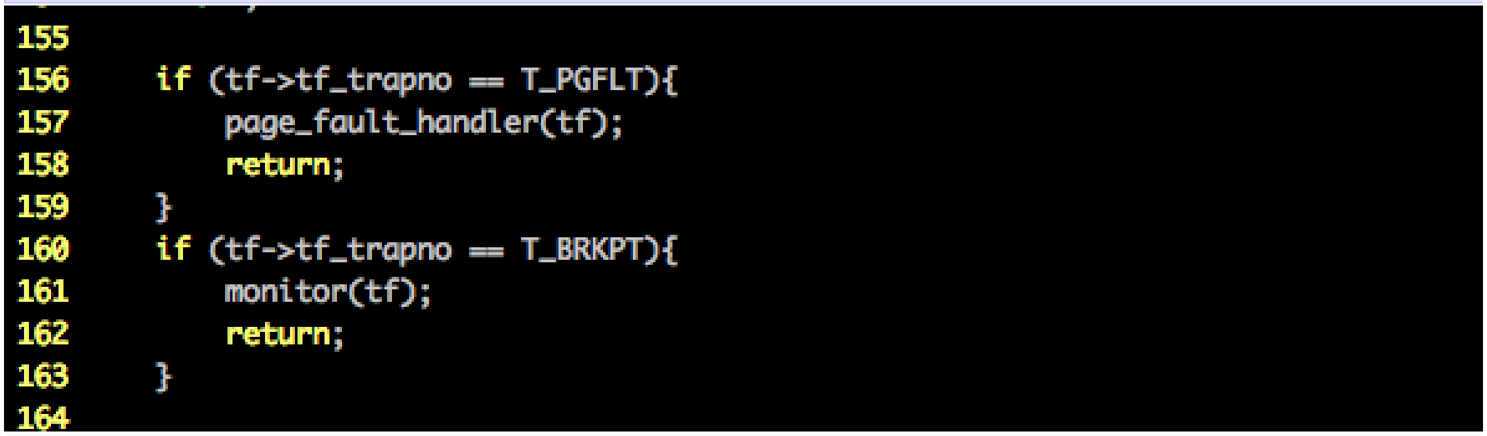
Exercise 5. Modify `trap_dispatch()` to dispatch page fault exceptions to `page_fault_handler()`. You should now be able to get **`make grade`** to succeed on the `faultread`, `faultreadkernel`, `faultwrite`, and `faultwritekernel` tests. If any of them don't work, figure out why and fix them. Remember that you can boot JOS into a particular user program using **`make run-x`** or **`make run-x-nox`**.

exercise 5 和 6 类似，都只要在trap_dispatch中添加一段代码就好了。

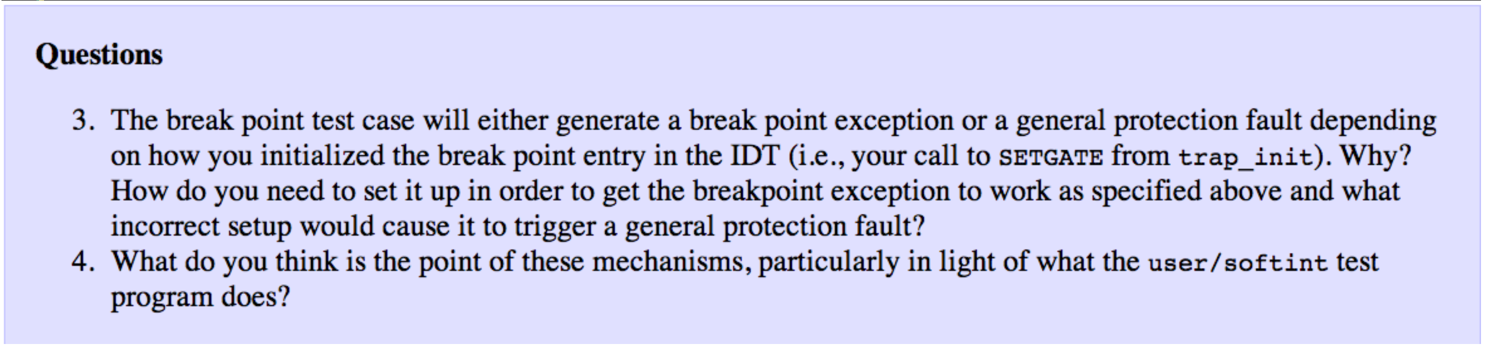
3.2 The Breakpoint Exception



Exercise 6. Modify `trap_dispatch()` to make breakpoint exceptions invoke the kernel monitor. You should now be able to get `make grade` to succeed on the `breakpoint` test.



```

    if (tf->tf_trapno == T_PGFLT){
        page_fault_handler(tf);
        return;
    }
    if (tf->tf_trapno == T_BRKPT){
        monitor(tf);
        return;
    }

```



Questions

3. 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`). Why? How do you need to set it up in order to get the breakpoint exception to work as specified above and what incorrect setup would cause it to trigger a general protection fault?
4. What do you think is the point of these mechanisms, particularly in light of what the `user/softint` test program does?

3. 可以使用int 3来抛出一个breakpoint exception，我们在IDT里设置了breakpoint exception的权限为3，于是在用户环境下就可以调用它了。如果问们设权限是0，那么User program就没有执行int 3 的权限，会因为权限问题引发GP fault。

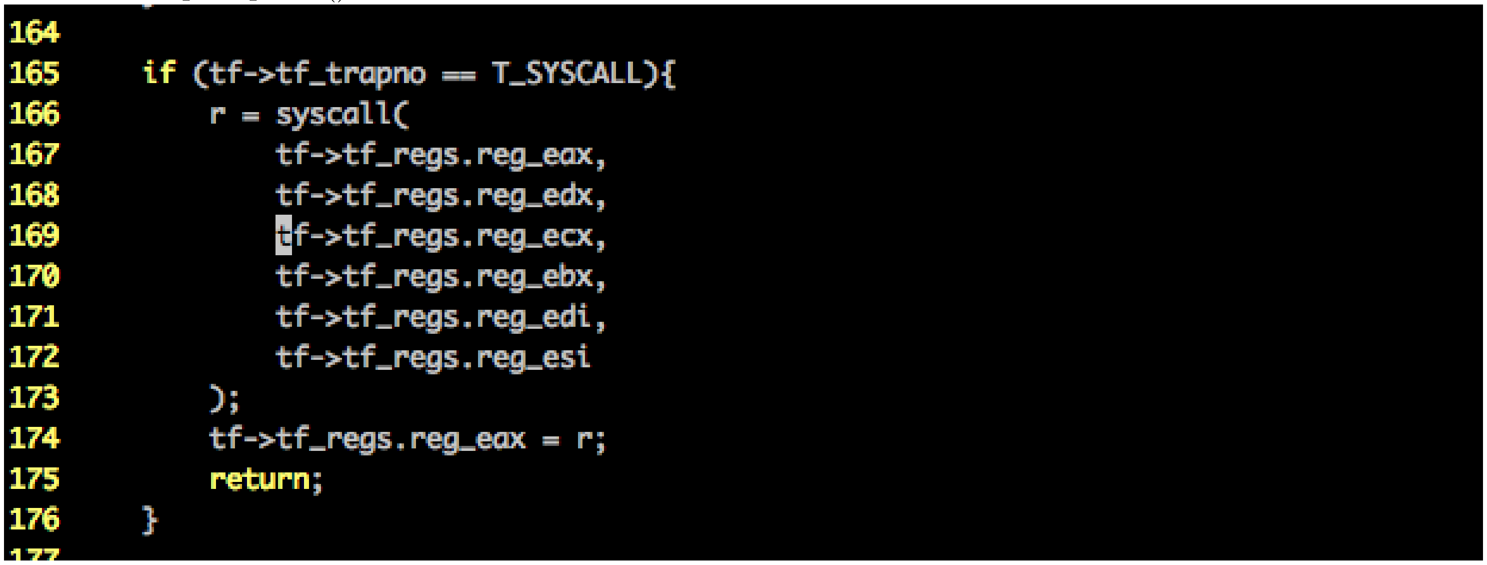
4. 这些机制，我认为，主要是用来限制user program的访问权限，使得user program 只能在一些限制通过有限的入口进入kernel，越少的入口越容易检查吧。减少被攻击的风险把。

3.3 System calls

Exercise 7. Add a handler in the kernel for interrupt vector `T_SYSCALL`. You will have to edit `kern/trapentry.S` and `kern/trap.c`'s `trap_init()`. You also need to change `trap_dispatch()` to handle the system call interrupt by calling `syscall()` (defined in `kern/syscall.c`) with the appropriate arguments, and then arranging for the return value to be passed back to the user process in `%eax`. Finally, you need to implement `syscall()` in `kern/syscall.c`. Make sure `syscall()` returns `-E_INVAL` if the system call number is invalid. You should read and understand `lib/syscall.c` (especially the inline assembly routine) in order to confirm your understanding of the system call interface. You may also find it helpful to read `inc/syscall.h`.

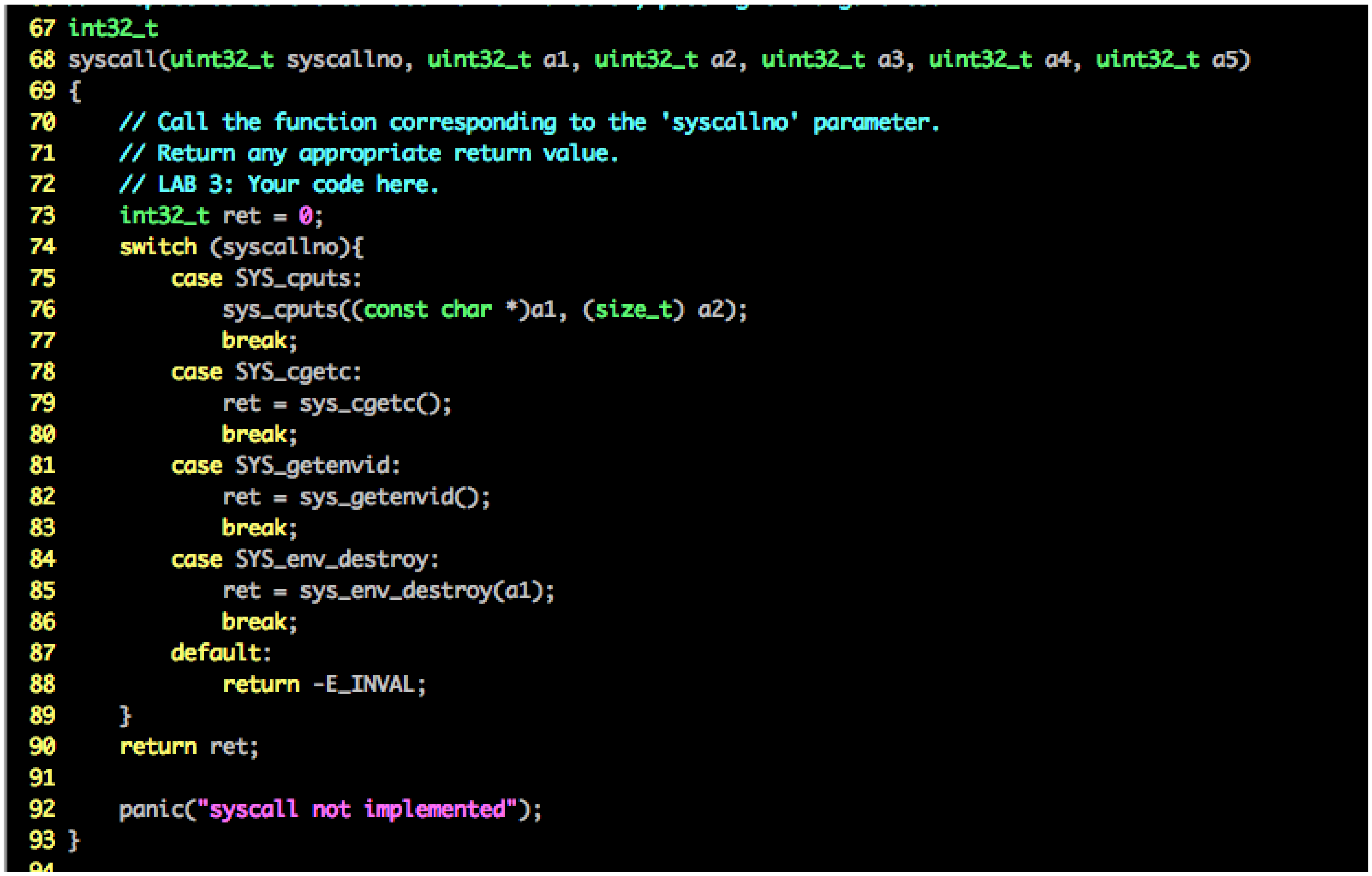
Run the `user/hello` program under your kernel (`make run-hello`). It should print "`hello, world`" on the console and then cause a page fault in user mode. If this does not happen, it probably means your system call handler isn't quite right. You should also now be able to get `make grade` to succeed on the `testbss` test.

添加0x30的system call这个我在上面的截图中已经加进去了，（做完了才开始写报告），就不重复贴图了。对于trap_dispatch()的代码，如下：



```

    if (tf->tf_trapno == T_SYSCALL){
        r = 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
        );
        tf->tf_regs.reg_eax = r;
        return;
    }
```

再就是syscall.c 中的修改，完成后通过textbss 的测试。

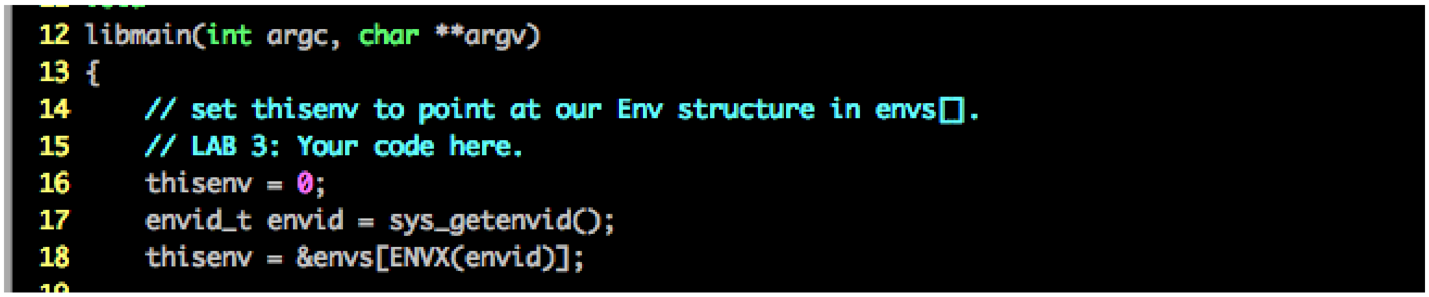


```
int32_t
syscall(uint32_t syscallno, uint32_t a1, uint32_t a2, uint32_t a3, uint32_t a4, uint32_t a5)
{
    // Call the function corresponding to the 'syscallno' parameter.
    // Return any appropriate return value.
    // LAB 3: Your code here.
    int32_t ret = 0;
    switch (syscallno){
        case SYS_cputs:
            sys_cputs((const char *)a1, (size_t) a2);
            break;
        case SYS_cgetc:
            ret = sys_cgetc();
            break;
        case SYS_getenvid:
            ret = sys_getenvid();
            break;
        case SYS_env_destroy:
            ret = sys_env_destroy(a1);
            break;
        default:
            return -E_INVAL;
    }
    return ret;

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

3.4 User-mode startup

Exercise 8. Add the required code to the user library, then boot your kernel. You should see `user/hello` print "`hello, world`" and then print "`i am environment 00001000`". `user/hello` then attempts to "exit" by calling `sys_env_destroy()` (see `lib/libmain.c` and `lib/exit.c`). Since the kernel currently only supports one user environment, it should report that it has destroyed the only environment and then drop into the kernel monitor. You should be able to get **`make grade`** to succeed on the `hello` test.

只需要在lib/libmain.c的libmain()里添加几行，将thisenv赋值。下面的libmain() 会调用umain来执行用户程序，在执行完后通过sys_env_destroy()结束用户程序



```
libmain(int argc, char **argv)
{
	// set thisenv to point at our Env structure in envs[].
	// LAB 3: Your code here.
	thisenv = 0;
	envid_t envid = sys_getenvid();
	thisenv = &envs[ENVX(envid)];
```

3.5 Page faults and memory protection

要处理的问题主要有：

1. Kernel要恩哪个判定user Program 所给指针的address space

2. kernel 要能判定user pprogram 所给指针的权限。

Exercise 9. Change `kern/trap.c` to panic if a page fault happens in kernel mode.

Hint: to determine whether a fault happened in user mode or in kernel mode, check the low bits of the `tf_cs`.

Read `user_mem_assert` in `kern/pmap.c` and implement `user_mem_check` in that same file.

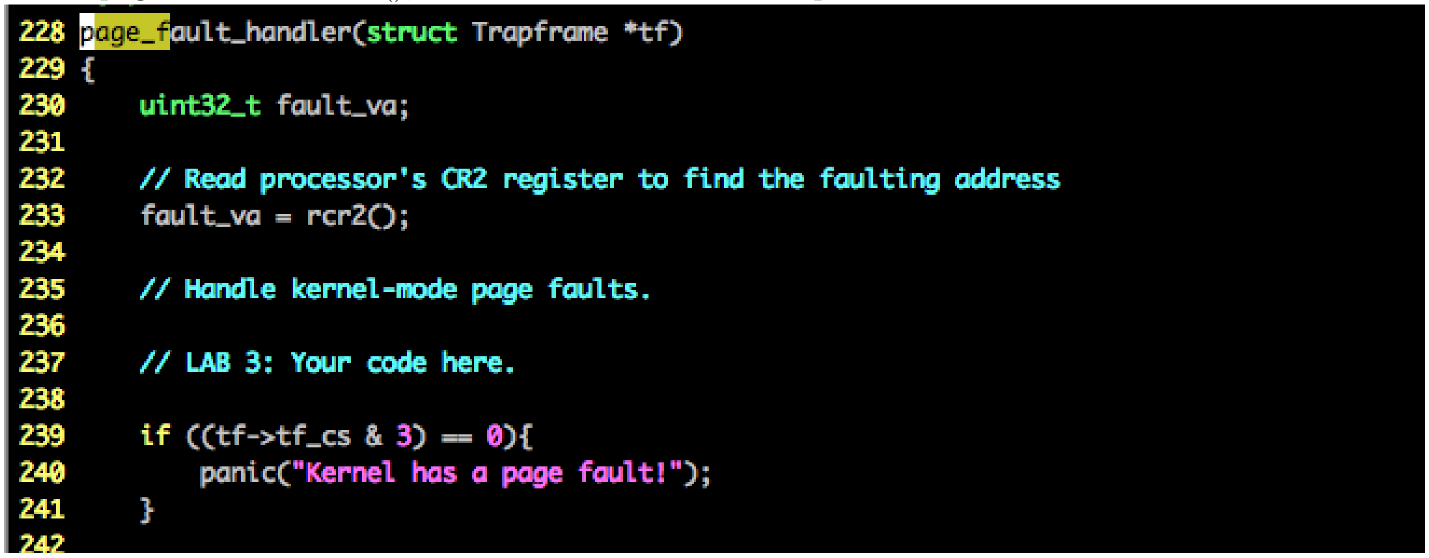
Change `kern/syscall.c` to sanity check arguments to system calls.

Boot your kernel, running `user/buggyhello`. The environment should be destroyed, and the kernel should *not* panic. You should see:

```
[00001000] user_mem_check assertion failure for va 00000001
[00001000] free env 00001000
Destroyed the only environment - nothing more to do!
```

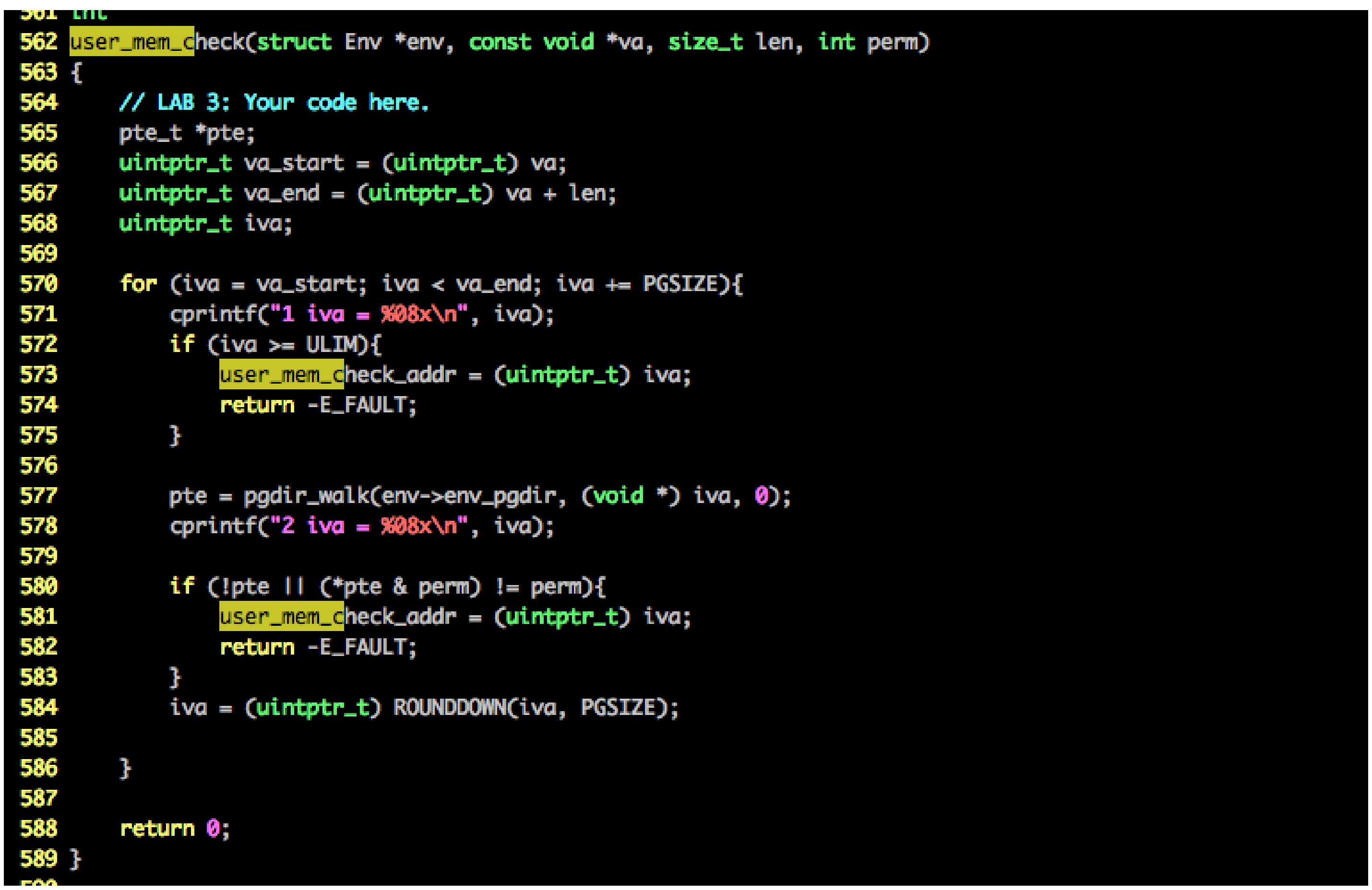
Finally, change `debuginfo_eip` in `kern/kdebug.c` to call `user_mem_check` on `usd`, `stabs`, and `stabstr`. If you now run `user/breakpoint`, you should be able to run **`backtrace`** from the kernel monitor and see the backtrace traverse into `lib/libmain.c` before the kernel panics with a page fault. What causes this page fault? You don't need to fix it, but you should understand why it happens.

- 首先是page_default_handler(),判定如果是kernel mode 就是panic.

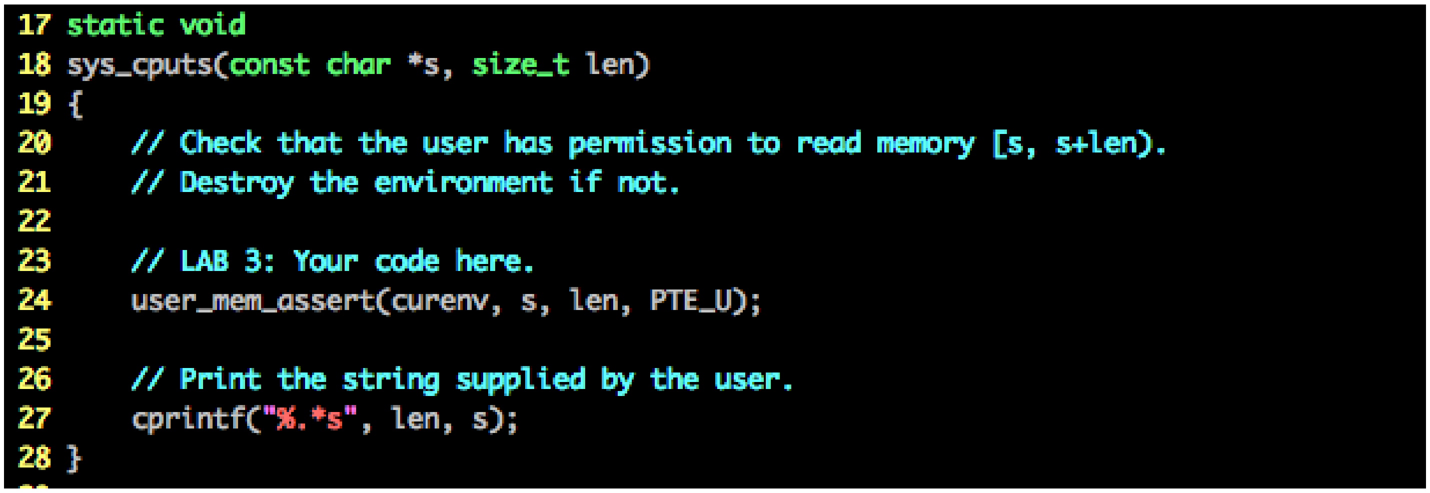


```
page_fault_handler(struct Trapframe *tf)
{
    uint32_t fault_va;

    // Read processor's CR2 register to find the faulting address
    fault_va = rcr2();

    // Handle kernel-mode page faults.

    // LAB 3: Your code here.

    if ((tf->tf_cs & 3) == 0){
        panic("Kernel has a page fault!");
    }

```

- 接下来是user_mem_check()和sys_cputs(),colorred 注意这里的user_mem_check()的判定，一开始不能ROUNDDOWN,不然无法得到正确的结果



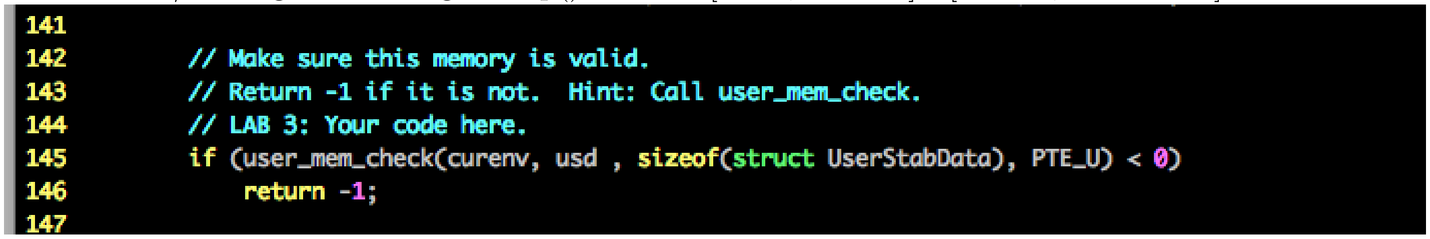
```
user_mem_check(struct Env *env, const void *va, size_t len, int perm)
{
    // LAB 3: Your code here.
    pte_t *pte;
    uintptr_t va_start = (uintptr_t) va;
    uintptr_t va_end = (uintptr_t) va + len;
    uintptr_t iva;

    for (iva = va_start; iva < va_end; iva += PGSIZE){
        cprintf("1 iva = %08x\n", iva);
        if (iva >= ULIM){
            user_mem_check_addr = (uintptr_t) iva;
            return -E_FAULT;
        }

        pte = pgdir_walk(env->env_pgdir, (void *) iva, 0);
        cprintf("2 iva = %08x\n", iva);

        if (!pte || (*pte & perm) != perm){
            user_mem_check_addr = (uintptr_t) iva;
            return -E_FAULT;
        }
        iva = (uintptr_t) ROUNDDOWN(iva, PGSIZE);

    }

    return 0;
}
```



```
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.

    // LAB 3: Your code here.
    user_mem_assert(curenv, s, len, PTE_U);

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

- 到上面为止就可以通过make run-buggyhello了。

  最后修改ker/kdebug.c中的debuginfo_eip()函数，对[stabs,stat_end]和[stabstr,stabstr_end]进行检查。

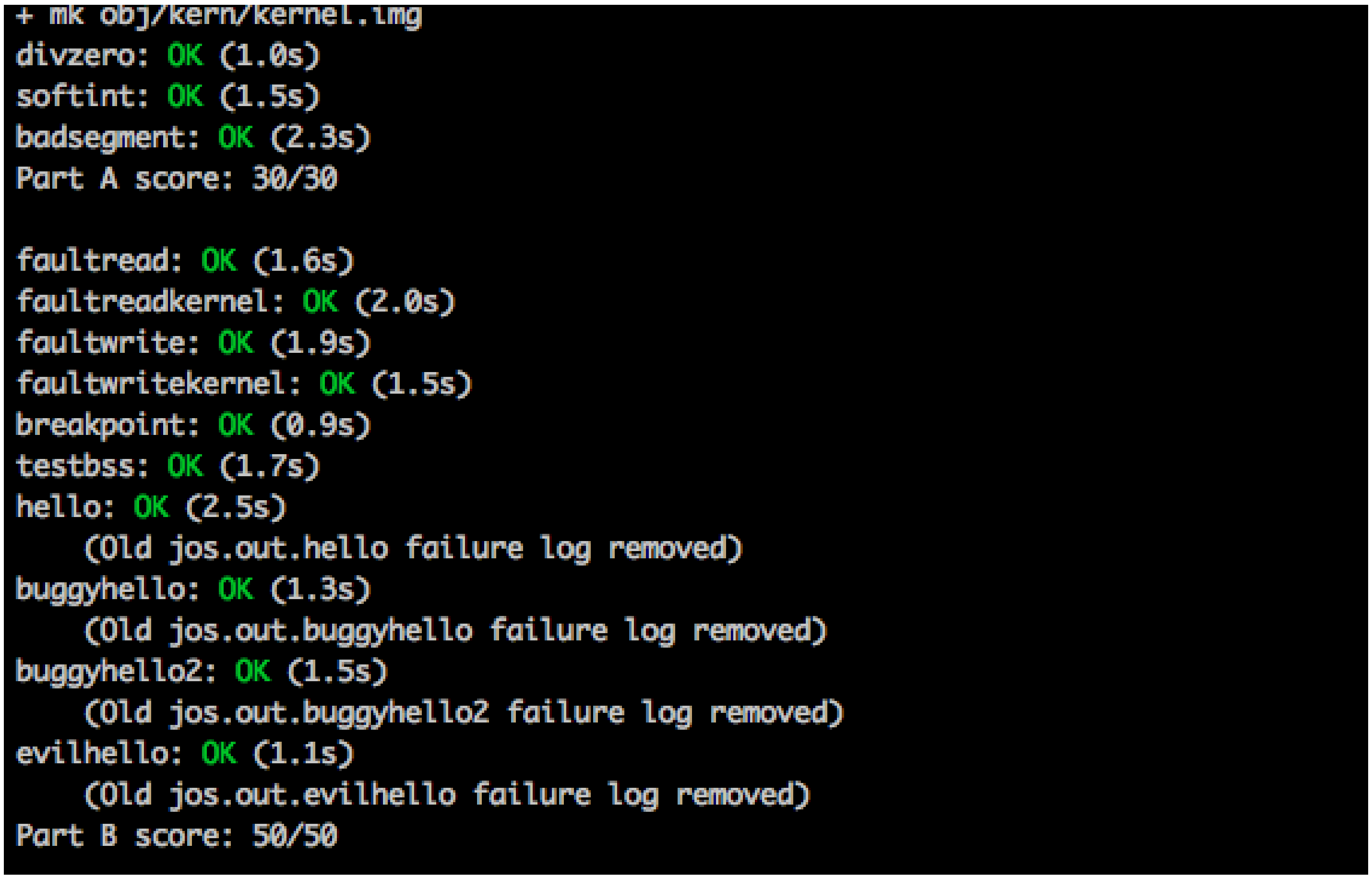


```

        // Make sure this memory is valid.
        // Return -1 if it is not.  Hint: Call user_mem_check.
        // LAB 3: Your code here.
        if (user_mem_check(curenv, usd , sizeof(struct UserStabData), PTE_U) < 0)
            return -1;

```

Exercise 10. Boot your kernel, running `user/evilhello`. The environment should be destroyed, and the kernel should not panic. You should see:

```
[00000000] new env 00001000
[00001000] user_mem_check assertion failure for va f0100020
[00001000] free env 00001000
```

前面作对了，这个貌似就对了？


```
+ mk obj/kern/kernel.img
divzero: OK (1.0s)
softint: OK (1.5s)
badsegment: OK (2.3s)
Part A score: 30/30

faultread: OK (1.6s)
faultreadkernel: OK (2.0s)
faultwrite: OK (1.9s)
faultwritekernel: OK (1.5s)
breakpoint: OK (0.9s)
testbss: OK (1.7s)
hello: OK (2.5s)
    (Old jos.out.hello failure log removed)
buggyhello: OK (1.3s)
    (Old jos.out.buggyhello failure log removed)
buggyhello2: OK (1.5s)
    (Old jos.out.buggyhello2 failure log removed)
evilhello: OK (1.1s)
    (Old jos.out.evilhello failure log removed)
Part B score: 50/50
```

4 Challange

Challenge! You probably have a lot of very similar code right now, between the lists of `TRAPHANDLER` in `trapentry.S` and their installations in `trap.c`. Clean this up. Change the macros in `trapentry.S` to automatically generate a table for `trap.c` to use. Note that you can switch between laying down code and data in the assembler by using the directives `.text` and `.data`.

这个在代码和做法在2.9

5 补充题

在 user/hello.c 程序中调用了 INT 0x30,这是系统调用;user/divzero.c 触发了除零中断。请参考网页 http://pdos.csail.mit.edu/6.828/2012/readings/i386/INT.htm 中的内容,写出这两个 INT 指令所做的具体工作,写出流程即可。

这两者需要区分开？我觉得两个都是软件中断？没有想清楚这两者流程的区别

user/hello.c调用INT 0x30,是系统调用了INTERRUPT-GATE,执行下面代码

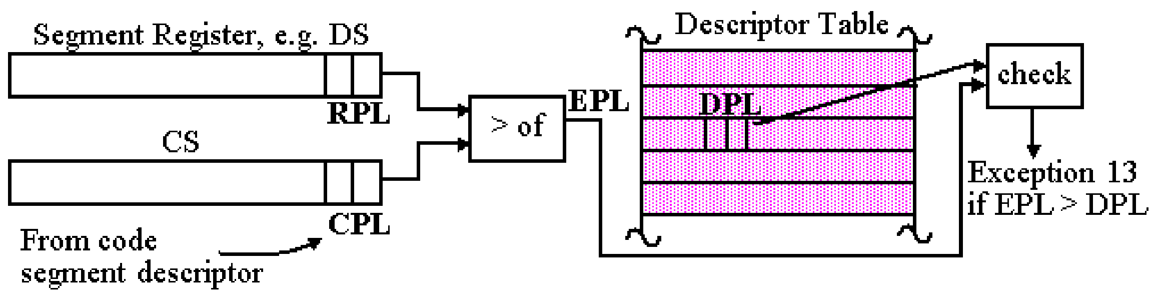
```
IF PE = 0
THEN GOTO REAL-ADDRESS-MODE;
ELSE GOTO PROTECTED-MODE;
FI;
```

然后PE是1，进入保护模式，

```
PROTECTED-MODE:
   Interrupt vector must be within IDT table limits,
      else #GP(vector number * 8+2+EXT);
   Descriptor AR byte must indicate interrupt gate, trap gate, or task gate,
      else #GP(vector number * 8+2+EXT);
   IF software interrupt (* i.e. caused by INT n, INT 3, or INTO *)
   THEN
      IF gate descriptor DPL < CPL
      THEN #GP(vector number * 8+2+EXT);
      FI;
   FI;
   Gate must be present, else #NP(vector number * 8+2+EXT);
   IF trap gate OR interrupt gate
   THEN GOTO TRAP-GATE-OR-INTERRUPT-GATE;
   ELSE GOTO TASK-GATE;
   FI;
```

中间涉及权限之类的，DPL(Descriptor Privilege Level),CPL(Code privilege level),如果超出IDT表，或者AR byte没有表明这是一个gate，又或者这是一个软件中断，而且，权限不够都抛出GP

- *Privilege Level Definitions:*


Segment Register, e.g. DS
RPL
CS
CPL
From code segment descriptor
> of
EPL
Descriptor Table
DPL
check
Exception 13 if EPL > DPL

假设没有权限问题，我们的int 0x30 中断和除零中断 然后通过Trap gate，或者Interrupt gate,被指派到INTERRUPT-TO-INNER-PRIVILEGE去执行。

```
TRAP-GATE-OR-INTERRUPT-GATE:
   Examine CS selector and descriptor given in the gate descriptor;
   Selector must be non-null, else #GP (EXT);
   Selector must be within its descriptor table limits
      ELSE #GP(selector+EXT);
   Descriptor AR byte must indicate code segment
      ELSE #GP(selector + EXT);
   Segment must be present, else #NP(selector+EXT);
   IF code segment is non-conforming AND DPL < CPL
   THEN GOTO INTERRUPT-TO-INNER-PRIVILEGE;
   ELSE
      IF code segment is conforming OR code segment DPL = CPL
      THEN GOTO INTERRUPT-TO-SAME-PRIVILEGE-LEVEL;
      ELSE #GP(CS selector + EXT);
      FI;
   FI;
```

同样的如果超过GDT（or LDT），AR byte 没有表明这是一个代码段 那么抛出GP

如此下去，一层层解释下去，设置IF，TF，SS，保存TSS，恢复TSS等，只看了个大概。