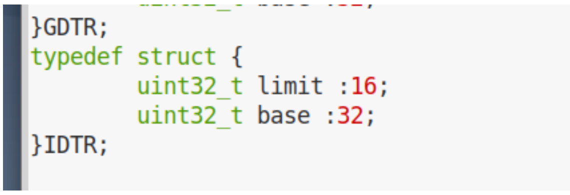
实验过程

通过自陷实现系统调用

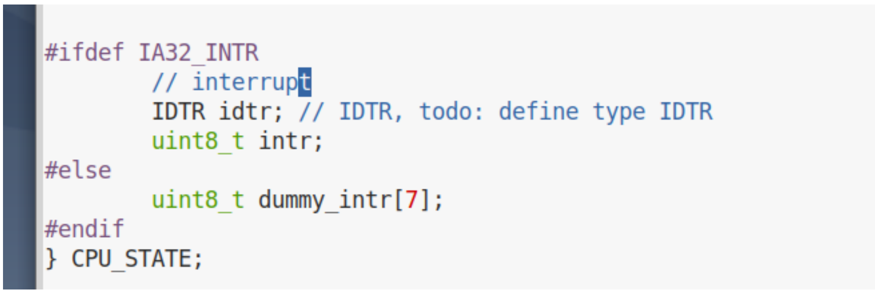
1. 在 include/config.h 中定义宏 IA32_INTR 并 make clean;

2. 在 `nemu/include/cpu/reg.h` 中定义 `IDTR` 结构体，并在 `CPU_STATE` 中添加 `idtr`;

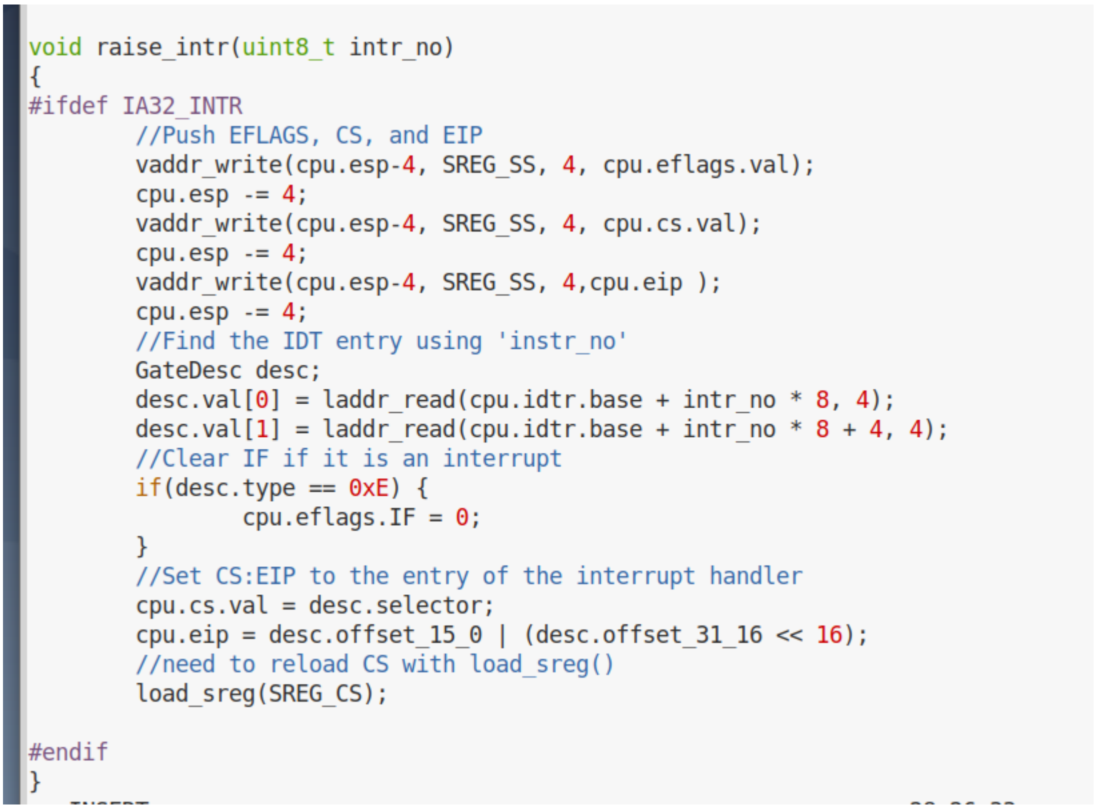


```
}GDTR;
typedef struct {
        uint32_t limit :16;
        uint32_t base :32;
}IDTR;
```

3. 在 nemu/src/cpu/intr.c 中实现 raise_intr()函数;



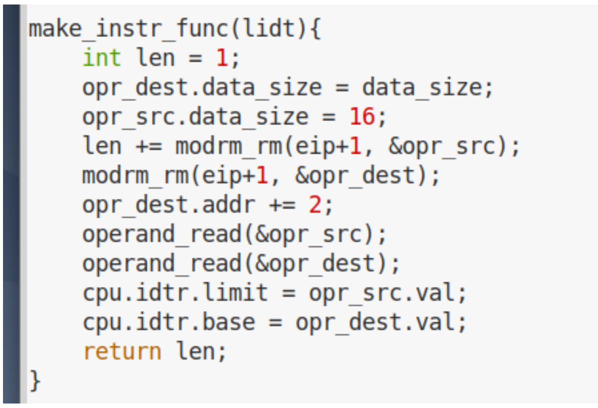
```
#ifdef IA32_INTR
        // interrupt
        IDTR idtr; // IDTR, todo: define type IDTR
        uint8_t intr;
#else
        uint8_t dummy_intr[7];
#endif
} CPU_STATE;
```



```
void raise_intr(uint8_t intr_no)
{
#ifdef IA32_INTR
        //Push EFLAGS, CS, and EIP
        vaddr_write(cpu.esp-4, SREG_SS, 4, cpu.eflags.val);
        cpu.esp -= 4;
        vaddr_write(cpu.esp-4, SREG_SS, 4, cpu.cs.val);
        cpu.esp -= 4;
        vaddr_write(cpu.esp-4, SREG_SS, 4,cpu.eip );
        cpu.esp -= 4;
        //Find the IDT entry using 'instr_no'
        GateDesc desc;
        desc.val[0] = laddr_read(cpu.idtr.base + intr_no * 8, 4);
        desc.val[1] = laddr_read(cpu.idtr.base + intr_no * 8 + 4, 4);
        //Clear IF if it is an interrupt
        if(desc.type == 0xE) {
                cpu.eflags.IF = 0;
        }
        //Set CS:EIP to the entry of the interrupt handler
        cpu.cs.val = desc.selector;
        cpu.eip = desc.offset_15_0 | (desc.offset_31_16 << 16);
        //need to reload CS with load_sreg()
        load_sreg(SREG_CS);

#endif
}
```

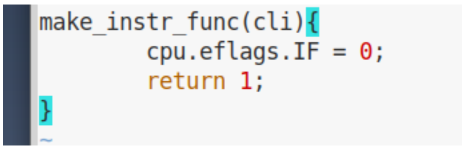
4. 实现包括 lidt、cli、sti、int、pusha、popa、iret 等指令;

lidt:



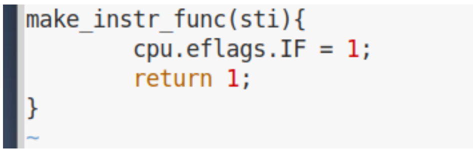
```c
make_instr_func(lidt){
    int len = 1;
    opr_dest.data_size = data_size;
    opr_src.data_size = 16;
    len += modrm_rm(eip+1, &opr_src);
    modrm_rm(eip+1, &opr_dest);
    opr_dest.addr += 2;
    operand_read(&opr_src);
    operand_read(&opr_dest);
    cpu.idtr.limit = opr_src.val;
    cpu.idtr.base = opr_dest.val;
    return len;
}
```

cli：



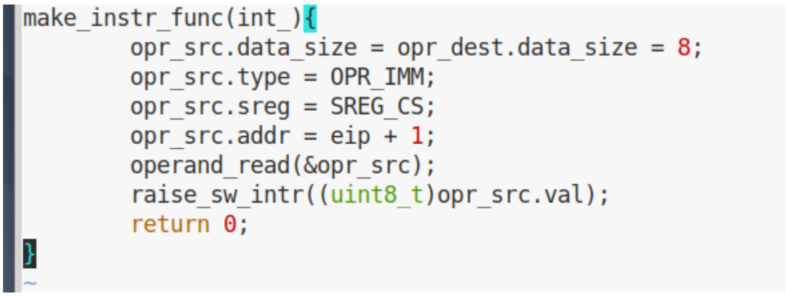
```c
make_instr_func(cli){
        cpu.eflags.IF = 0;
        return 1;
}
~
```

sti:



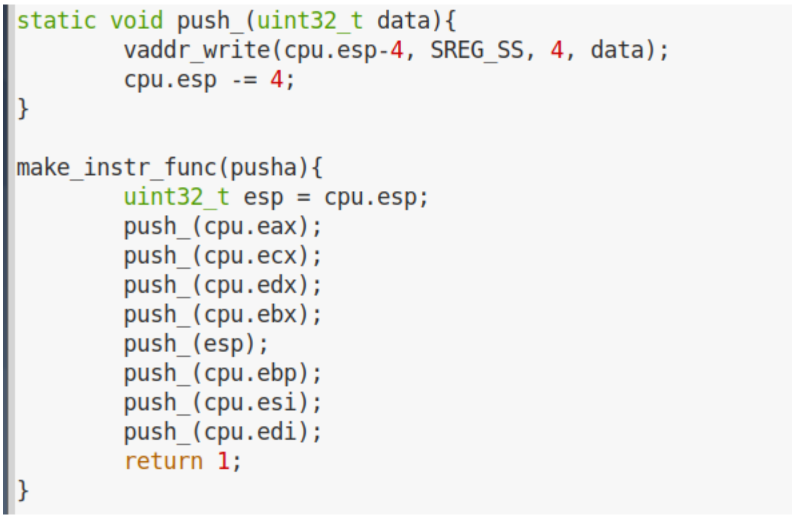
```c
make_instr_func(sti){
        cpu.eflags.IF = 1;
        return 1;
}
~
```

int_:



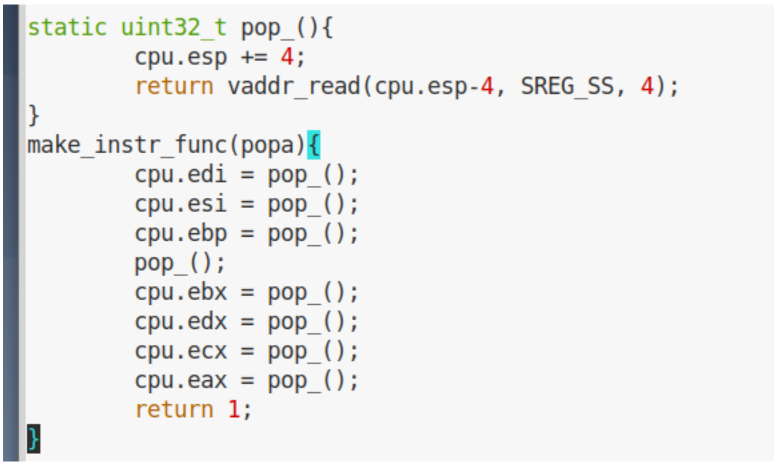
```c
make_instr_func(int_){
        opr_src.data_size = opr_dest.data_size = 8;
        opr_src.type = OPR_IMM;
        opr_src.sreg = SREG_CS;
        opr_src.addr = eip + 1;
        operand_read(&opr_src);
        raise_sw_intr((uint8_t)opr_src.val);
        return 0;
}
~
```

pusha:



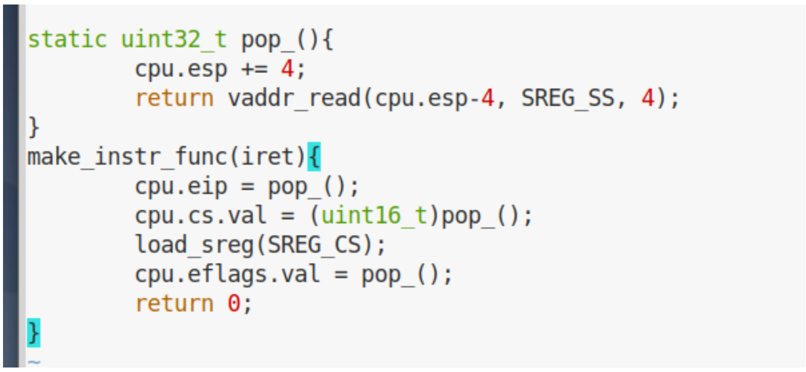
```c
static void push_(uint32_t data){
        vaddr_write(cpu.esp-4, SREG_SS, 4, data);
        cpu.esp -= 4;
}

make_instr_func(pusha){
        uint32_t esp = cpu.esp;
        push_(cpu.eax);
        push_(cpu.ecx);
        push_(cpu.edx);
        push_(cpu.ebx);
        push_(esp);
        push_(cpu.ebp);
        push_(cpu.esi);
        push_(cpu.edi);
        return 1;
}
```

popa:



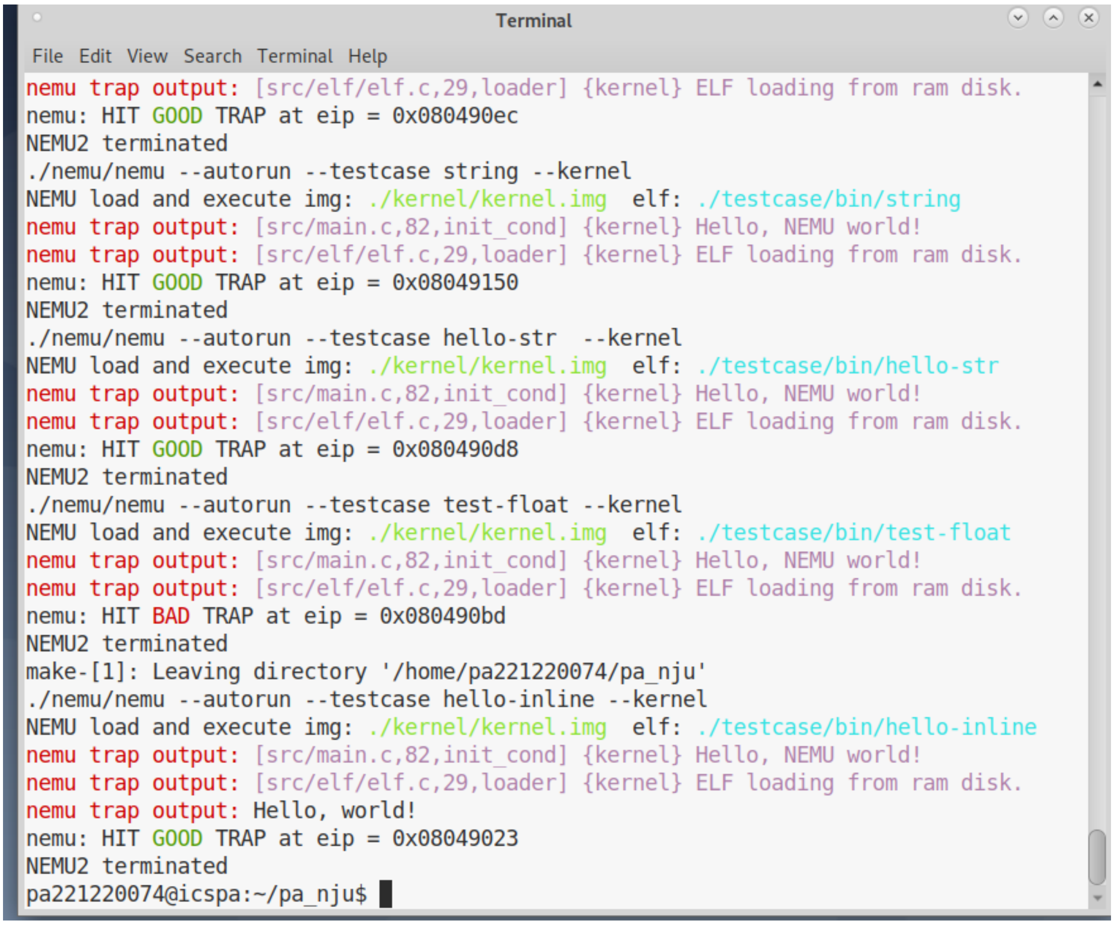
```
static uint32_t pop_(){
        cpu.esp += 4;
        return vaddr_read(cpu.esp-4, SREG_SS, 4);
}
make_instr_func(popa){
        cpu.edi = pop_();
        cpu.esi = pop_();
        cpu.ebp = pop_();
        pop_();
        cpu.ebx = pop_();
        cpu.edx = pop_();
        cpu.ecx = pop_();
        cpu.eax = pop_();
        return 1;
}
```

iret:



```
static uint32_t pop_(){
        cpu.esp += 4;
        return vaddr_read(cpu.esp-4, SREG_SS, 4);
}
make_instr_func(iret){
        cpu.eip = pop_();
        cpu.cs.val = (uint16_t)pop_();
        load_sreg(SREG_CS);
        cpu.eflags.val = pop_();
        return 0;
}
~
```

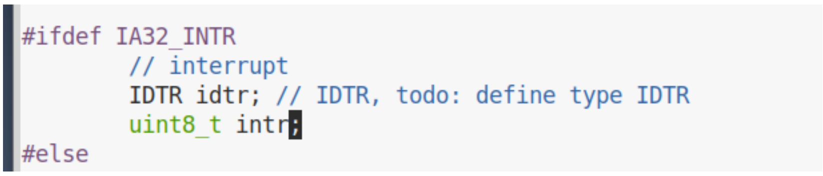
make test_pa-4-1:



```
nemu trap output: [src/elf/elf.c,29,loader] {kernel} ELF loading from ram disk.
nemu: HIT GOOD TRAP at eip = 0x080490ec
NEMU2 terminated
./nemu/nemu --autorun --testcase string --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/string
nemu trap output: [src/main.c,82,init_cond] {kernel} Hello, NEMU world!
nemu trap output: [src/elf/elf.c,29,loader] {kernel} ELF loading from ram disk.
nemu: HIT GOOD TRAP at eip = 0x08049150
NEMU2 terminated
./nemu/nemu --autorun --testcase hello-str  --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/hello-str
nemu trap output: [src/main.c,82,init_cond] {kernel} Hello, NEMU world!
nemu trap output: [src/elf/elf.c,29,loader] {kernel} ELF loading from ram disk.
nemu: HIT GOOD TRAP at eip = 0x080490d8
NEMU2 terminated
./nemu/nemu --autorun --testcase test-float --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/test-float
nemu trap output: [src/main.c,82,init_cond] {kernel} Hello, NEMU world!
nemu trap output: [src/elf/elf.c,29,loader] {kernel} ELF loading from ram disk.
nemu: HIT BAD TRAP at eip = 0x080490bd
NEMU2 terminated
make-[1]: Leaving directory '/home/pa221220074/pa_nju'
./nemu/nemu --autorun --testcase hello-inline --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/hello-inline
nemu trap output: [src/main.c,82,init_cond] {kernel} Hello, NEMU world!
nemu trap output: [src/elf/elf.c,29,loader] {kernel} ELF loading from ram disk.
nemu trap output: Hello, world!
nemu: HIT GOOD TRAP at eip = 0x08049023
NEMU2 terminated
pa221220074@icspa:~/pa_nju$ 
```

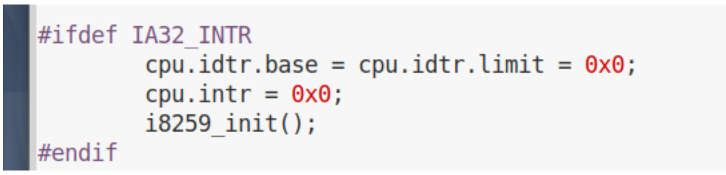
响应时钟中断

1. 在 include/config.h 中定义宏 HAS_DEVICE_TIMER 并 make clean；
2. 在 `nemu/include/cpu/reg.h` 的 `CPU_STATE` 中添加 `uint8_t intr` 成员，模拟中断引脚；



```
#ifdef IA32_INTR
        // interrupt
        IDTR idtr; // IDTR, todo: define type IDTR
        uint8_t intr;
#else
```

在 nemu/src/cpu/cpu.c 的 init_cpu()中初始化 cpu.intr = 0；

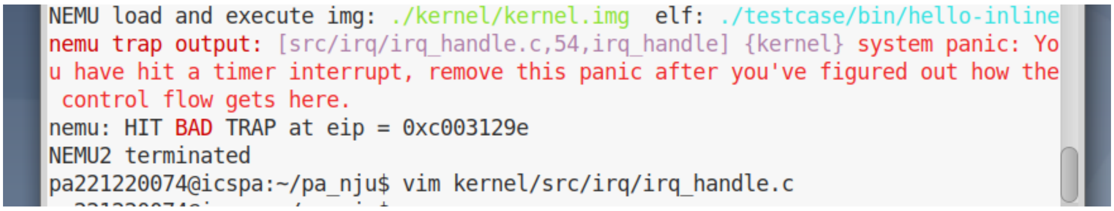


```
#ifdef IA32_INTR
        cpu.idtr.base = cpu.idtr.limit = 0x0;
        cpu.intr = 0x0;
        i8259_init();
#endif
```

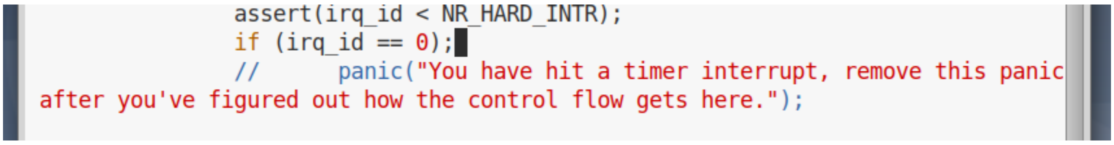
在 nemu/src/cpu/cpu.c 的 exec()函数 while 循环体，每次执行完一条指令后调用 do_intr()函数查看并处理中断事件；

执行 make test_pa-4-1；

看 kernel 的报错信息 找到对应的文件 移除 panic

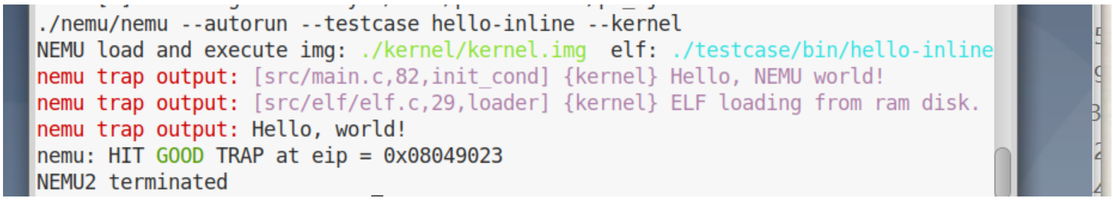


```
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/hello-inline
nemu trap output: [src/irq/irq_handle.c,54,irq_handle] {kernel} system panic: Yo
u have hit a timer interrupt, remove this panic after you've figured out how the
 control flow gets here.
nemu: HIT BAD TRAP at eip = 0xc003129e
NEMU2 terminated
pa221220074@icspa:~/pa_nju$ vim kernel/src/irq/irq_handle.c
```



```
                assert(irq_id < NR_HARD_INTR);
                if (irq_id == 0);
                //      panic("You have hit a timer interrupt, remove this panic
after you've figured out how the control flow gets here.");
```

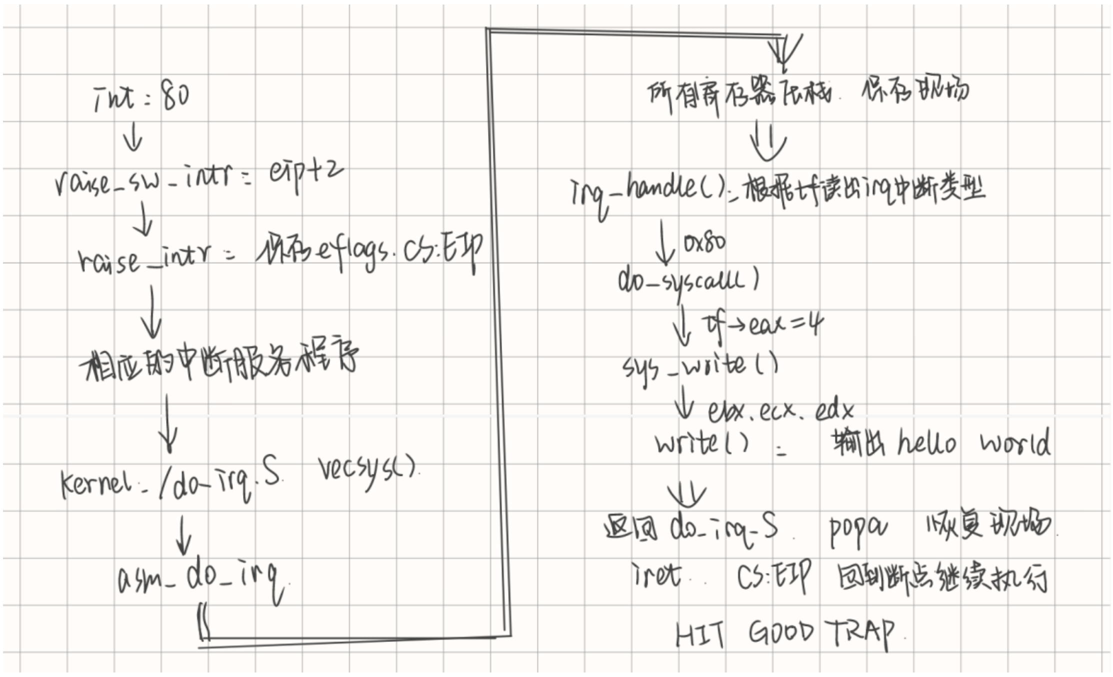
make test_pa-4-1:



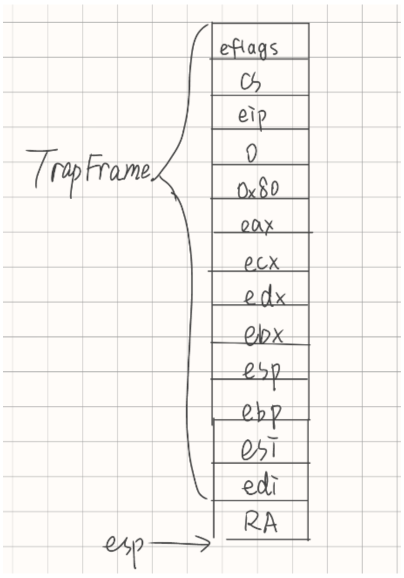
```
./nemu/nemu --autorun --testcase hello-inline --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/hello-inline
nemu trap output: [src/main.c,82,init_cond] {kernel} Hello, NEMU world!
nemu trap output: [src/elf/elf.c,29,loader] {kernel} ELF loading from ram disk.
nemu trap output: Hello, world!
nemu: HIT GOOD TRAP at eip = 0x08049023
NEMU2 terminated
```

§4-1.3.1 通过自陷实现系统调用

1. 详细描述从测试用例中的 int $0x80 开始一直到 HIT_GOOD_TRAP 为止的详细的系统行为（完整描述控制的转移过程，即相关函数的调用和关键参数传递过程），可以通过文字或画图的方式来完成；

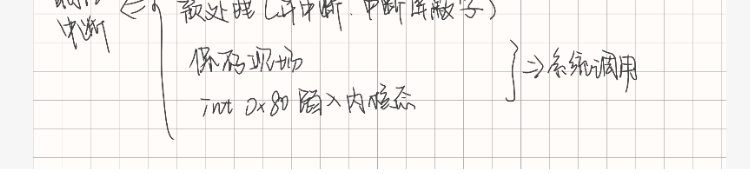

Int : 80
raise_sw_intr : eip+2
raise_intr : 保存eflags. CS:EIP
相应的中断服务程序
kernel : /do-irq.S vecsys()
asm_do_irq
所有寄存器压栈. 保存现场
irq_handle() : 根据tf读出irq中断类型
0x80
do_syscall()
tf→eax=4
sys_write()
ebx. ecx. edx
write() : 输出 hello world
返回 do_irq.S . popa 恢复现场.
iret . CS:EIP 回到断点继续执行
HIT GOOD TRAP.

2. 在描述过程中，回答 kernel/src/irq/do_irq.S 中的 push %esp 起什么作用，画出在 call irq_handle 之前，系统栈的内容和 esp 的位置，指出 TrapFrame 对应系统栈的哪一段内容。



§4-1.3.2 响应时钟中断

1. 详细描述 NEMU 和 Kernel 响应时钟中断的过程和先前的系统调用过程不同之处在哪里？相同的地方又在哪里？可以通过文字或画图的方式来完成。


响应中断
判断是否有中断请求
预处理（开中断、中断屏蔽字）
保存现场
Int 0x80 陷入内核态
系统调用