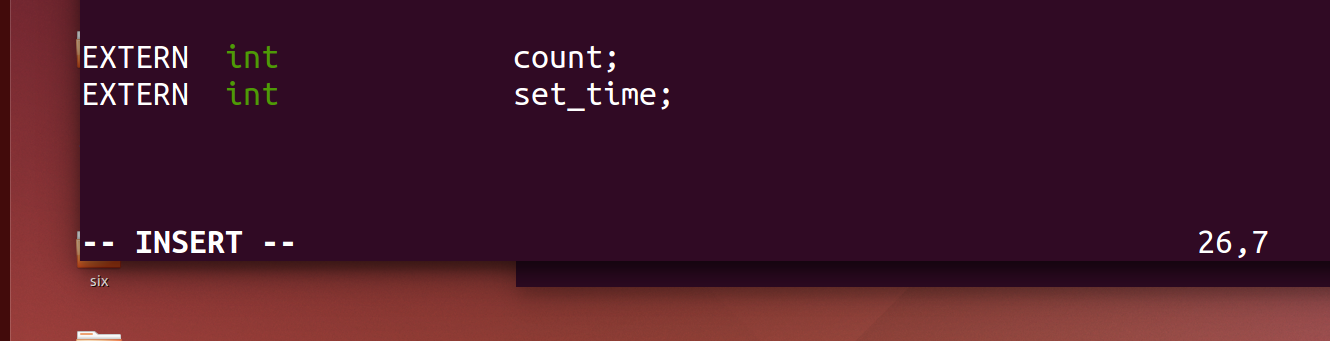
二、设计一个模块，每隔一个自定义时间就运行，并对当前运行的进程代码段和数据

段进行完整性检查。

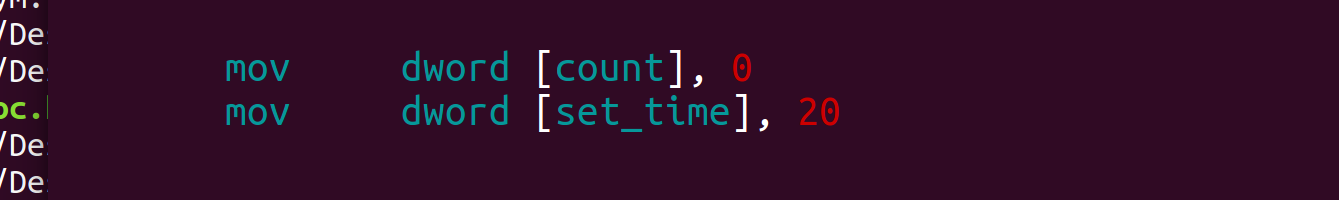
显然，本部分我们会用到并修改时钟中断；

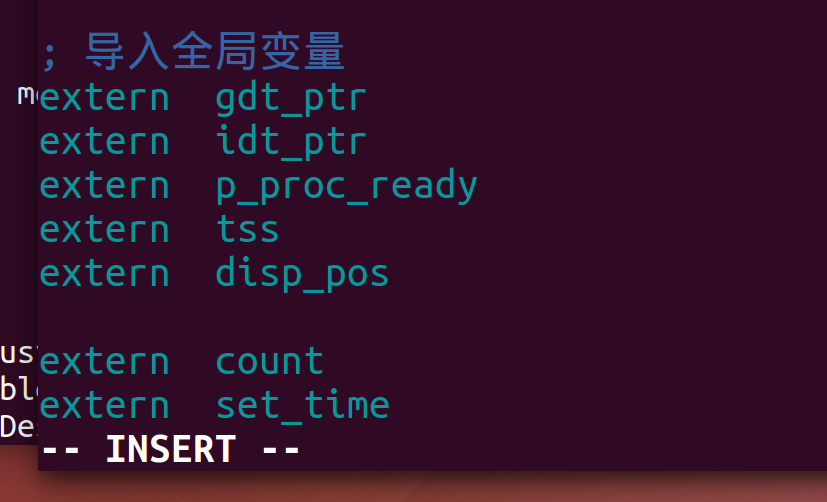
第一步，我们决定先实现一个简单的定时器，至于任意给定时间，笔者采用的实现方式是定义一个全局变量作为计数器，每次发生时钟中断就让这个计时器增加1，之后再定义一个全局的设定值作为设定的时间，用定时器的值和这个全局设定值作比较，这样就可以实现定时功能了；

由于kernel.asm文件的全局变量都定义在global.h文件中，所以我们也如法炮制：



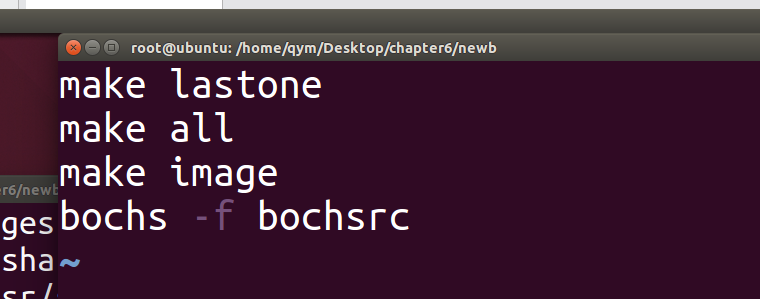
在kernel中对这两个全局变量进行初始化：



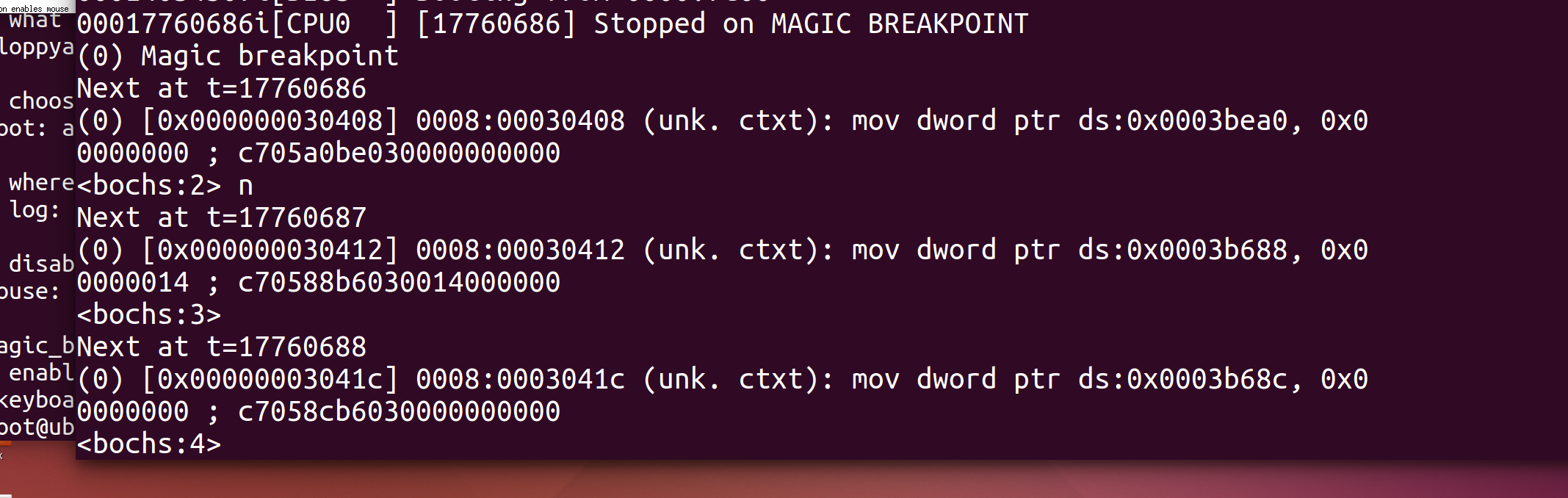


这里先积累一个经验：

就是Makefile的使用，我们需要依次使用以下三个make命令，才能依次实现复制汇编代码、重新编译&&汇编&&链接、重新写入到img中，并最终运行：



下面截图就是给这两个全局变量初始化：



下面为了传参方便，我们采用计算机病毒中常用的call pop方式写了一个定时器效果检验的函数, 每20次时钟中断就会打印一个”hello time\_setter”:

push    eax

    push    ebx

    mov eax, [count]

    mov ebx, [set\_time]

    inc eax

    cmp eax, ebx

    jnz .2

    call    .1

    db  "hello time\_setter", 10, 0

.1:

    call    disp\_str

.2:

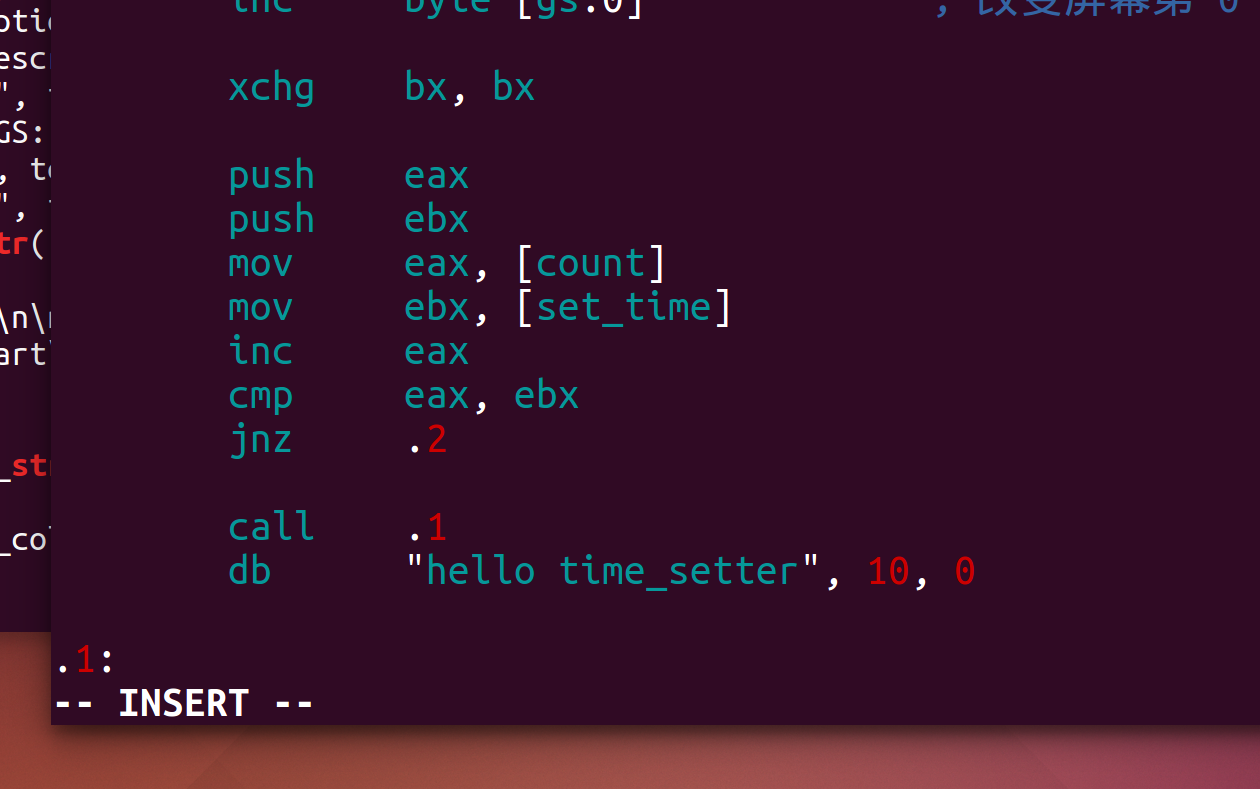
    mov [count], eax

    mov [set\_time], ebx

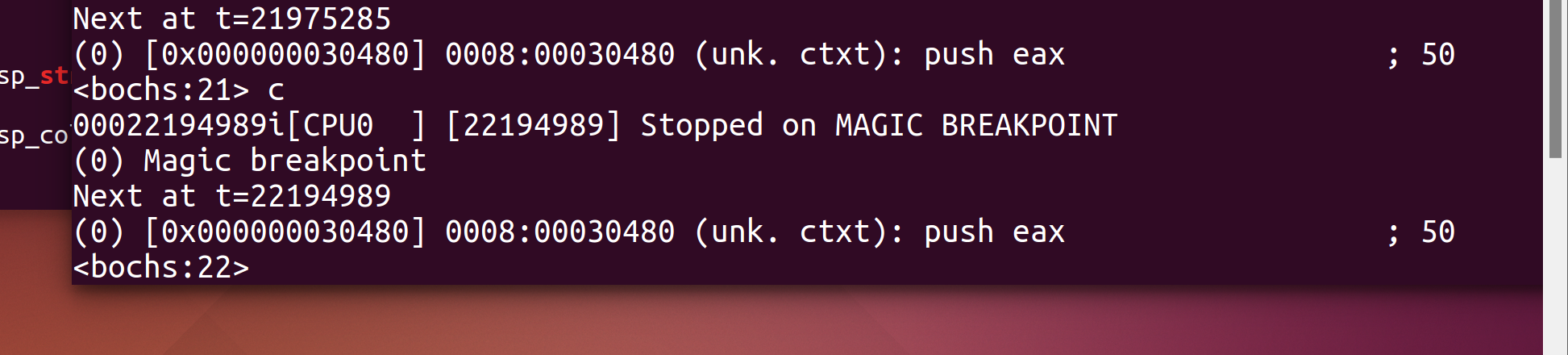
    pop ebx

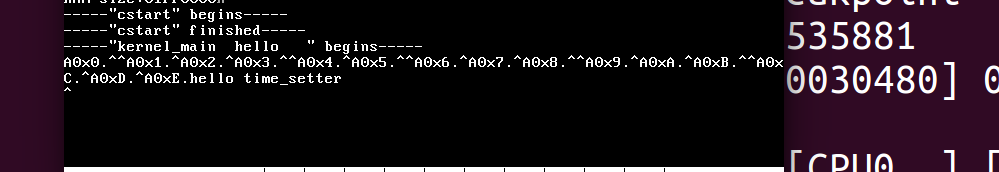
    pop eax

然后我们在时钟中断处下断点：



执行20次断点代码：





可以看到20组字符后边才有我们预期的字符串；

有了定时器之后，下一步就是获取当前进程的text和data段了；

通过逆向分析代码，我们得到灵感，我们是可以通过全局变量p\_proc\_ready找到当前进程的上下文信息的：

restart:

    mov esp, [p\_proc\_ready]

    ;p\_proc\_ready定义域global.h文件中，是一个PCB指针，用于指向已经就虚的进程的PCB

    lldt    [esp + P\_LDT\_SEL]

    lea eax, [esp + P\_STACKTOP]

    mov dword [tss + TSS3\_S\_SP0], eax

restart\_reenter:

    dec dword [k\_reenter]

    pop gs  ;PCB的前4个元素就依次是下面4个寄存器

    pop fs

    pop es

    pop ds

    popad ;popad弹出7个寄存器值

    add esp, 4 ;跳过返回地址retaddr

    iretd ;但是这里还是有eip cs eflags esp ss

我们先分别写一个获取eip、cs值的函数：

get\_eip:

    push    ebx

        mov     ebx, [p\_proc\_ready]

        mov     eax, [ebx + 13 \*4] ;得到当前进程cs寄存器的值

        pop     ebx

        ret

get\_cs:

    push    ebx

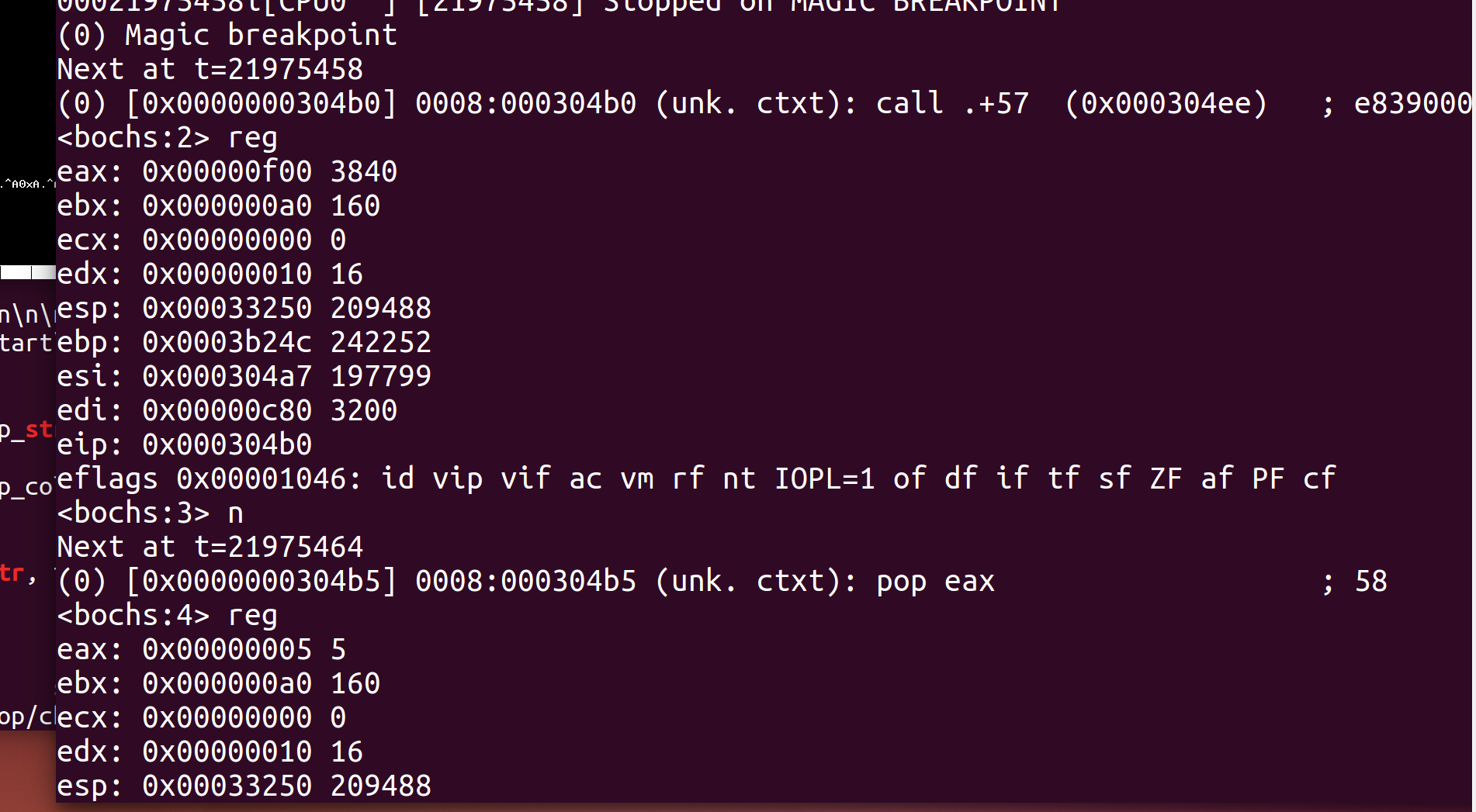
    mov ebx, [p\_proc\_ready]

    mov eax, [ebx + 14 \*4] ;得到当前进程cs寄存器的值

    pop ebx

    ret

在该处下断点并执行，可以看到我们得到了cs寄存器的值为5：



下面写一个简单的验证地址是否正确的程序：

check\_mem:

    push    es

    push    eax

    push    ebx

    mov eax, [esp + 8]  ;第2个参数是获取的cs寄存器值

    mov es, ax

    mov eax, [esp + 4]  ;第1个参数是获取的eip寄存器值

    xchg    bx, bx

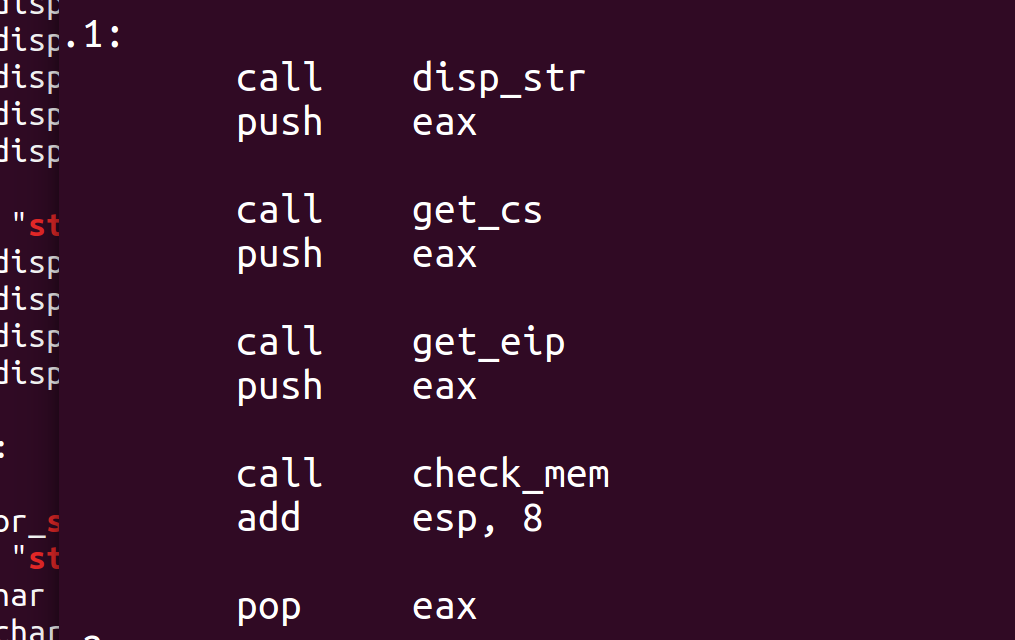
    pop ebx

    pop eax

    pop es

    ret

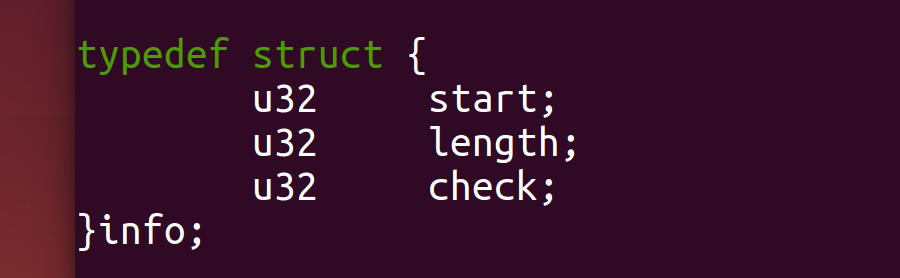
测试调用如下：



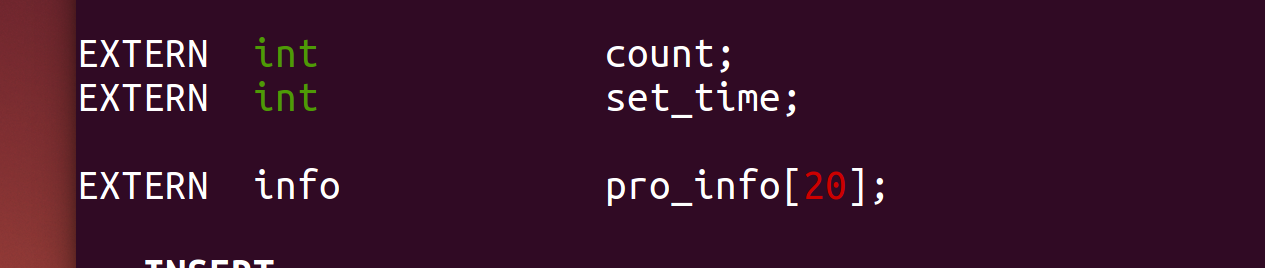
可见，我们正确找到了函数进入中断前的地址：

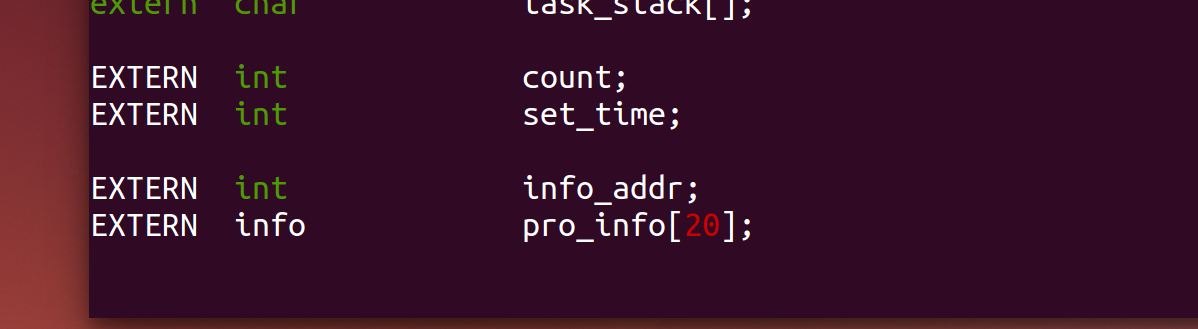


Proc.h

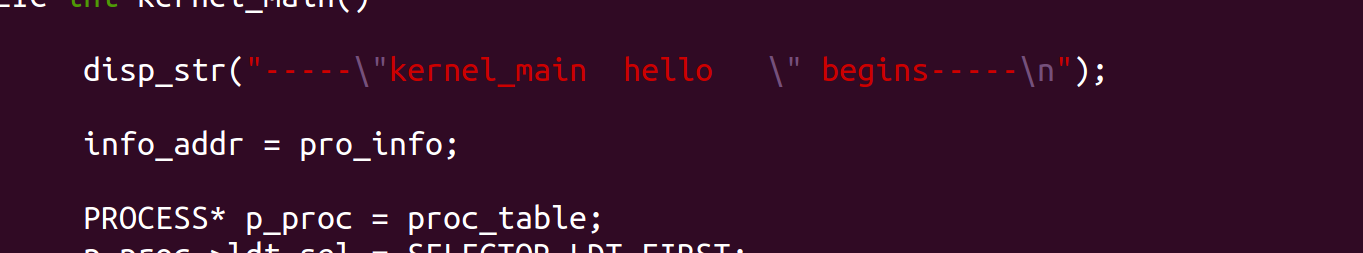


Global.h

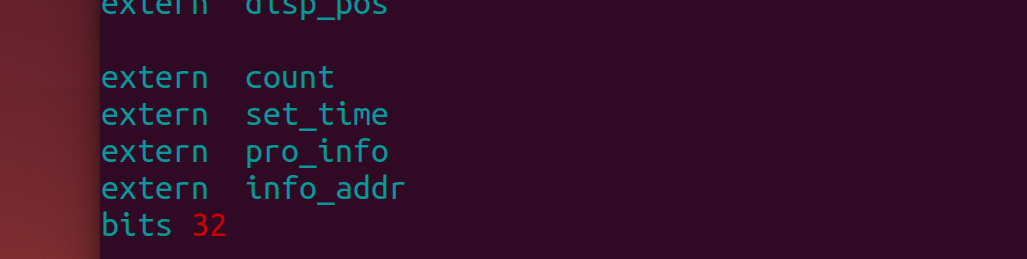




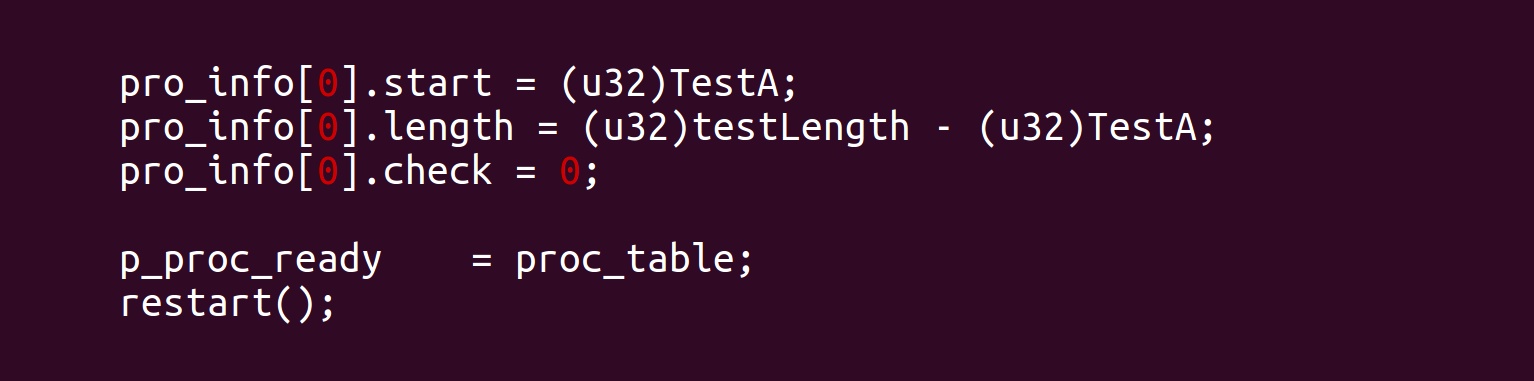
Main.c



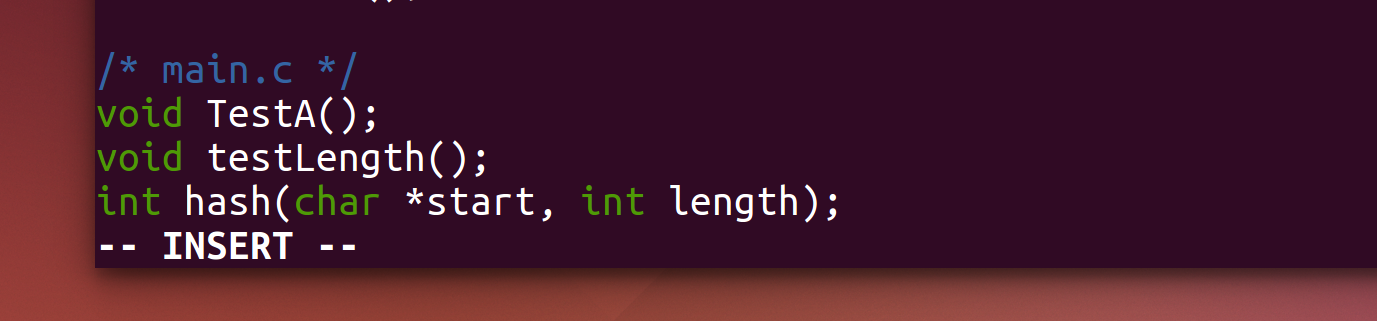
Kernel.asm







Proto.h



Hash

**int** hash(**char** **\***start, **int** length){

**int** a = 0xdeedbeef;

**int** b = 0xabcd1234;

**int** \*p = start;

        length = length / 4 + 1;

**int** res = 0;

**int** tmp = 0;

**int** i = 0;

        while(1){

                if(i >= length) break;

                tmp = a \* (\*p) + b;

                p++;

                res += tmp;

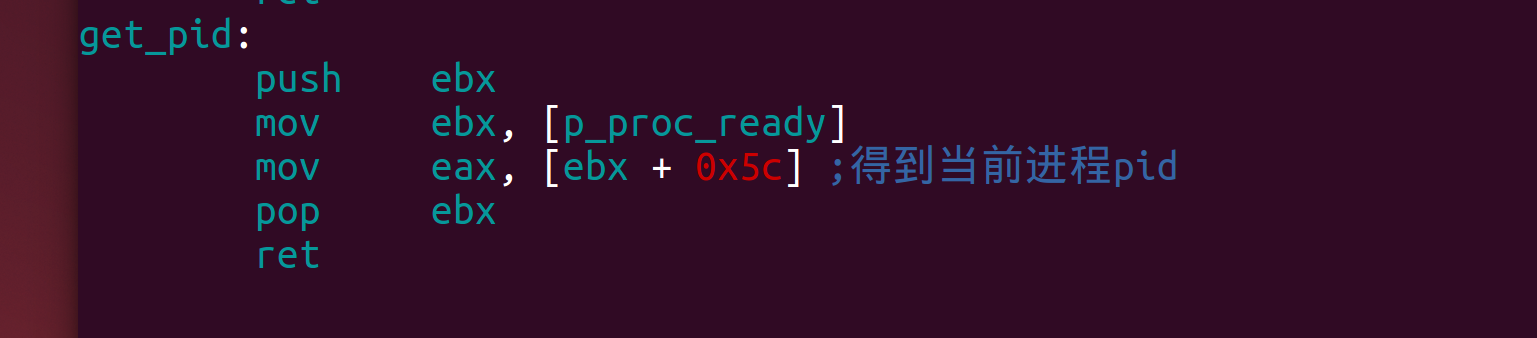
                ++i;

        }

        return res;

}

得到当前进程的pid：



得到当前进程的校验码函数get\_check\_value：

get\_check\_value:

        push    ebx

        push    ecx

        push    edx

        mov     ebx, [info\_addr]

        xor     edx, edx

        mov     ecx, [esp+4] ;第一个参数是进程的pid

        test    ecx, ecx

        jz      .4

.3:

        add     edx, 0xc

        loop    .3

.4:

        ;edx 得到偏移

        add     edx, ebx

        mov     eax, [edx+8]

        ;eax得到check的值

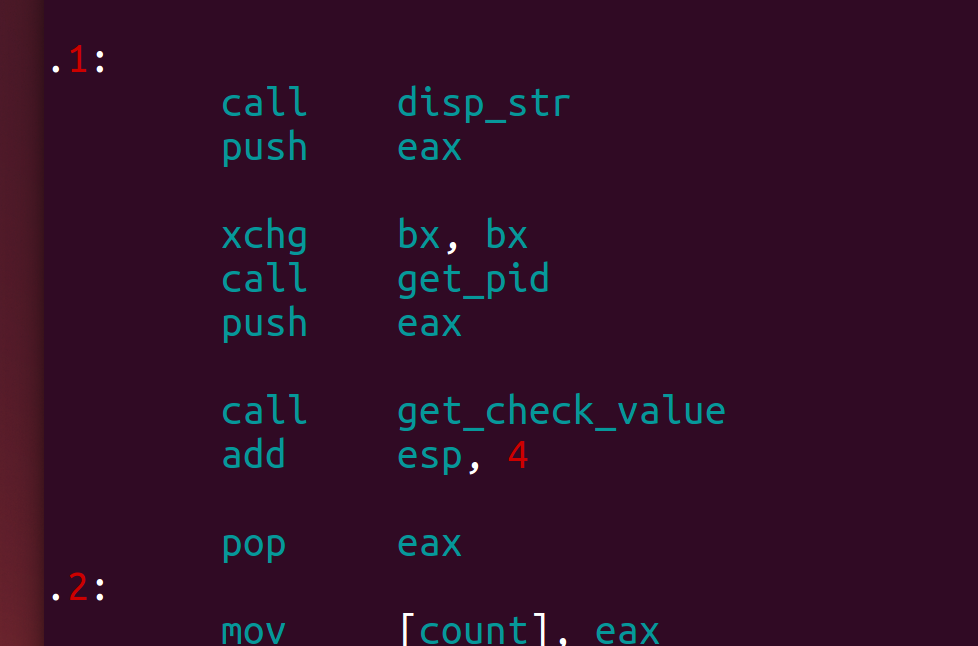
        pop     edx

        pop     ecx

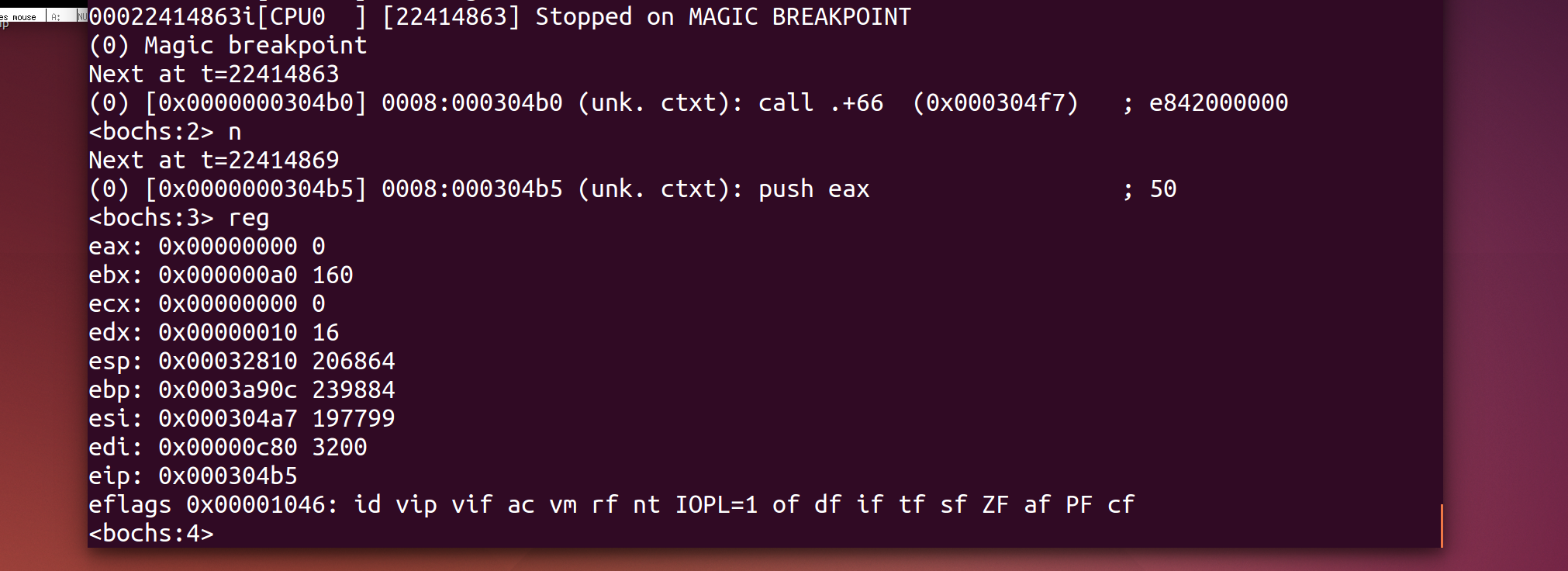
        pop     ebx

        ret

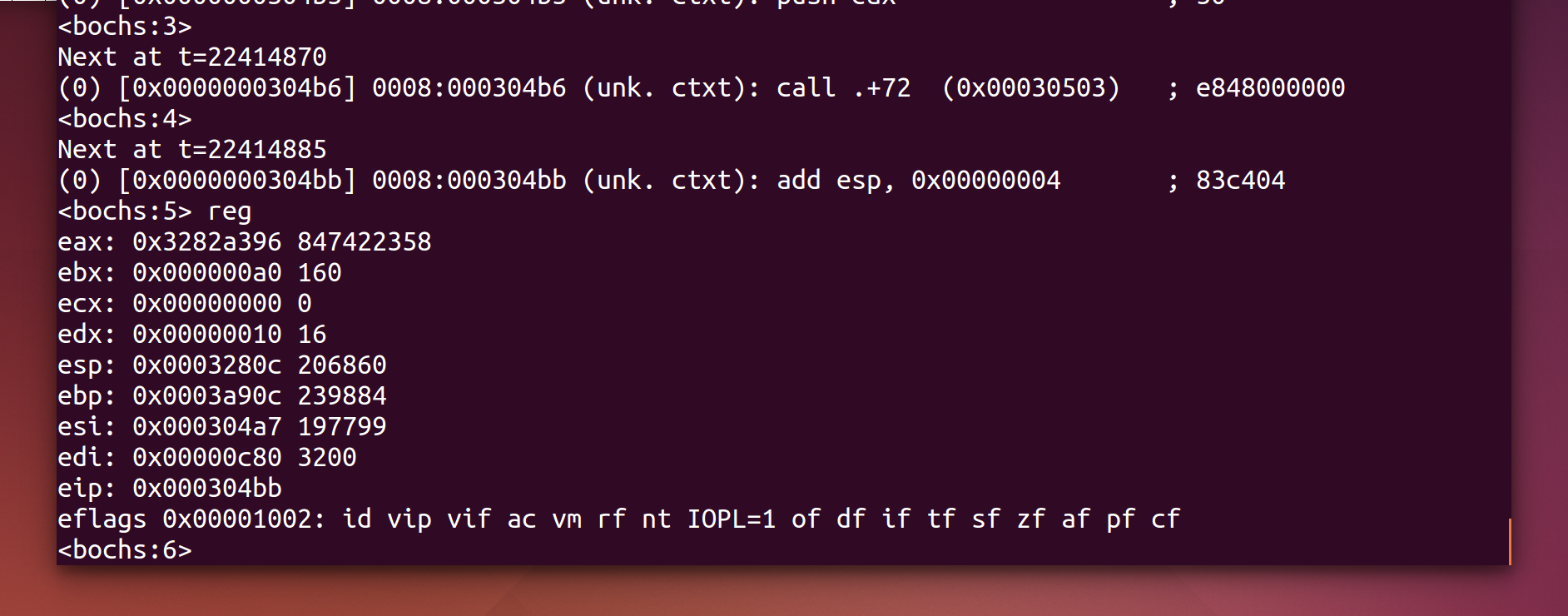
测试代码：



得到进程pid为0：

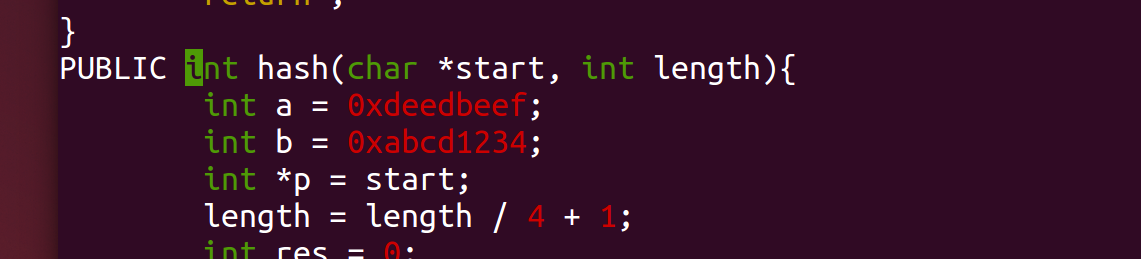


得到校验码：



下面我们只需要调用main里边的hash函数来检验这个校验码是否正确就可以了；

给main.c中的hash函数加上PUBLIC关键字：



在kernel中将其导入：



同时仿照get\_check\_value，写两个函数分别获得起始地址和长度：

get\_start:

        push    ebx

        push    ecx

        push    edx

        mov     ebx, [info\_addr]

        xor     edx, edx

        mov     ecx, [esp+4] ;第一个参数是进程的pid

        test    ecx, ecx

        jz      .4

.3:

        add     edx, 0xc

        loop    .3

.4:

        ;edx 得到偏移

        add     edx, ebx

        mov     eax, [edx]

        pop     edx

        pop     ecx

        pop     ebx

        ret

get\_length:

        push    ebx

        push    ecx

        push    edx

        mov     ebx, [info\_addr]

        xor     edx, edx

        mov     ecx, [esp+4] ;第一个参数是进程的pid

        test    ecx, ecx

        jz      .4

.3:

        add     edx, 0xc

        loop    .3

.4:

        ;edx 得到偏移

        add     edx, ebx

        mov     eax, [edx+4]

        pop     edx

        pop     ecx

        pop     ebx

        ret

最终就是比较新生成的校验码和原来保存的，然后打印不同字符串：

mov eax, [count]

    mov ebx, [set\_time]

    inc eax

    cmp eax, ebx

    jnz .2

    call    .1

    db  "hello time\_setter", 10, 0

.1:

    call    disp\_str

    push    eax

    call    get\_pid

    push    eax

    call    get\_check\_value

    add esp, 4

    mov edi, eax ;edi用来存放check\_value

    call    get\_pid

    push    eax

    call    get\_length

    add esp, 4

    push    eax

    call    get\_pid

    push    eax

    call    get\_start

    add esp, 4

    push    eax

    xchg    bx, bx

    call    hash

    cmp eax, edi

    jz  .6

    call    .5

    db  10, 10, 10, 10, 10, "text section is destoried", 10, 10, 10, 10, 10, 10, 10, 0

.5  call    disp\_str

    jmp .8

.6

    call    .7

    db  10, 10, 10, 10, "text section is okk", 10, 10, 10, 10, 0

.7

    call    disp\_str

.8

    pop eax

.2:

    mov [count], eax

    mov [set\_time], ebx

    pop     edi

    pop ebx

    pop eax

    mov al, EOI         ; `. reenable

    out INT\_M\_CTL, al       ; /  master 8259

    push    clock\_int\_msg

    call    disp\_str

    add esp, 4

    mov esp, [p\_proc\_ready] ; 离开内核栈

    lea eax, [esp + P\_STACKTOP]

    mov dword [tss + TSS3\_S\_SP0], eax

    pop gs  ; `.

    pop fs  ;  |

    pop es  ;  | 恢复原寄存器值

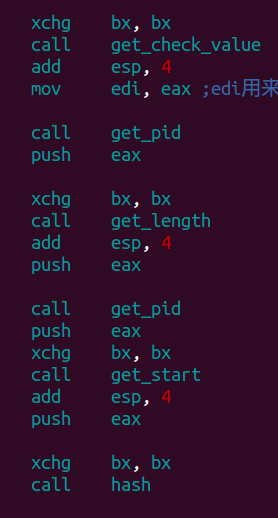
    pop ds  ;  |

    popad       ; /

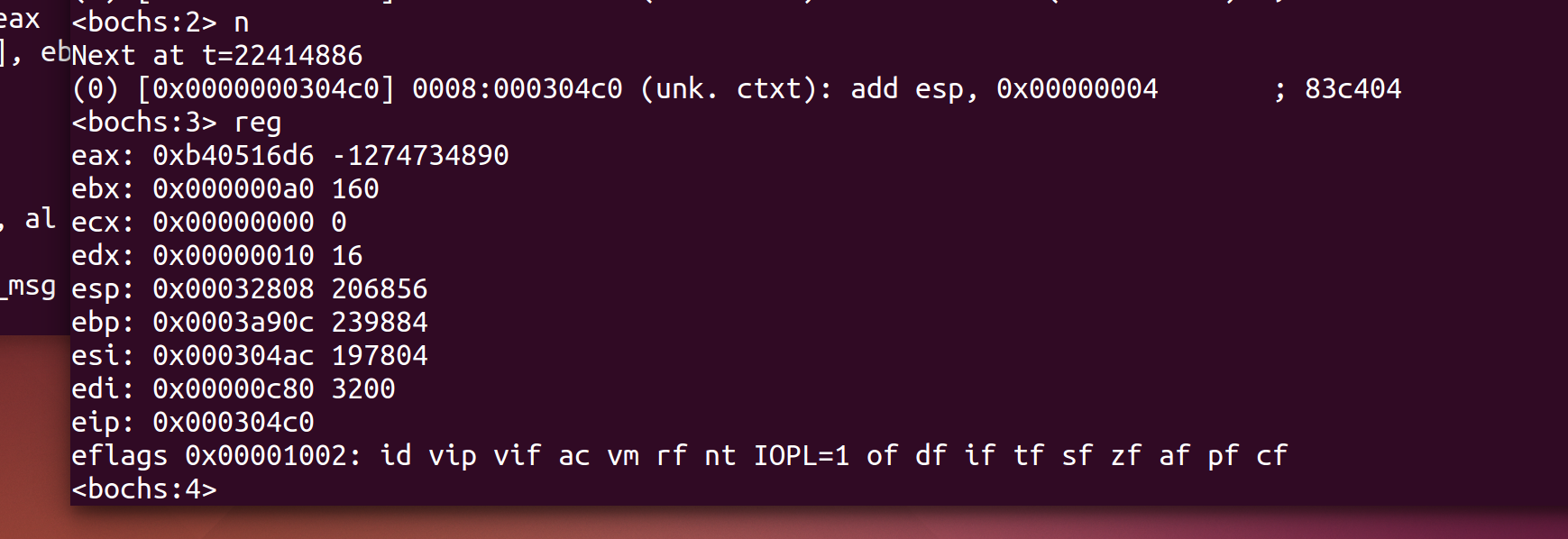
    add esp, 4

    iretd

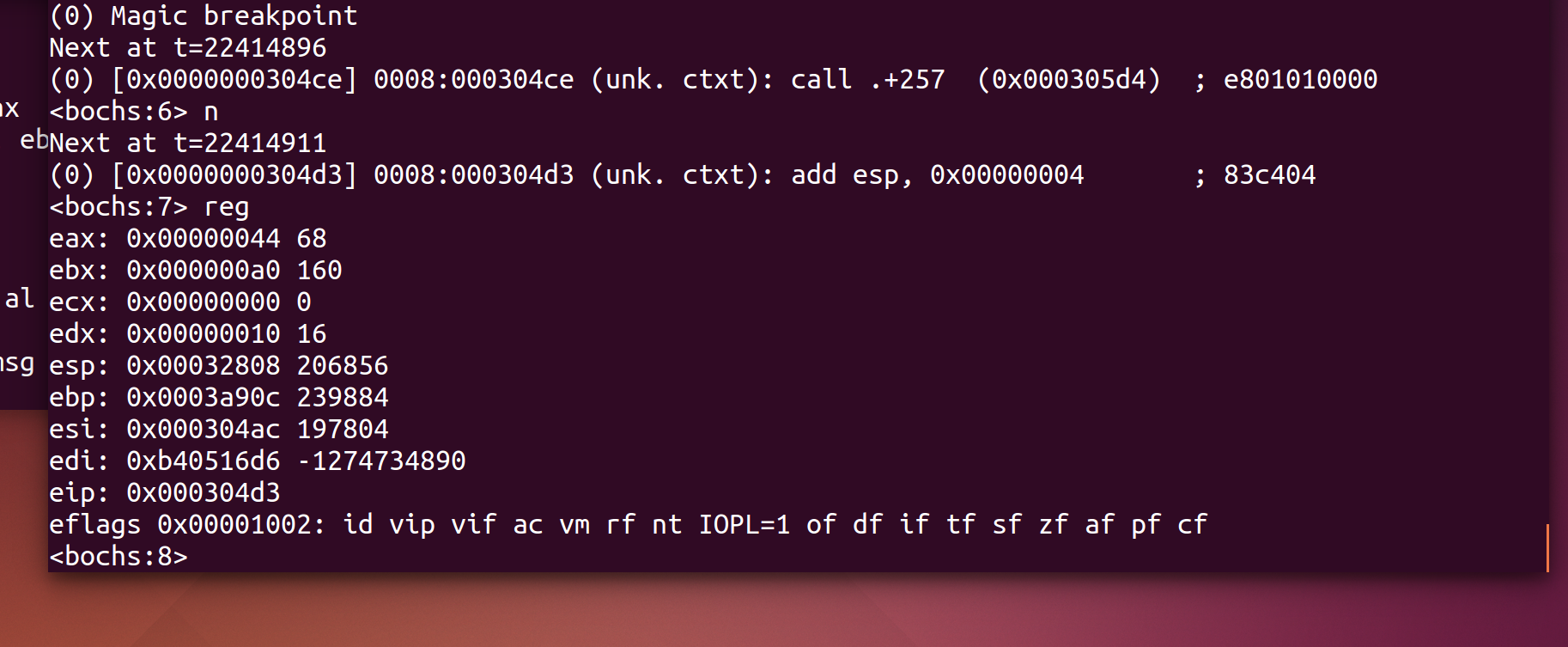
下面下断点调试：



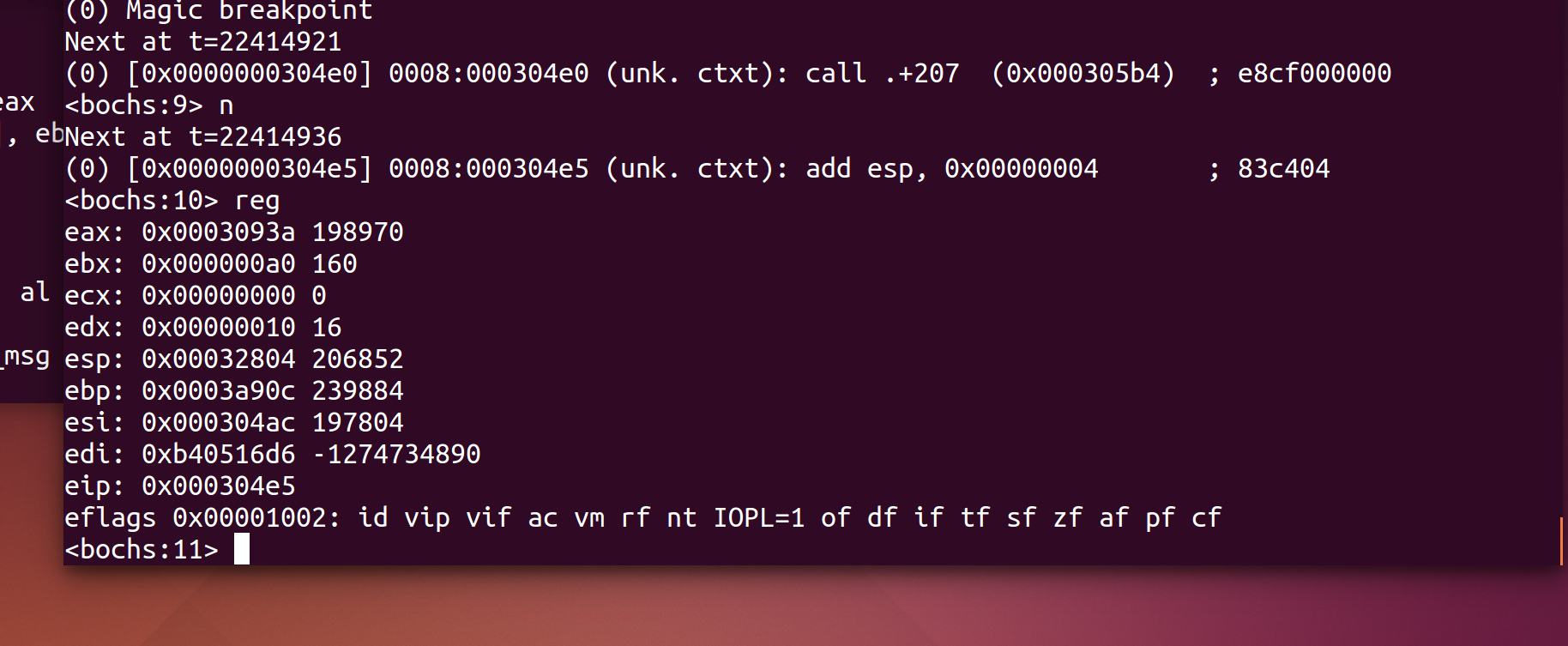
得到保存的校验值：



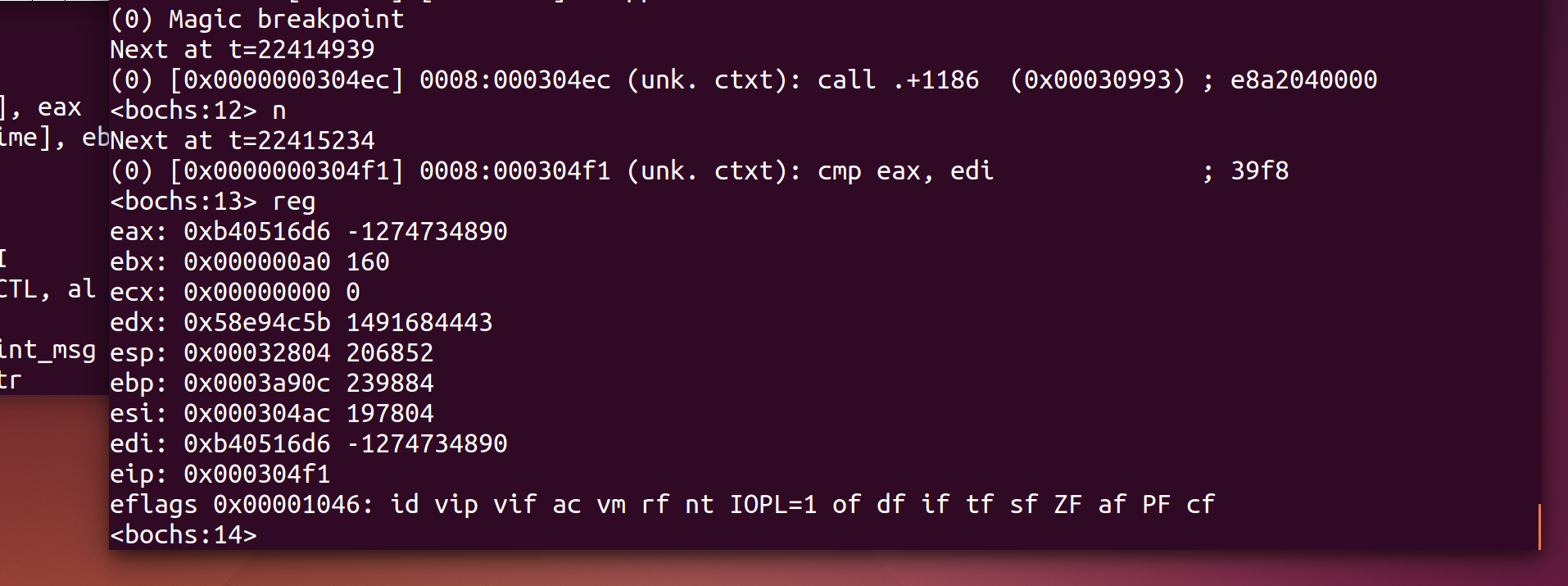
得到代码段长度：



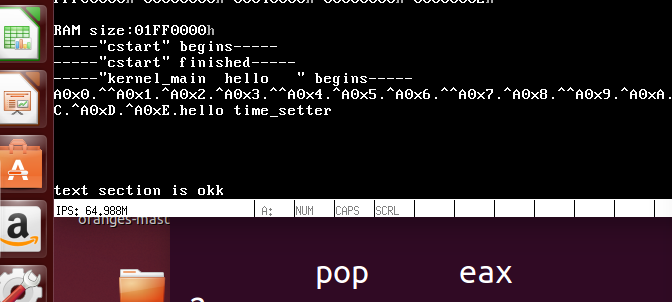
得到代码段起始地址：



重新计算哈希校验值：



最终运行结果：



下面给出完整代码：

hwint00:        ; Interrupt routine for irq 0 (the clock).

    sub esp, 4

    pushad      ; `.

    push    ds  ;  |

    push    es  ;  | 保存原寄存器值

    push    fs  ;  |

    push    gs  ; /

    mov dx, ss

    mov ds, dx

    mov es, dx

    mov esp, StackTop       ; 切到内核栈

    inc byte [gs:0]     ; 改变屏幕第 0 行, 第 0 列的字符

    push    eax

    push    ebx

    push    edi

    mov eax, [count]

    mov ebx, [set\_time]

    inc eax

    cmp eax, ebx

    jnz .2

    call    .1

    db  "hello time\_setter", 10, 0

.1:

    call    disp\_str

    push    eax

    call    get\_pid

    push    eax

    xchg    bx, bx

    call    get\_check\_value

    add esp, 4

    mov edi, eax ;edi用来存放check\_value

    call    get\_pid

    push    eax

    xchg    bx, bx

    call    get\_length

    add esp, 4

    push    eax

    call    get\_pid

    push    eax

    xchg    bx, bx

    call    get\_start

    add esp, 4

    push    eax

    xchg    bx, bx

    call    hash

    cmp eax, edi

    jz  .6

    call    .5

    db  10, 10, 10, 10, 10, "text section is destoried", 10, 10, 10, 10, 10, 10, 10, 0

.5  call    disp\_str

    jmp .8

.6

    call    .7

    db  10, 10, 10, 10, "text section is okk", 10, 10, 10, 10, 0

.7

    call    disp\_str

.8

    pop eax

.2:

    mov [count], eax

    mov [set\_time], ebx

    pop     edi

    pop ebx

    pop eax

    mov al, EOI         ; `. reenable

    out INT\_M\_CTL, al       ; /  master 8259

    push    clock\_int\_msg

    call    disp\_str

    add esp, 4

    mov esp, [p\_proc\_ready] ; 离开内核栈

    lea eax, [esp + P\_STACKTOP]

    mov dword [tss + TSS3\_S\_SP0], eax

    pop gs  ; `.

    pop fs  ;  |

    pop es  ;  | 恢复原寄存器值

    pop ds  ;  |

    popad       ; /

    add esp, 4

    iretd

get\_pid:

        push    ebx

        mov     ebx, [p\_proc\_ready]

        mov     eax, [ebx + 0x5c] ;得到当前进程pid

        pop     ebx

        ret

get\_check\_value:

    push    ebx

    push    ecx

    push    edx

    mov ebx, [info\_addr]

    xor edx, edx

    mov ecx, [esp+4] ;第一个参数是进程的pid

    test    ecx, ecx

    jz  .4

.3:

    add edx, 0xc

    loop    .3

.4:

    ;edx 得到偏移

    add edx, ebx

    mov eax, [edx+8]

    ;eax得到check的值

    pop edx

    pop ecx

    pop ebx

    ret

get\_start:

        push    ebx

        push    ecx

        push    edx

        mov     ebx, [info\_addr]

        xor     edx, edx

        mov     ecx, [esp+4] ;第一个参数是进程的pid

        test    ecx, ecx

        jz      .4

.3:

        add     edx, 0xc

        loop    .3

.4:

        ;edx 得到偏移

        add     edx, ebx

        mov     eax, [edx]

        pop     edx

        pop     ecx

        pop     ebx

        ret

get\_length:

        push    ebx

        push    ecx

        push    edx

        mov     ebx, [info\_addr]

        xor     edx, edx

        mov     ecx, [esp+4] ;第一个参数是进程的pid

        test    ecx, ecx

        jz      .4

.3:

        add     edx, 0xc

        loop    .3

.4:

        ;edx 得到偏移

        add     edx, ebx

        mov     eax, [edx+4]

        pop     edx

        pop     ecx

        pop     ebx

        ret