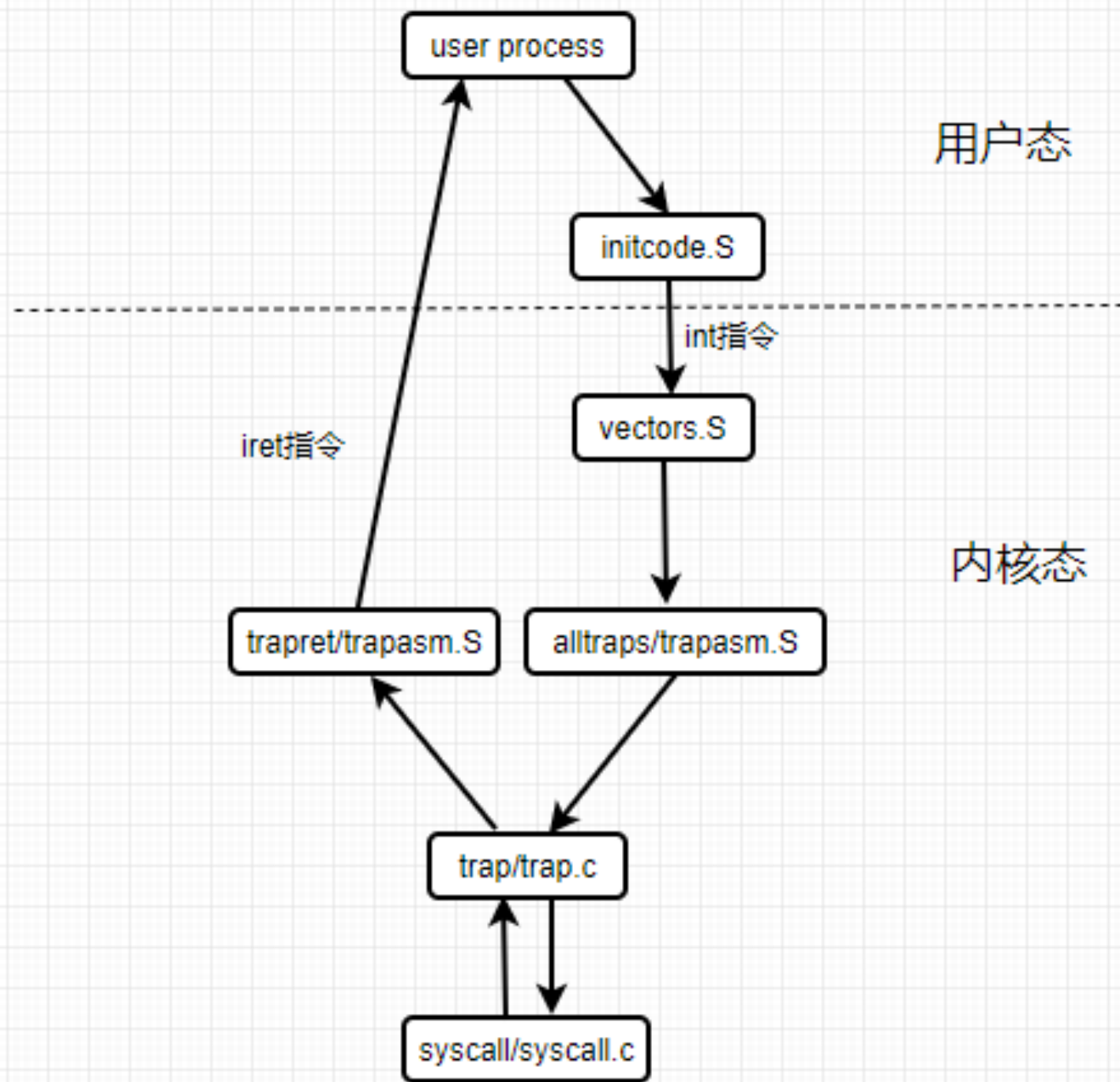


实例分析第二部分

黄上京、余成、袁艺、张心茹



问题1

- `initcode.S`这段用户态代码是如何向内核传递参数的？

initcode.S

```
1  # Initial process execs /init.
2  # This code runs in user space.
3
4  #include "syscall.h"
5  #include "traps.h"
6
7
8  # exec(init, argv)
9  .globl start
10 start:
11     pushl $argv
12     pushl $init
13     pushl $0 // where caller pc would be
14     movl $SYS_exec, %eax
15     int $T_SYSCALL
16
```

```
17 # for(;;) exit();
18 exit:
19     movl $SYS_exit, %eax
20     int $T_SYSCALL
21     jmp exit
22
23 # char init[] = "/init\0";
24 init:
25     .string "/init\0"
26
27 # char *argv[] = { init, 0 };
28 .p2align 2
29 argv:
30     .long init
31     .long 0
```

T_SYSCALL的参数——系统调用号

- initcode.S中将\$SYS_exec写入%eax
- alltrap将%eax压入内核栈

sys_exec的参数

- initcode.S将\$argv, \$init, \$0压入用户栈
- sys_exec执行时直接从用户栈获取参数

问题2

- initcode.S代码里，int指令的作用？

- x86 有四个特权级，从0（特权最高）编号到3（特权最低）。在实际使用中，大多数的操作系统都使用两个特权级，0和3被称为内核模式和用户模式。当前执行指令的特权级存在于%cs寄存器中的CPL域中。
- 在x86中，中断处理程序的入口在中断描述符表（IDT）中被定义。这个表有256个表项，每一个都提供了相应的 %cs 和 %eip。

一个程序要在 x86 上进行一个系统调用，它需要调用 `int n` 指令，这里 `n` 就是 IDT 的索引。`int` 指令进行下面一些步骤：

- 从 IDT 中获得第 `n` 个描述符，`n` 就是 `int` 的参数。
- 检查 `%cs` 的域 $CPL \leq DPL$ ，`DPL` 是描述符中记录的特权级。
- 如果目标段选择符的 $PL < CPL$ ，就在 CPU 内部的寄存器中保存 `%esp` 和 `%ss` 的值。
- 从一个任务段描述符中加载 `%ss` 和 `%esp`。
- 将保存在 CPU 内部寄存器中的 `%ss` 压栈。
- 将保存在 CPU 内部寄存器中的 `%esp` 压栈。
- 将 `%eflags` 压栈。
- 将 `%cs` 压栈。
- 将 `%eip` 压栈。
- 清除 `%eflags` 的一些位。
- 设置 `%cs` 和 `%eip` 为描述符中的值。

因此， initcode.S中int指令的作用：

- 完成了从用户态向内核态的转换
 - 依据指定中断描述符建立了内核栈，加载了 %cs 和 %eip
 - 向内核栈压入了用户栈及进程的部分信息

问题3

- int指令执行完后，栈指针esp指向的用户栈还是内核栈？
- 内核栈发生了什么变化？
- int指令执行完后，控制转移到了什么地方？为什么？

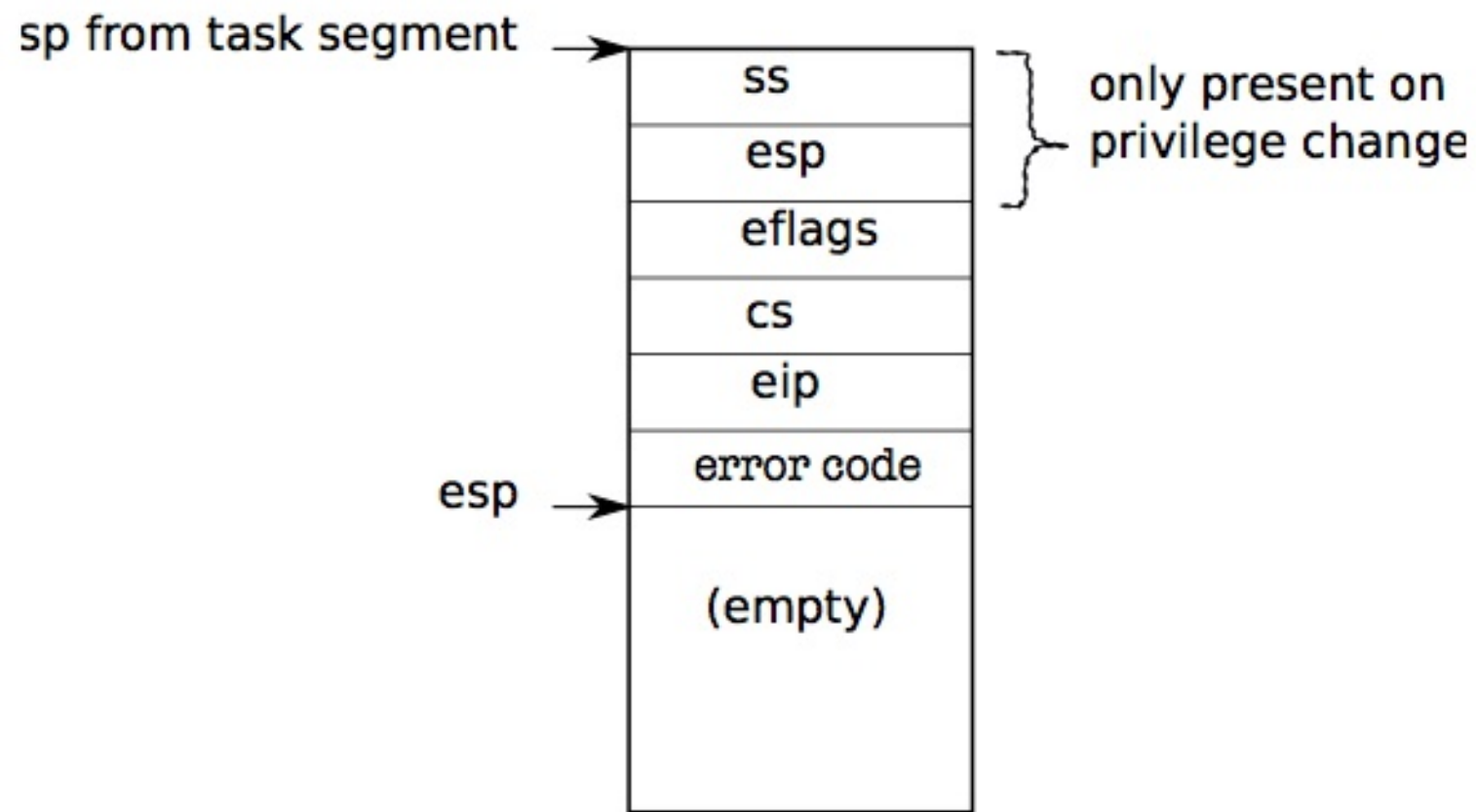



Figure 3-1. Kernel stack after an int instruction.

- xv6使用一个perl脚本来产生IDT表项指向的中断处理函数入口，并跳转到alltraps/trapasm.S

 vectors.pl

```
1  #!/usr/bin/perl -w
2
3  # Generate vectors.S, the trap/interrupt entry points.
4  # There has to be one entry point per interrupt number
5  # since otherwise there's no way for trap() to discover
6  # the interrupt number.
7
8  print "# generated by vectors.pl - do not edit\n";
9  print "# handlers\n";
10 print ".globl alltraps\n";
11 for(my $i = 0; $i < 256; $i++){
12     print ".globl vector$i\n";
13     print "vector$i:\n";
14     if(!($i == 8 || ($i >= 10 && $i <= 14) || $i == 17)){
15         print "    pushl \ $0\n";
16     }
17     print "    pushl \ $$i\n";
18     print "    jmp alltraps\n";
19 }
```

控制转移到alltrap/trapasm.S

ASM trapasm.S

```
1  #include "mmu.h"
2
3  # vectors.S sends all traps here.
4  .globl alltraps
5  alltraps:
6  # Build trap frame.
7  pushl %ds
8  pushl %es
9  pushl %fs
10 pushl %gs
11 pushal
12
```

```
13 # Set up data segments.
14 movw $(SEG_KDATA<<3), %ax
15 movw %ax, %ds
16 movw %ax, %es
17
18 # Call trap(tf), where tf=%esp
19 pushl %esp
20 call trap
21 addl $4, %esp
22
```

alltrap保存上下文并调用c函数trap/trap.c

问题4

- 执行系统调用时，trapframe中的trapno的值是多少？
- 在什么地方压栈的？

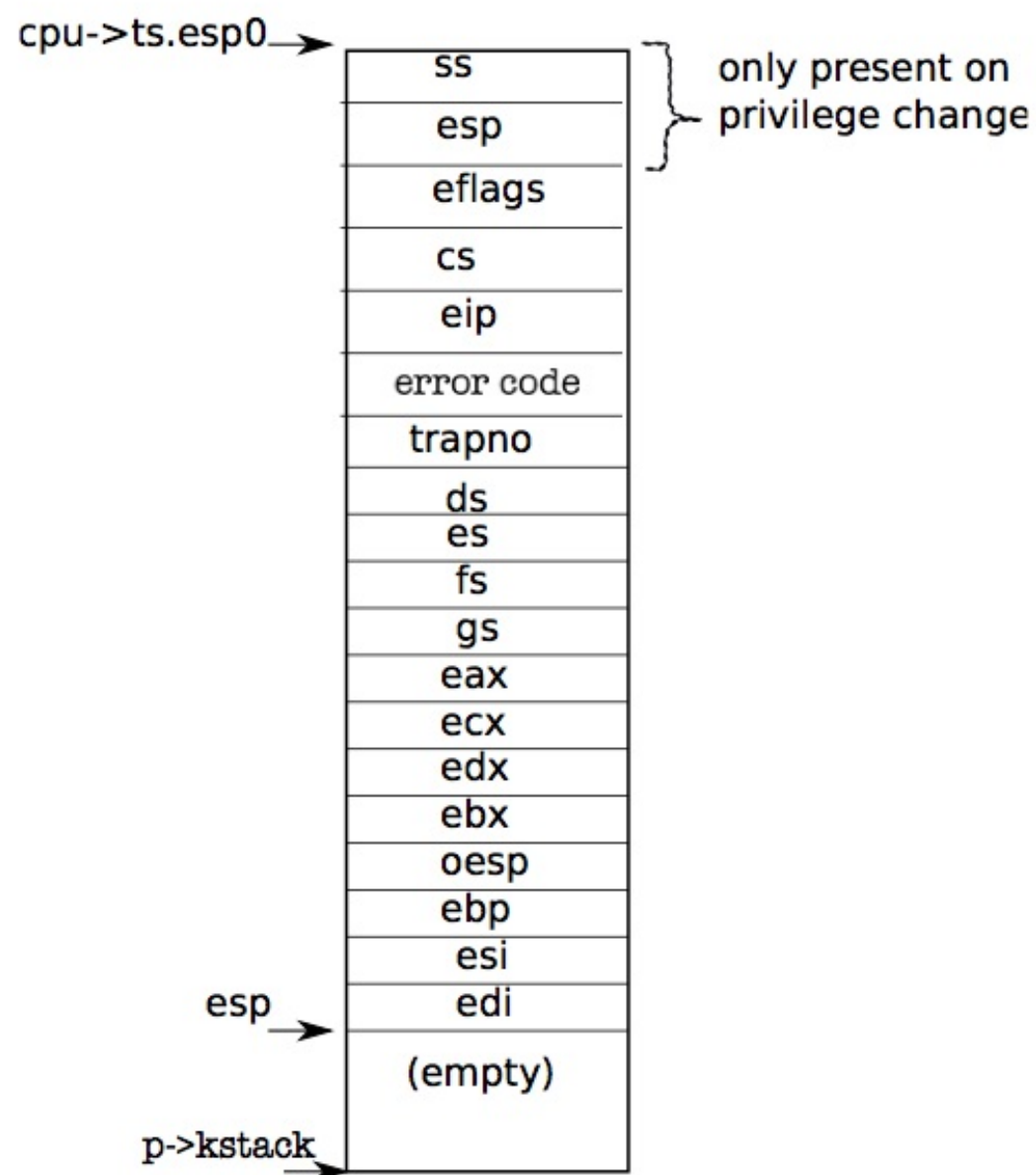


Figure 3-2. The trapframe on the kernel stack

trapno的值是\$T_SYSCALL (宏定义为64)

- trapno的值是由vector64压入内核栈的

```
🐼 vectors.pl
1  #!/usr/bin/perl -w
2
3  # Generate vectors.S, the trap/interrupt entry points.
4  # There has to be one entry point per interrupt number
5  # since otherwise there's no way for trap() to discover
6  # the interrupt number.
7
8  print "# generated by vectors.pl - do not edit\n";
9  print "# handlers\n";
10 print ".globl alltraps\n";
11 for(my $i = 0; $i < 256; $i++){
12     print ".globl vector$i\n";
13     print "vector$i:\n";
14     if(!($i == 8 || ($i >= 10 && $i <= 14) || $i == 17)){
15         print "    pushl \\\$0\n";
16     }
17     print "    pushl \\\$i\n";
18     print "    jmp alltraps\n";
19 }
```

问题5

- 汇编代码在调用c代码时，是如何向c传递参数的？以trap函数的tf参数为例子。

alltrap建立trap frame (内陷帧)

ASM trapasm.S

```
1  #include "mmu.h"
2
3  # vectors.S sends all traps here.
4  .globl alltraps
5  alltraps:
6  # Build trap frame.
7  pushl %ds
8  pushl %es
9  pushl %fs
10 pushl %gs
11 pushal
12
```

建立数据段并调用trap函数

```
13      # Set up data segments.
14      movw $(SEG_KDATA<<3), %ax
15      movw %ax, %ds
16      movw %ax, %es
17
18      # Call trap(tf), where tf=%esp
19      pushl %esp
20      call trap
21      addl $4, %esp
22
```

c函数trap根据%esp找到trap frame

```
36 void
37 trap(struct trapframe *tf)
38 {
39     if(tf->trapno == T_SYSCALL){
40         if(myproc()->killed)
41             exit();
42         myproc()->tf = tf;
43         syscall();
44         if(myproc()->killed)
45             exit();
46         return;
47     }
```

(tf = %esp)

问题6

- 系统调用的返回值是如何从内核态返回到用户态的？

syscall返回值写入到trap frame的eax域

```
131 void
132 syscall(void)
133 {
134     int num;
135     struct proc *curproc = myproc();
136
137     num = curproc->tf->eax;
138     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
139         curproc->tf->eax = syscalls[num]();
140     } else {
141         cprintf("%d %s: unknown sys call %d\n",
142             curproc->pid, curproc->name, num);
143         curproc->tf->eax = -1;
144     }
145 }
```

trapret将返回值弹回寄存器，返回用户态

```
18     # Call trap(tf), where tf=%esp
19     pushl %esp
20     call trap
21     addl $4, %esp
22
23     # Return falls through to trapret...
24     .globl trapret
25 trapret:
26     popal
27     popl %gs
28     popl %fs
29     popl %es
30     popl %ds
31     addl $0x8, %esp # trapno and errcode
32     iret
```


问题7

- initcode.S执行的是哪个syscall？
- 是通过什么方式告知内核的？

initcode.S执行的是SYS_exec

```
8   # exec(init, argv)
9   .globl start
10  start:
11      pushl $argv
12      pushl $init
13      pushl $0 // where caller pc would be
14      movl $SYS_exec, %eax
15      int $T_SYSCALL
```

syscall识别系统调用号

```
131 void
132 syscall(void)
133 {
134     int num;
135     struct proc *curproc = myproc();
136
137     num = curproc->tf->eax;
138     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
139         curproc->tf->eax = syscalls[num]();
140     } else {
141         cprintf("%d %s: unknown sys call %d\n",
142             curproc->pid, curproc->name, num);
143         curproc->tf->eax = -1;
144     }
145 }
```

问题8

- trap函数执行完syscall后，return到哪里？

trapasm.S

```
18  # Call trap(tf), where tf=%esp
19  pushl %esp
20  call trap
21  addl $4, %esp
22
23  # Return falls through to trapret...
24  .globl trapret
25  trapret:
26  popal
27  popl %gs
28  popl %fs
29  popl %es
30  popl %ds
31  addl $0x8, %esp # trapno and errcode
32  iret
```

trap

```
36  void
37  trap(struct trapframe *tf)
38  {
39      if(tf->trapno == T_SYSCALL){
40          if(myproc()->killed)
41              exit();
42          myproc()->tf = tf;
43          syscall();
44          if(myproc()->killed)
45              exit();
46          return;
47      }
```

trap执行完后返回到trapasm.S, 由trapret恢复上下文并返回用户态