homework 2 & 3 solution

Du Jiajun

2024.10.25

homework 2

```
150  struct trapframe {
151    // registers as pushed by pusha
152    uint edi;
153    uint esi;
154    uint ebp;
155    uint oesp;      // useless & ignored
156    uint ebx;
157    uint edx;
158    uint ecx;
159    uint eax;
160
161    // rest of trap frame
162    ushort gs;
163    ushort padding1;
164    ushort fs;
165    ushort padding2;
166    ushort es;
167    ushort padding3;
168    ushort ds;
169    ushort padding4;
170    uint trapno;
171
172    // below here defined by x86 hardware
173    uint err;
174    uint eip;
175    ushort cs;
176    ushort padding5;
177    uint eflags;
178
179    // below here only when crossing rings, such as from user to kernel
180    uint esp;
181    ushort ss;
182    ushort padding6;
183  };
```

在发生中断、异常、syscall 后，内核需要捕捉关于被中断上下文、产生原因的各种信息，根据这些信息来做出相应的处理，这些信息在 xv6 中就是定义为了 struct trapframe，所以在内核处理中断之前需要硬件、软件配合把这个结构体构建出来。

中断产生后，CPU 会：

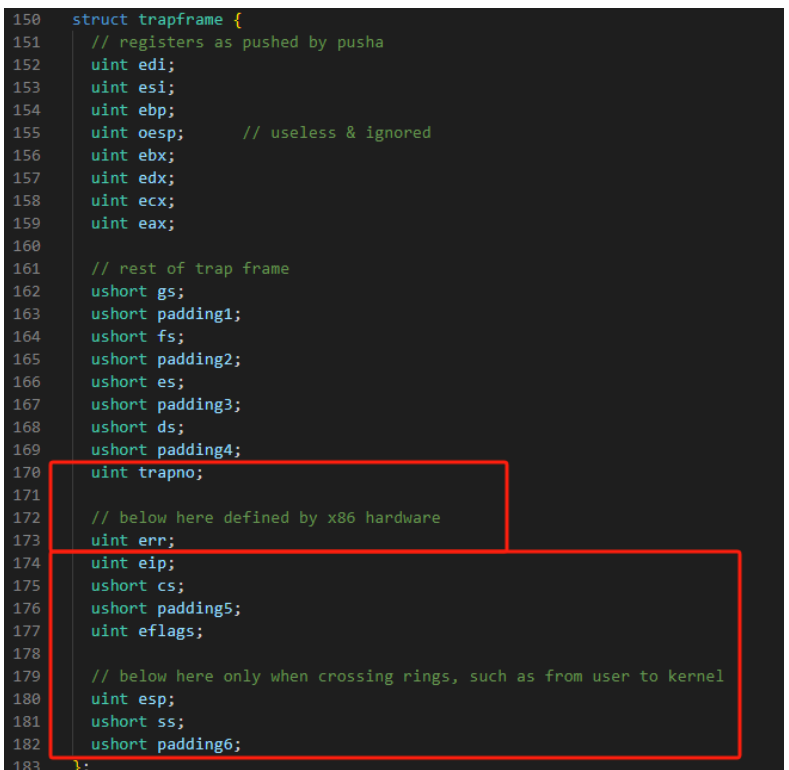
1. 临时将 SS、ESP、EFLAGS、CS、EIP 寄存器的内容保存；
2. 将 TSS 中的 segment selector（ss0）、栈顶指针（esp0）存放到 SS、ESP 寄存器中，切换到内核栈；
3. 将保存的 SS、ESP、EFLAGS、CS、EIP 的寄存器值压到内核栈中；
4. 如果发生了错误，则向栈顶压入 error code。因为此处发生的是除 0 异常，Divide Error Exception 没有 Exception Error Code，所以 CPU 并不会向内核栈压入 error code；
5. 从 idtr 寄存器指向的 IDT 中找到对应的 segment selector（即 SEG_KCODE），将其指向的代码段地址加载到 CS 寄存器，更改 EIP 寄存器为代码段中对应 ISR 第一条指令的地址。对于除 0 异常来说，EIP 指向了 vector0 的第一条指令的地址；
6. 如果 trap 产生的原因是中断，还会清理 EFLAGS 中的 IF flag，关闭中断；

[1] x86 中断处理过程：Intel® 64 and IA-32 Architectures Software Developer's Manual Volume1 6.5.1

```
151   // registers as pushed by pusha
152   uint edi;
153   uint esi;
154   uint ebp;
155   uint oesp;      // useless & ignored
156   uint ebx;
157   uint edx;
158   uint ecx;
159   uint eax;
160
161   // rest of trap frame
162   ushort gs;
163   ushort padding1;
164   ushort fs;
165   ushort padding2;
166   ushort es;
167   ushort padding3;
168   ushort ds;
169   ushort padding4;
170   uint trapno;
171
172   // below here defined by x86 hardware
173   uint err;
174   uint eip;
175   ushort cs;
176   ushort padding5;
177   uint eflags;
178
179   // below here only when crossing rings, such as from user to kernel
180   uint esp;
181   ushort ss;
182   ushort padding6;
```

在 vector0 函数里，手动向内核栈顶压入 0 来代表除 0 异常的 error code，并且压入 trapnumber，然后调用到 alltraps。

```
# Divide Error Exception
vector0:
  pushl $0
  pushl $0
  jmp alltraps
```



```
struct trapframe {
  // registers as pushed by pusha
  uint edi;
  uint esi;
  uint ebp;
  uint oesp;      // useless & ignored
  uint ebx;
  uint edx;
  uint ecx;
  uint eax;

  // rest of trap frame
  ushort gs;
  ushort padding1;
  ushort fs;
  ushort padding2;
  ushort es;
  ushort padding3;
  ushort ds;
  ushort padding4;
  uint trapno;

  // below here defined by x86 hardware
  uint err;
  uint eip;
  ushort cs;
  ushort padding5;
  uint eflags;

  // below here only when crossing rings, such as from user to kernel
  uint esp;
  ushort ss;
  ushort padding6;
};
```

```
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
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
```

1. 保存寄存器现场，以便后续恢复现场：

   1.1 向内核栈压入 %ds、%es、%fs、%gs

   1.2 压入其他通用寄存器

   此时内核栈顶构成了一个完整的 struct trapframe

2. 为了执行内核代码，设置 %ds（内核数据段）、%es；
3. 调用 trap 函数，参数的入栈和出栈需要调用者管理：

   3.1 压入栈顶指针 %esp 作为指针参数

   3.2 call trap，处理除 0 异常

   3.3 参数出栈

4. 正常情况下，因为没有 ret，所以会继续执行 trapret，恢复现场，继续执行用户程序

在 trap 函数中，xv6 会先后处理 syscall、中断、异常：

```
37  void
38  trap(struct trapframe *tf)
39  {
40    if(tf->trapno == T_SYSCALL){
41      if(myproc()->killed)
42        exit();
43      myproc()->tf = tf;
44      syscall();
45      if(myproc()->killed)
46        exit();
47      return;
48    }
```

```
50    switch(tf->trapno){
51    case T_IRQ0 + IRQ_TIMER:
52      if(cpuid() == 0){
53        acquire(&tickslock);
54        ticks++;
55        wakeup(&ticks);
56        release(&tickslock);
57      }
58      lapiceoi();
59      break;
60    case T_IRQ0 + IRQ_IDE:
61      ideintr();
62      lapiceoi();
63      break;
64    case T_IRQ0 + IRQ_IDE+1:
65      // Bochs generates spurious IDE1 interrupts.
66      break;
67    case T_IRQ0 + IRQ_KBD:
68      kbdintr();
69      lapiceoi();
70      break;
71    case T_IRQ0 + IRQ_COM1:
72      uartintr();
73      lapiceoi();
74      break;
75    case T_IRQ0 + 7:
76    case T_IRQ0 + IRQ_SPURIOUS:
77      cprintf("cpu%d: spurious interrupt at %x:%x\n",
78              cpuid(), tf->cs, tf->eip);
79      lapiceoi();
80      break;
```

```
83    default:
84      if(myproc() == 0 || (tf->cs&3) == 0){
85        // In kernel, it must be our mistake.
86        cprintf("unexpected trap %d from cpu %d eip %x (cr2=0x%x)\n",
87                tf->trapno, cpuid(), tf->eip, rcr2());
88        panic("trap");
89      }
90      // In user space, assume process misbehaved.
91      cprintf("pid %d %s: trap %d err %d on cpu %d "
92              "eip 0x%x addr 0x%x--kill proc\n",
93              myproc()->pid, myproc()->name, tf->trapno,
94              tf->err, cpuid(), tf->eip, rcr2());
95      myproc()->killed = 1;
96    }
```

xv6 对发生除 0 异常的用户程序会直接 kill，并且会调用 exit()函数，释放资源（关闭文件等等），最终调用到 sched()函数切换到其他程序继续执行，不会向下继续执行，不会返回到发生除 0 异常的用户程序。

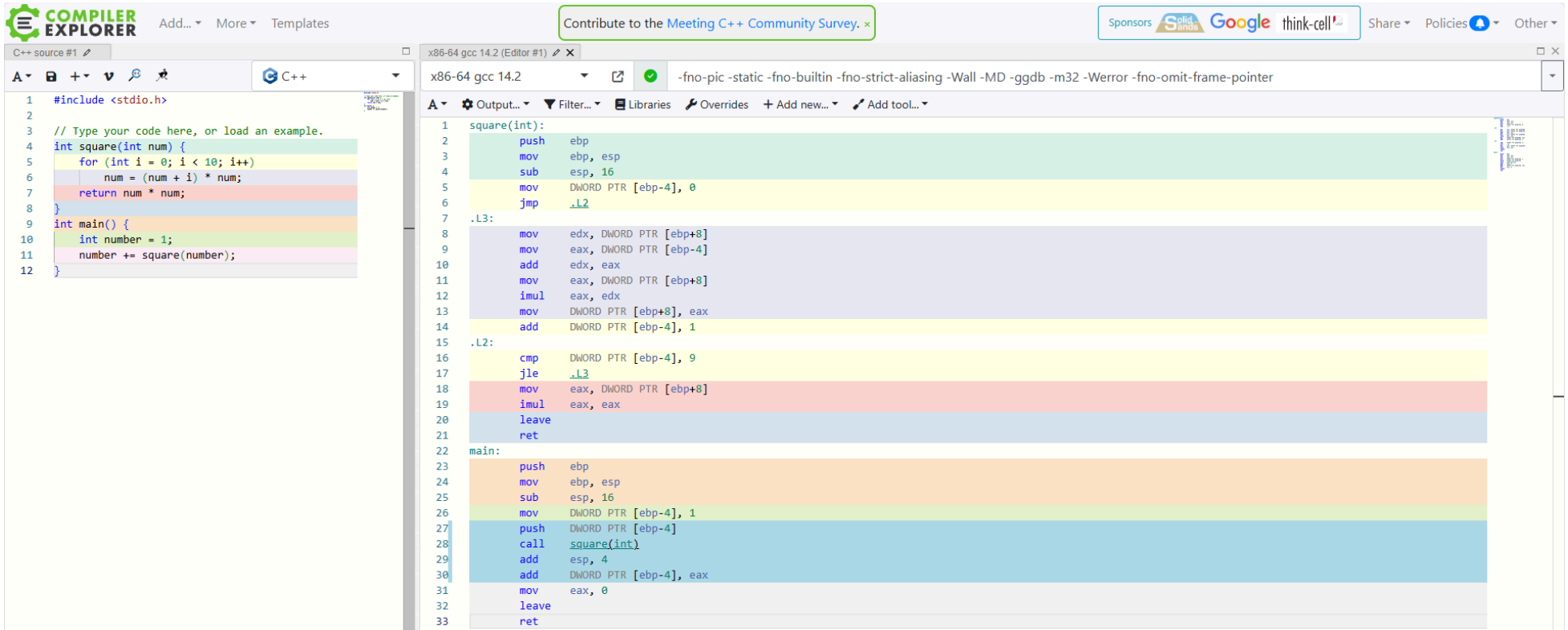
```
82  //PAGEBREAK: 13
83  default:
84    if(myproc() == 0 || (tf->cs&3) == 0){
85      // In kernel, it must be our mistake.
86      cprintf("unexpected trap %d from cpu %d eip %x (cr2=0x%x)\n",
87              tf->trapno, cpuid(), tf->eip, rcr2());
88      panic("trap");
89    }
90    // In user space, assume process misbehaved.
91    cprintf("pid %d %s: trap %d err %d on cpu %d "
92            "eip 0x%x addr 0x%x--kill proc\n",
93            myproc()->pid, myproc()->name, tf->trapno,
94            tf->err, cpuid(), tf->eip, rcr2());
95    myproc()->killed = 1;
96  }
97
98  // Force process exit if it has been killed and is in user space.
99  // (If it is still executing in the kernel, let it keep running
100 // until it gets to the regular system call return.)
101 if(myproc() && myproc()->killed && (tf->cs&3) == DPL_USER)
102   exit();
```

```
227 void
228 exit(void)
229 {
230   struct proc *curproc = myproc();
231   struct proc *p;
232   int fd;
233
234   if(curproc == initproc)
235     panic("init exiting");
236
237   // Close all open files.
238   for(fd = 0; fd < NOFILE; fd++){
239     if(curproc->ofile[fd]){
240       fileclose(curproc->ofile[fd]);
241       curproc->ofile[fd] = 0;
242     }
243   }
244
245   begin_op();
246   iput(curproc->cwd);
247   end_op();
248   curproc->cwd = 0;
249
250   acquire(&ptable.lock);
251
252   // Parent might be sleeping in wait().
253   wakeup1(curproc->parent);
254
255   // Pass abandoned children to init.
256   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
257     if(p->parent == curproc){
258       p->parent = initproc;
259       if(p->state == ZOMBIE)
260         wakeup1(initproc);
261     }
262   }
263
264   // Jump into the scheduler, never to return.
265   curproc->state = ZOMBIE;
266   sched();
267   panic("zombie exit");
268 }
```

question 1

疑惑：

如果是正常系统调用，trap 函数返回后，会将 esp 向上移动四字节恢复到原来的值，但保存 esp 的意义是什么呢，明明是通过计算恢复的，并没有用保存的值去赋值。



[2] : compiler explorer

homework 3

1. 配置环境、动手实践
2. 作业内容不难，难点在于配置环境
3. 言之有理即可