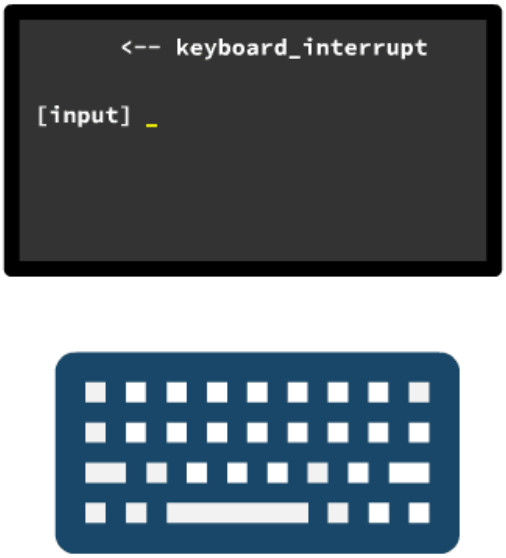
你的键盘是什么时候生效的？

Original 闪客 低并发编程 2021-12-29 16:30

收录于合集
#操作系统源码

43个

当你的计算机刚刚启动时，你按下键盘是不生效的，但是过了一段时间后，再按下键盘就有效果了。



那我们今天就来刨根问底一下，**到底过了多久之后，按下键盘才有效果呢？**

当然首先你得知道，按下键盘后会触发中断，CPU 收到你的键盘中断后，根据中断号，寻找由操作系统写好的键盘中断处理程序。

中断的原理和过程不了解的，可以看我的文章，认认真真的聊聊中断

这个中断处理程序会把你的键盘码放入一个队列中，由相应的用户程序或内核程序读取，并显示在控制台，或者其他用途，这就代表你的键盘生效了。

不过放宽心，我们不展开讲这个中断处理程序以及用户程序读取键盘码后的处理细节，我们把关注点放在，究竟是**“什么时候”**，按下键盘才会有这个效果。

我们以 **Linux 0.11** 源码为例，发现进入内核的 main 函数后不久，有这样一行代码。

```
void main(void) {
    ...
    trap_init();
    ...
}
```

看到这个方法的全部代码后，你可能会会心一笑，也可能一脸懵逼。

```
void trap_init(void) {
    int i;
    set_trap_gate(0,&divide_error);
    set_trap_gate(1,&debug);
    set_trap_gate(2,&nmi);
    set_system_gate(3,&int3);   /* int3-5 can be called from all */
    set_system_gate(4,&overflow);
    set_system_gate(5,&bounds);
    set_trap_gate(6,&invalid_op);
    set_trap_gate(7,&device_not_available);
    set_trap_gate(8,&double_fault);
    set_trap_gate(9,&coprocessor_segment_overrun);
    set_trap_gate(10,&invalid_TSS);
    set_trap_gate(11,&segment_not_present);
    set_trap_gate(12,&stack_segment);
    set_trap_gate(13,&general_protection);
    set_trap_gate(14,&page_fault);
    set_trap_gate(15,&reserved);
    set_trap_gate(16,&coprocessor_error);
    for (i=17;i<48;i++)
        set_trap_gate(i,&reserved);
    set_trap_gate(45,&irq13);
    set_trap_gate(39,&parallel_interrupt);
}
```

这啥玩意？这么多 **set_xxx_gate**。

有密集恐惧症的话，绝对看不下去这个代码，所以我就给他简化一下。

把相同功能的去掉。

```
void trap_init(void) {
    int i;
    // set 了一堆 trap_gate
    set_trap_gate(0, &divide_error);
    ...
    // 又 set 了一堆 system_gate
    set_system_gate(45, &bounds);
    ...
    // 又又批量 set 了一堆 trap_gate
    for (i=17;i<48;i++)
        set_trap_gate(i, &reserved);
    ...
}
```

这就简单多了，我们一块一块看。

首先我们看 **set_trap_gate** 和 **set_system_gate** 这俩货，发现了这么几个宏定义。

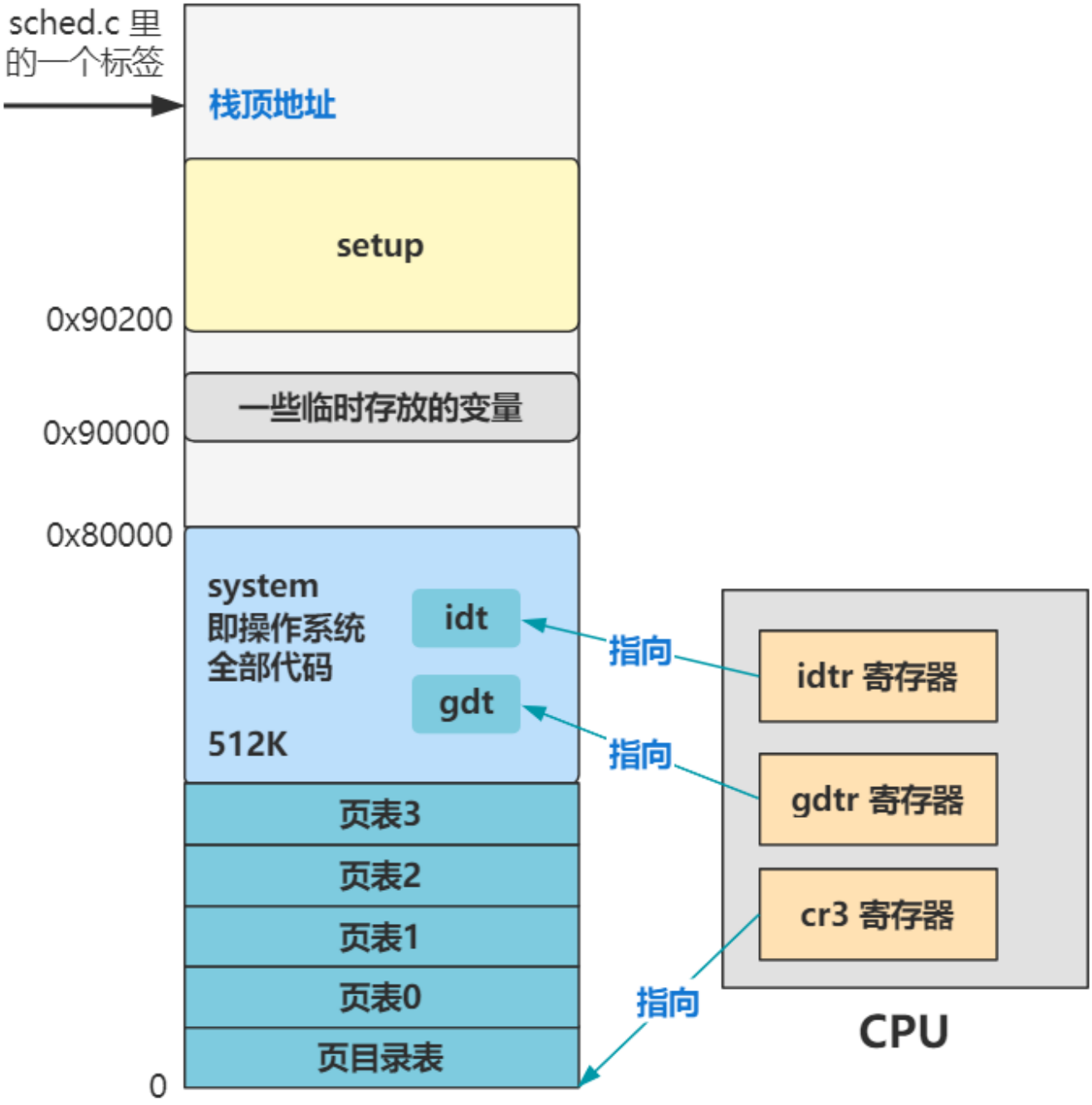
```
#define _set_gate(gate_addr,type,dpl,addr) \
__asm__ ("movw %%dx,%%ax\n\t" \
    "movw %0,%%dx\n\t" \
    "movl %%eax,%1\n\t" \
    "movl %%edx,%2" \
    : \
    : "i" ((short) (0x8000+(dpl<<13)+(type<<8))), \
    "o" (*((char *) (gate_addr))), \
    "o" (*(4+(char *) (gate_addr))), \
    "d" ((char *) (addr)),"a" (0x00080000))

#define set_trap_gate(n,addr) \
    _set_gate(&idt[n],15,0,addr)

#define set_system_gate(n,addr) \
    _set_gate(&idt[n],15,3,addr)
```

别怕，我也看不懂。

不过这俩都是最终指向了相同的另一个宏定义 **_set_gate**，说明是有共性的。

啥共性呢？我直接说吧，那段你完全看不懂的代码，是将汇编语言嵌入到 c 语言了，这种内联汇编的格式非常恶心，所以我也不想搞懂它，最终的效果就是**在中断描述符表中插入了一个中断描述符。**

中断描述符表还记得吧，英文叫 idt。



这段代码就是往这个 idt 表里一项一项地写东西，其对应的中断号就是第一个参数，中断处理程序就是第二个参数。

产生的效果就是，之后如果来一个中断后，CPU 根据其中断号，就可以到这个中断描述符表 idt 中找到对应的中断处理程序了。

比如这个。

```
set_trap_gate(0,&divide_error);
```

就是设置 **0 号中断**，对应的中断处理程序是 **divide_error**。

等 CPU 执行了一条除零指令的时候，会从硬件层面发起一个 0 号异常中断，然后执行由我们操作系统定义的 divide_error 也就是除法异常处理程序，执行完之后再返回。

再比如这个。

```
set_system_gate(5,&overflow);
```

就是设置 5 号中断，对应的中断处理程序是 overflow，是边界出错中断。

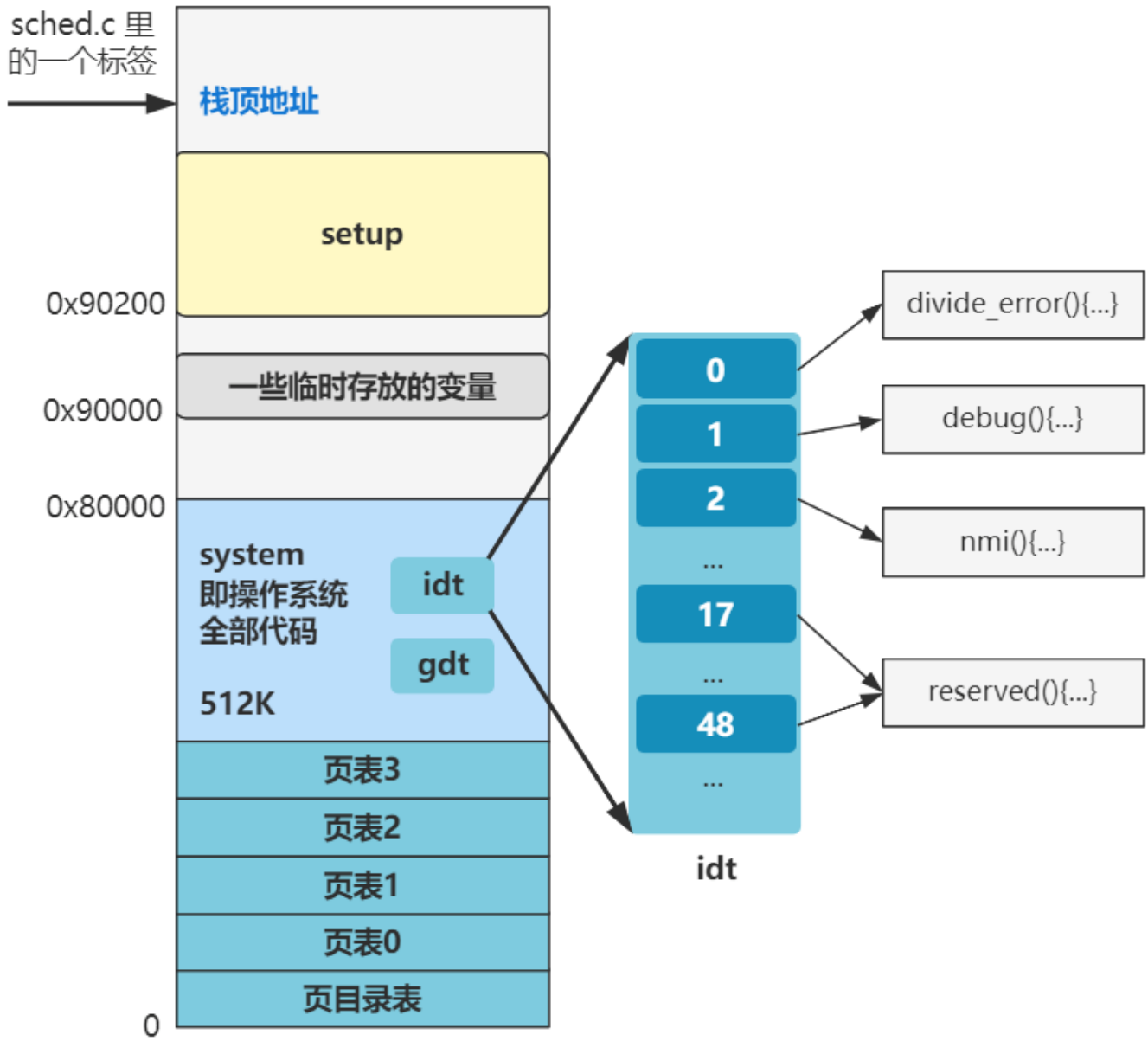
TIPS：这个 system 与 trap 的区别仅仅在于，设置的中断描述符的特权级不同，前者是 0（内核态），后者是　3（用户态），这块展开将会是非常严谨的、绕口的、复杂的特权级相关的知识，不明白的话先不用管，就理解为都是设置一个中断号和中断处理程序的对应关系就好了。

再往后看，批量操作这里。

```
void trap_init(void) {
    ...
    for (i=17;i<48;i++)
        set_trap_gate(i,&reserved);
    ...
}
```

17 到 48 号中断都批量设置为了 **reserved** 函数，这是暂时的，后面各个硬件初始化时要重新设置好这些中断，把暂时的这个给覆盖掉，此时你留个印象。

所以整段代码执行下来，内存中那个 idt 的位置会变成如下的样子。



好了，我们看到了设置中断号与中断处理程序对应的地方，那这行代码过去后，键盘好使了么?

NO

键盘产生的中断的中断号是 **0x21**，此时这个中断号还仅仅对应着一个临时的中断处理程序 &reserved，我们接着往后看。

在这行代码往后几行，还有这么一行代码。

```
void main(void) {
    ...
    trap_init();
    ...
    tty_init();
    ...
}

void tty_init(void) {
    rs_init();
    con_init();
}

void con_init(void) {
    ...
    set_trap_gate(0x21,&keyboard_interrupt);
    ...
}
```

我省略了大量的代码，只保留了我们关心的。

注意到 trap_init 后有个 **tty_init**，最后根据调用链，会调用到一行添加 0x21 号中断处理程序的代码，就是刚刚熟悉的 **set_trap_gate**。

而后面的 **keyboard_interrupt** 根据名字也可以猜出，就是键盘的中断处理程序嘛！

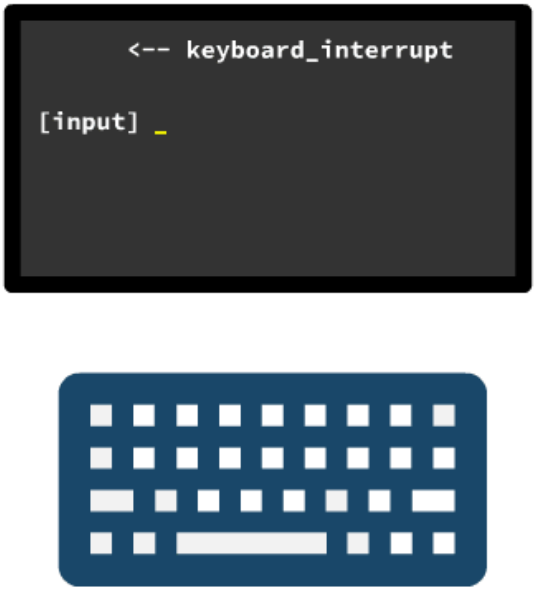
好了，那我们终于找到大案了，就是从这一行代码开始，我们的键盘生效了！

没错，不过还有点小问题，不过不重要，就是我们现在的中断处于**禁用状态**，不论是键盘中断还是其他中断，通通都不好使。

而 main 方法继续往下读，还有一行这个东西。

```
void main(void) {
    ...
    trap_init();
    ...
    tty_init();
    ...
    sti();
    ...
}
```

sti 最终会对应一个同名的汇编指令 sti，表示**允许中断**。所以这行代码之后，键盘才真正开始生效！



动画酷不酷？好啦，今天的文章就到这里了，中断的原理和细节，就看我之前的文章，认认真真的聊聊中断。

键盘处理的具体流程，可以跟着我今天的代码深入进去看看哟，Linux 0.11 里还是很简单的。

本文可以当做 你管这破玩意叫操作系统源码 系列文章的第 14 回。

为了让不追更系列的读者也能很方便阅读并学到东西，我把它改造成了单独的不依赖系列上下文的文章，具体原因可以看 坚持不下去了...

点击下方的阅读原文可以跳转到本系列的 GitHub 页，那里也有完整目录和规划，以及一些辅助的资料，欢迎提出各种问题。

低并发编程

战略上藐视技术，战术上重视技术

175篇原创内容

Official Account

收录于合集 #操作系统源码 43

上一篇

操作系统就用一张大表管理内存?

下一篇

读取硬盘前的准备工作有哪些?