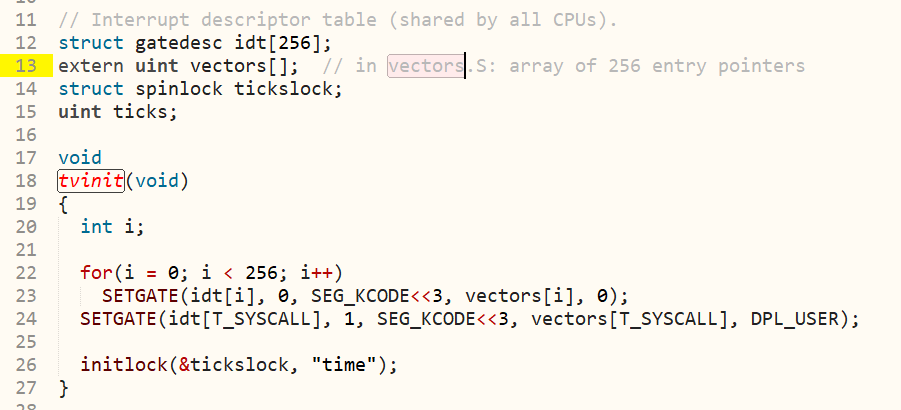
用户输入命令

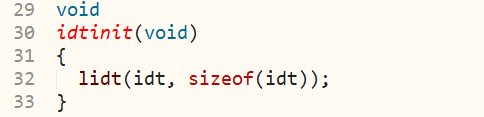
我们要在键盘上输入 cat test.txt\n 这些字符,在输入这些字符的过程之中,每输入一个字符,键盘通过usb数据线向cpu发出了中断请求,cpu会转去执行中断服务程序,我们的中断服务程序是由xv6系统定义的.

**trap.c**这个文件包含了xv6对不同的中断的处理过程以及中断表的初始化。

Tvinit函数主要完成对idt表进行初始化。其中vectors中存放的是每个中断处理程序的入口地址。vectors的定义是在vectors.S中， 由一个perl程序vectors.pl生成。另外对tickslock这个锁进行初始化，这个锁是用来处理时钟中断的。

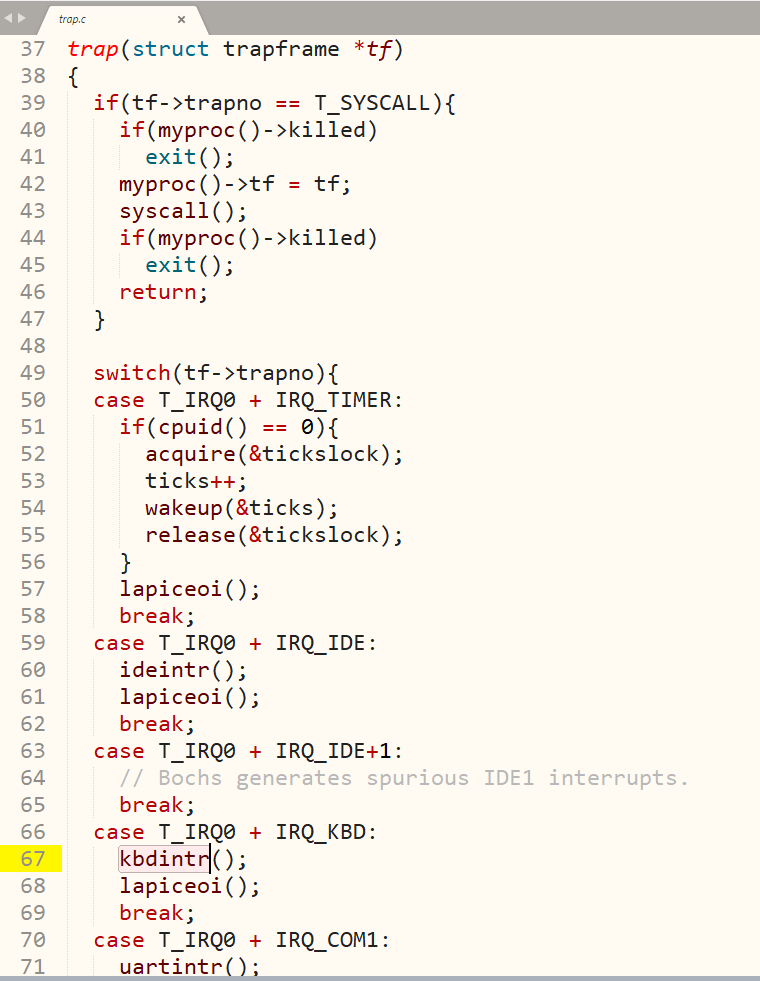


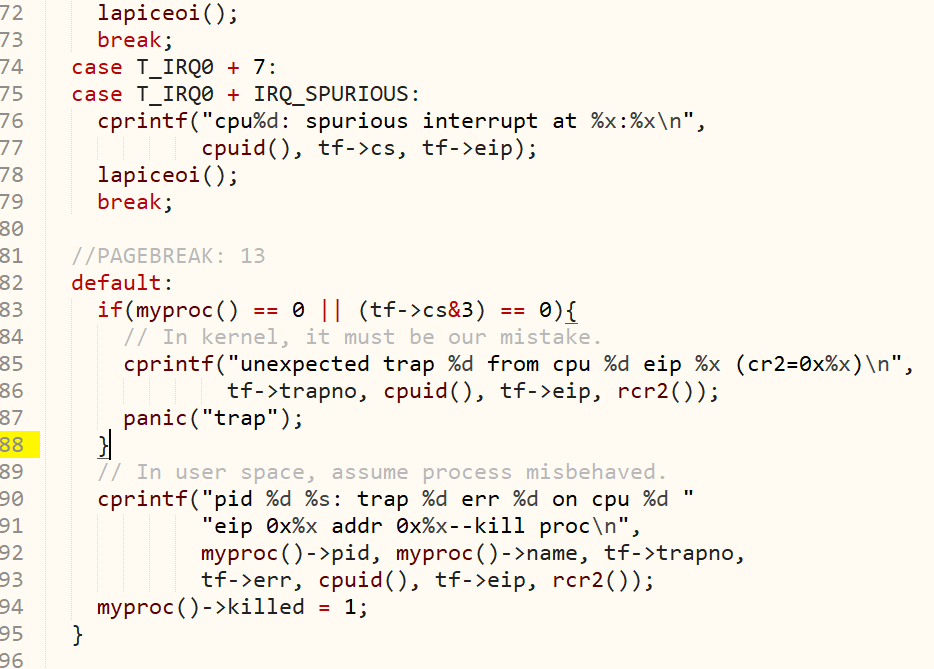
idtinit是用来加载idt表的。idtinit之所以和tvinit分开的原因是xv6支持多核心,所以idtinit会被多个CPU调用。但是tvinit过程只需要被调用一次。



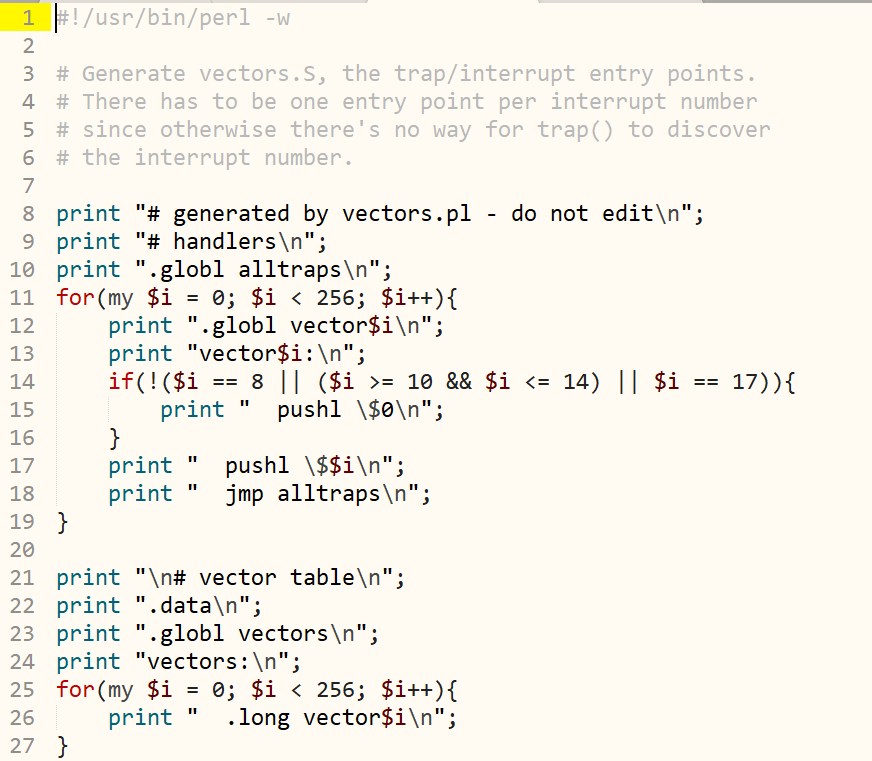
对应的**中断处理过程是在**trap函数之中.所有的中断在经过中断入口程序(trapasm.S中定义）后，都会最终跳转到这里。其处理过程如下：

1. 在39~47行，判断中断是系统调用，若是则通过syscall过程完成系统调用的处理。值得注意的是，这里进行了当前进程是否被其他进程kill的情况。也就是说，如果一个进程被其他进程Kill，那么此进程将在进入或退出内核态时被切断。
2. 49～95行是对不同的中断进行处理。
3. 第50行case IRQ\_OFFSET+IRQ\_TIMER是处理时钟中断，如果当前CPU是0号CPU，那么ticks将增加一，同时进行对处于sleeping状态的进程进行检查（通过wakeup)。在wakeup中，一旦发现某些进程休眠的时间结束，则把其转化成RUNNABLE；
4. 59～62行通过 ideintr()完成磁盘中断处理；
5. 66～69行通过kbdintr()完成键盘中断处理；

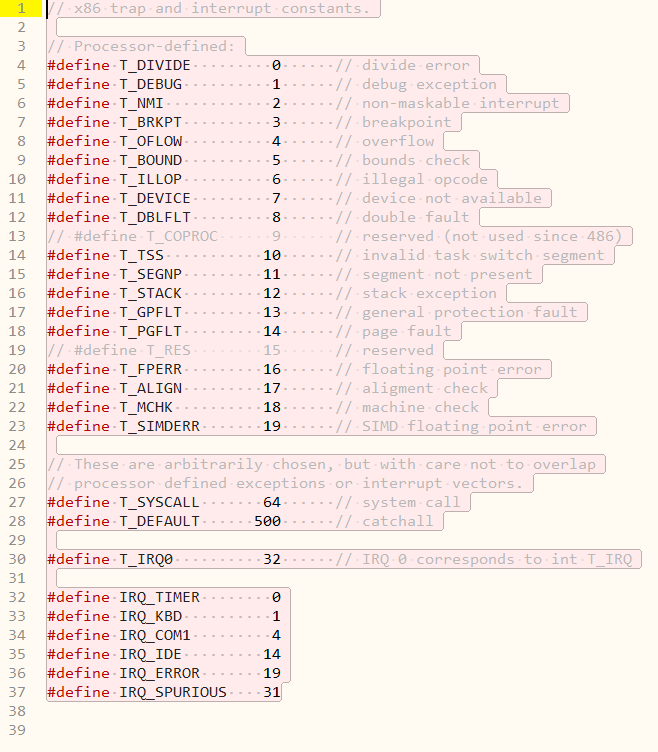




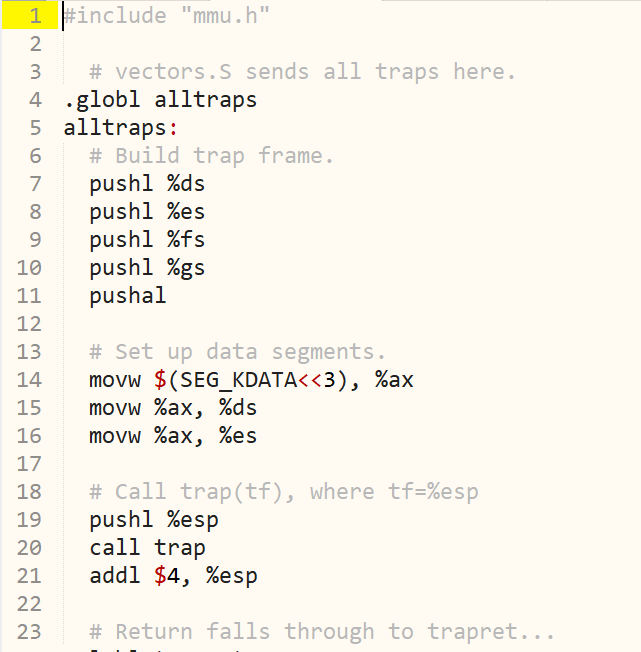
而中断向量的入口地址和入口程序是由**vectors.S来定义的.(** vectors.S是由vectors.pl生成的**)**可以注意到，中断分成两类：一类是**压入错误编码的(error code)，**另一类**不会压入错误编码**。因此对于第二类，vectors.S将压入一个0。此外vectors.S还会将中断号压入栈。在压完两个必要的值之后，所有中断都将统一的跳转进入alltraps入口程序。



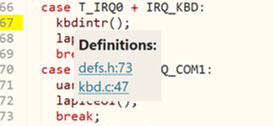
Cpu在检测到键盘的中断请求之后,如果允许中断的话,就转去执行中断服务程序,而中断向量号在traps.h之中就已经定义好了.



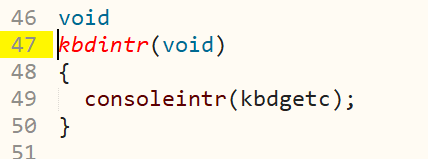
由于xv6在启动过程之中已经完成了中断表的初始化,现在由于键盘事件产生了中断请求, 在**vectors.S之中,将在栈内压入一个0,键盘事件的中断号然后就会跳转到**alltraps.完成一些保存寄存器的操作,然后在14~16行是将段寄存器换到内核段。然后调用trap.c中的trap函数。因为trap的参数是指向trapframe的指针，所以将esp寄存器压入栈，其正好是trapframe的地址。

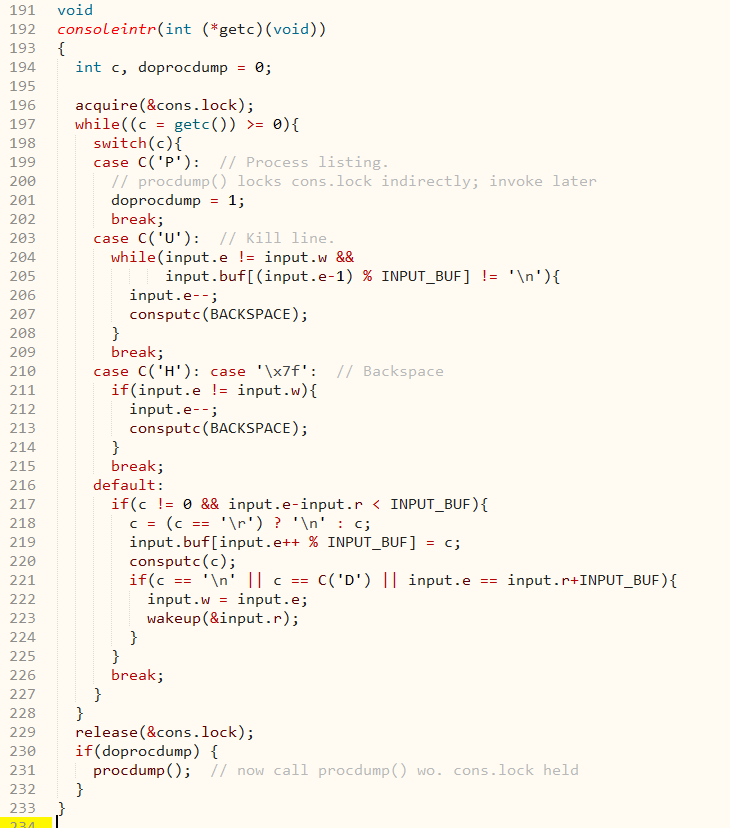


由于之前栈你已经有了中断向量号,trap就会去执行中断服务程序kbdintr,而这个函数是定义在kdb.c函数之中,

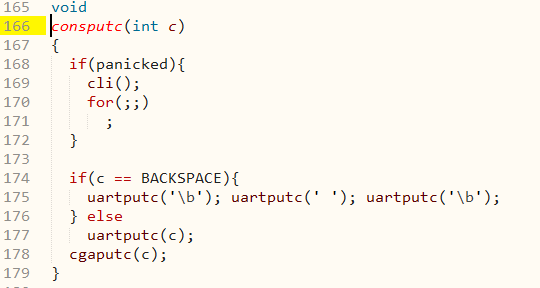


转到对应的位置,发现kbdintr的定义如下图所示, kbdintr调用了consoleintr函数

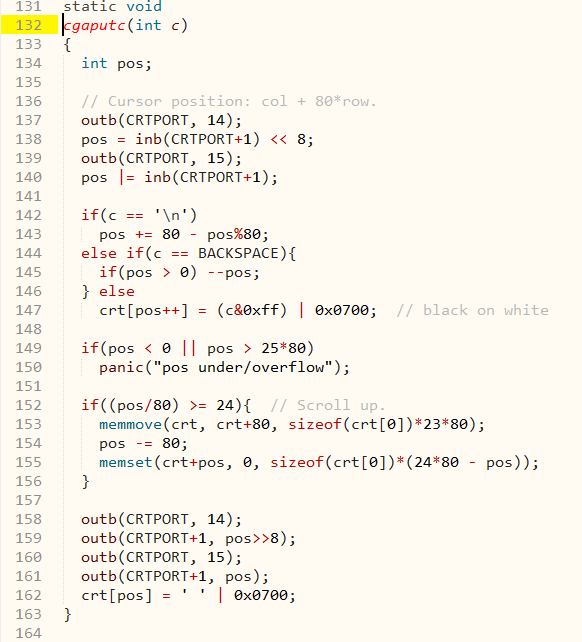




通过分析consoleintr 函数我们可以得知, consoleintr 函数会根据参数getc获取用户的输入,根据用户的输入做出不同的响应, consoleintr在处理很多字符的时候会调用 consputc 函数



然后consputc函数会调用 cgaputs 函数来将数据输出到显示器之上.



在执行完中断处理函数之后,返回到我们的os中,继续执行我们的代码.