**CPU alarm**

一、实验目的

向xv6添加一个功能，当进程使用CPU时间时，该功能会定期发出警报。第一步是添加alarm系统调用，第二步是设置陷入后内核的处理，即修改trap.c/trap()函数

二、实验内容

[In-class: xv6 CPU alarm (mit.edu)](https://pdos.csail.mit.edu/6.828/2018/homework/xv6-alarm.html)

三、实验步骤与实验结论

(1) 新建一个文件 alarmtest.c

(2) 根据实验提示向每个文件中加入代码

1）修改 Makefile 以使 alarmtest.c 编译为 xv6 用户程序。

2）修改user.h

int alarm(int ticks, void (\*handler)());

3）更新syscall.h和usys.s直至允许警报测试调用警报系统调用。

sys\_alarm存储警报间隔和指向proc结构中新字段的处理函数的指针

int

sys\_alarm(void)

{

int ticks;

void (\*handler)();

if(argint(0, &ticks) < 0)

return -1;

if(argptr(1, (char\*\*)&handler, 1) < 0)

return -1;

myproc()->alarmticks = ticks;

myproc()->alarmhandler = handler;

return 0;

}

修改至syscall.c 并将SYS\_ALARM条目添加到该文件中的 syscalls 数组中

4）在struct proc中创建新字段，在proc.c的allocproc()中初始化proc字段。

5）alarmticks初始化为-1，可以在没有系统调用alarm的时候，永远不触发alarm

6）如果只看到一个“alarm！”，将alarmtest.c中的迭代次数增加10倍。

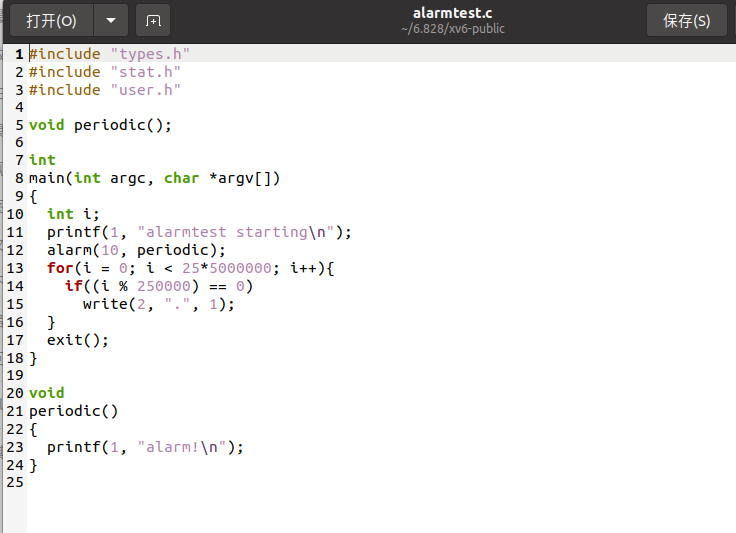
(3) 这个 HW 使用到了时钟中断。根据提示，我们需要在 trap （）中的 case T\_IRQ0 + IRQ\_TIMER:写入我们的处理逻辑

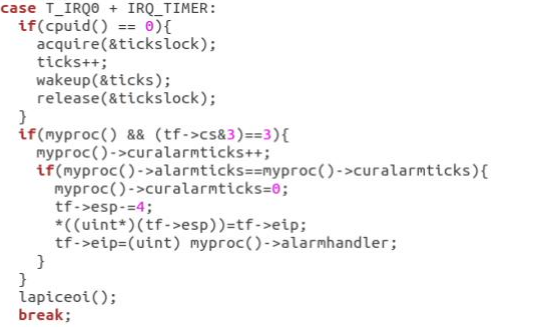
时钟中断

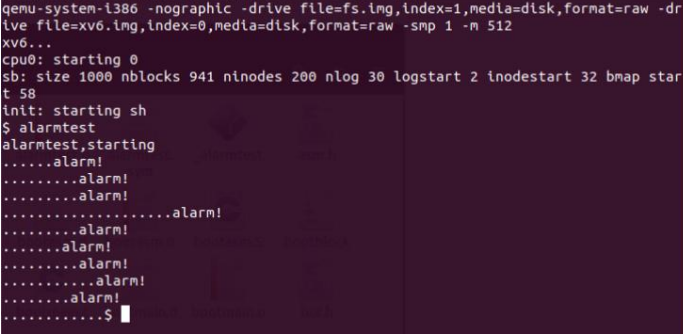
每当经过一个tick时钟中断会触发一次，更新相应的信息，而具体一个tick是多久因计算机而异，如对于普通计算机，要求一秒钟起码要有60个tick来维持60hz的刷新率。每次触发中断都要进行上下文保存、切换以及tlb刷新，因此一个tick太短会影响总体的吞吐量，而一个tick太长会降低系统与用户的交互性

alarm

因此我们需要做的就是在每个时钟中断到来之时，检查当前进程（因为现在我们只有这一个进程）的handler是否为空，如果不为空，检查一下是否经过了足够长的interval(alarmtick)，满足的话进行必要的检查以及状态更新，接着修改原先的trapframe和栈来使得内核态回到用户态时不再是回到原来触发时钟中断的地方，而是跳转到handler函数，而handler函数执行完后自动return（通过合理的栈内容安排）到当前进程被中断的地方。







四、反思总结

为什么会陷入到case T\_IRQ0 + IRQ\_TIMER？

是因为系统在固定的时间间隔都发生一次时钟中断，每当计数器溢出的时候，就会产生中断，陷入到case T\_IRQ0 + IRQ\_TIMER去处理，处理完后在计数器重置，返回原进程。

注意：在Makefile中加入UPROGS时，要写\_alarmtest而不是\_alarm