xv6 CPU alarm

- 姓名：陈姝宇
- 学号：3017218119
- 班级软工3班

实验内容

将向xv6添加一个功能，该功能在进程使用cpu时间时定期向进程发出警报。这对于那些希望限制cpu占用时间的计算限制进程，或者对于那些希望计算但同时也希望采取一些周期性操作的进程，都是很有用的。一般来说，您将实现用户级中断/故障处理程序的基本形式；例如，您可以使用类似的方法来处理应用程序中的页面故障。

您应该添加一个新的报警（间隔，处理程序）系统调用。如果应用程序调用alarm（n，fn），那么在程序消耗的每n个cpu时间“滴答”之后，内核将导致调用应用程序函数fn。当fn返回时，应用程序将恢复到它关闭的位置。勾号是XV6中相当任意的时间单位，由硬件计时器生成中断的频率决定。

实验步骤

1. 根据hint在`Makefile`中加入alarmtest使其成为xv6的用户程序

```
185         _alarmtest\
```

2. 在`user.h`中加入alarm的system call

```
 27 int alarm(int ticks, void (*handler)());
```

3. 在`syscall.h`中添加系统调用的数字常量

```
 24 #define SYS_alarm  23
```

4. 在`syscall.c`中添加外部函数，在static int (*syscalls[])(void)中添加alarm

```
109 extern int sys_alarm(void);
110
134 [SYS_alarm]   sys_alarm,
```

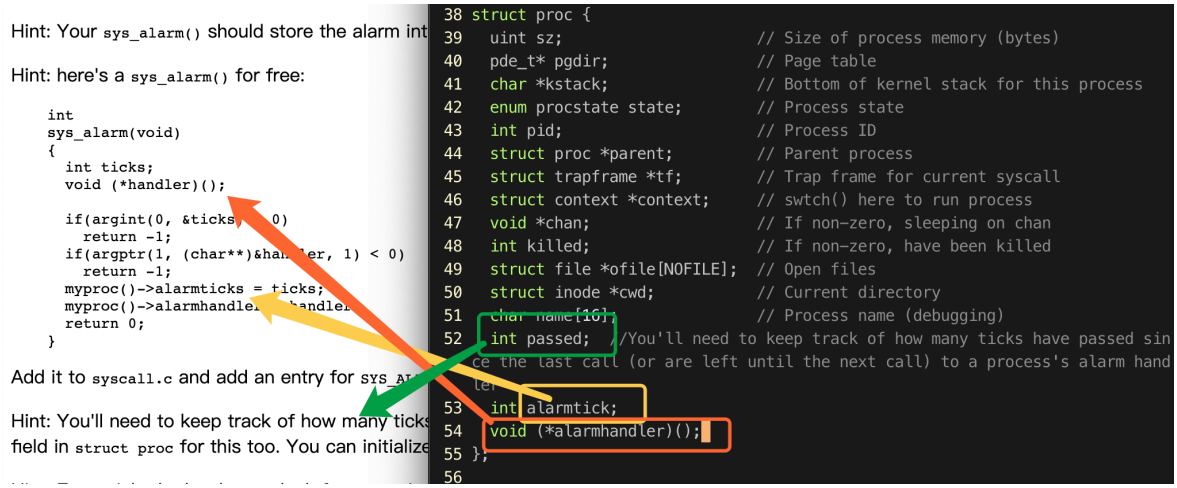
5. 在`usys.`S中添加syscall(alarm)，注意大小写

```
 33 SYSCALL(alarm)
```

6. 在 `sysproc.c` 文件中添加给的 sys_alarm 方法

```
102 int
103 sys_alarm(void)
104 {
105   int ticks;
106   void (*handler)();
107
108   if(argint(0, &ticks) < 0)
109     return -1;
110   if(argptr(1, (char**)&handler, 1) < 0)
111     return -1;
112   myproc()->alarmticks = ticks;
113   myproc()->alarmhandler = handler;
114   return 0;
115 }
```

7. 根据提示在proc.h文件的proc结构体中添加成员变量。此外在sys_alarm代码中用到了alarmticks成员变量和alarmhandler成员方法，在原始的pro中并没有这两个变量，也应当添加进去。



可以看到在proc.c中通过allocproc()方法初始化proc

```
1. vim proc.c (ssh)
126   p = allocproc();
```

8. 最后要求上交trap.c代码，所以阅读 trap.c 代码，找到提示中说的T_IRQ0 + IRQ_TIMER

```
49   switch(tf->trapno){
50   case T_IRQ0 + IRQ_TIMER:
51     if(cpuid() == 0){
52       acquire(&tickslock);
53       ticks++;
54       wakeup(&ticks);
55       release(&tickslock);
56     }
57     lapiceoi();
58     break;
```

先用grep找到tf在proc.h文件中，发现struct trapframe结构体为结构体proc中的成员变量。

```
proc.h:45:  struct trapframe *tf;        // Trap frame for current syscall
 37 // Per-process state
 38 struct proc {
 39   uint sz;                     // Size of process memory (bytes)
 40   pde_t* pgdir;                // Page table
 41   char *kstack;                // Bottom of kernel stack for this process
 42   enum procstate state;        // Process state
 43   int pid;                     // Process ID
 44   struct proc *parent;         // Parent process
 45   struct trapframe *tf;        // Trap frame for current syscall
 46   struct context *context;     // swtch() here to run process
 47   void *chan;                  // If non-zero, sleeping on chan
 48   int killed;                  // If non-zero, have been killed
 49   struct file *ofile[NOFILE];  // Open files
 50   struct inode *cwd;           // Current directory
 51   char name[16];               // Process name (debugging)
 52 };
```

默认处理中有个hint类似的if判断句，提示中第一个是不等于——if(myproc() != 0 && (tf->cs & 3) == 3)，默认中是等于0。给出的if条件句中，myproc() != 0是表示进程正在运行，(tf->cs & 3) == 3中断来自用户空间。

```
 81   //PAGEBREAK: 13
 82   default:
 83     if(myproc() == 0 || (tf->cs&3) == 0){
 84       // In kernel, it must be our mistake.
 85       cprintf("unexpected trap %d from cpu %d eip %x (cr2=0x%x)\n",
 86               tf->trapno, cpuid(), tf->eip, rcr2());
 87       panic("trap");
 88     }
 89     // In user space, assume process misbehaved.
 90     cprintf("pid %d %s: trap %d err %d on cpu %d "
 91             "eip 0x%x addr 0x%x--kill proc\n",
 92             myproc()->pid, myproc()->name, tf->trapno,
 93             tf->err, cpuid(), tf->eip, rcr2());
 94     myproc()->killed = 1;
 95   }
```

9. 在case T_IRQ0 + IRQ_TIMER 中添加如下代码，先将passed加1，如果alarmticks =passalarmticks，就触发handler的调用，先将pass清零，调用之前先将当前线程的栈顶指针esp减4，并保存CPU的eip寄存器中存放的下一个CPU指令存放的内存地址，便于返回当前函数，最后让eip指向handler，就可以调用handler函数。

```
 49   switch(tf->trapno){
 50   case T_IRQ0 + IRQ_TIMER:
 51     if(myproc() != 0 && (tf->cs & 3) == 3){
 52       myproc()->passed++;
 53       if(myproc()->alarmticks == myproc()->passed){
 54         myproc()->passed = 0;
 55         tf->esp -= 4;
 56         *((uint *)(tf->esp)) = tf->eip;
 57         tf->eip =(uint) myproc()->alarmhandler;
 58       }
 59     }
                                                    53,49         20%
```

10. 按照提示增加alarmtest.c

```c
#include "types.h"
#include "stat.h"
#include "user.h"

void periodic();

int
main(int argc, char *argv[])
{
  int i;
  printf(1, "alarmtest starting\n");
  alarm(10, periodic);
  for(i = 0; i < 25*500000; i++){
    if((i % 250000) == 0)
      write(2, ".", 1);
  }
  exit();
}

void
periodic()
{
  printf(1, "alarm!\n");
}
~
```

实验结果

运行发现没有alarm。

```
init: starting sh
$ alarmtest
alarmtest starting
.................................................$
```

增加循环次数，发现有输出，但是.号太多

```c
  for(i = 0; i < 500*500000; i++){
    if((i % 250000) == 0)
```

```
init: starting sh
$ alarmtest
alarmtest starting
........................................................................................
........................................................................................
alarm!
........................................................................................
.......................................alarm!
........................................................................................
.................................................alarm!
........................................................................................
..................................................................alarm!
........................................................................................
.......................................................alarm!
........................................................................................
...................................................................alarm!
........................................................................................
........................................................................alarm!
..................................................$ _
```

将. 的输出频率调小，重新观察。

```
12    alarm(10, periodic);
13    for(i = 0; i < 500*500000; i++){
14      if((i % 2500000) == 0)
15        write(2, ".", 1);
```

```
$ alarmtest
alarmtest starting
.............alarm!
.............alarm!
.............alarm!
.............alarm!
.............alarm!
.............alarm!
.............alarm!
......$
```