Assignment 2/describe the execution of ioctl()

Yuqi Zhou, Illinois Institute of Technology 10/05/2018

xv6/vectors.S

Push trap number onto the stack, and call alltraps to do most of the context saving.

xv6/trapasm.S

The result of this effort is that the kernel stack now contains a struct trapframe containing the processor registers at the time of the trap. The trap frame contains all the information necessary to restore the user mode processor registers when the kernel returns to the current processor.

xv6/trap.c

Trap handler handles all type of interrupts and traps, so first checks the trap number. It is T_SYSCALL, so handles the system call.

```
1  void
2  trap(struct trapframe *tf)
3  {
4    if(tf->trapno == T_SYSCALL){
5      if(myproc()->killed)
6        exit();
7      int num;
8      int i;
9      num = tf->eax;
10     i = num - 1;
11     myproc()->tf = tf;
12     myproc()->countSyscall[i] = myproc()->countSyscall[i] + 1;
13     syscall();
14     if(myproc()->killed)
15       exit();
16     return;
17   }
```

xv6/syscall.c

System call number has been pass in %eax and we use myproc()→tf→eax to call the appropriate routine and in our case, sys_close() will be call.

```
1  static int (*syscalls[])(void) = {
2  ...
3  [SYS_close]   sys_close,
4  ...
5  };
6  void
7  syscall(void)
8  {
```

```
 9   int num;
10   //int i;
11   struct proc *curproc = myproc();
12
13   num = curproc->tf->eax;
14   //i = num - 1;
15   if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
16     curproc->tf->eax = syscalls[num]();
17     //myproc()->countSyscall[i] = myproc()->countSyscall[i] + 1;
18   } else {
19     cprintf("%d %s: unknown sys call %d\n",
20             curproc->pid, curproc->name, num);
21     curproc->tf->eax = -1;
22   }
23 }
```

xv6/sysfile.c

argfd uses argint to retrieve a file descriptor number, checks if it is valid file descriptor, and returns the corresponding struct file. If it is valid file descriptor, call fileclose(f)(f is the corresponding struct file). In our case, fd>=NOFILE. argfd() returns -1 and sys_close() returns -1. System calls conventionally use negative numbers to indicate errors. The file descriptor is invalid, syscall prints an error and returns -1.

```
1  int
2  sys_close(void)
3  {
4    int fd;
5    struct file *f;
6
7    if(argfd(0, &fd, &f) < 0)
8      return -1;
9    myproc()->ofile[fd] = 0;
10   fileclose(f);
11   return 0;
12 }
```

xv6/trapasm.S

Pop what was pushed before to restore the context of the running process and issue iret(lows privilege level to user mode and resume the user processor).

```
1   # Return falls through to trapret...
2  .globl trapret
3  trapret:
4    popal
5    popl %gs
6    popl %fs
7    popl %es
8    popl %ds
9    addl $0x8, %esp  # trapno and errcode
10   iret
```