# xv6 system calls - part1


- 姓名：陈姝宇
- 学号：3017218119
- 班级：软工三班


### xv6 system calls – part1

## 1.实验内容

1）修改xv6内核，为每次系统调用打印一行代码。打印系统调用的名称和返回值；不需要打印系统调用参数；

2）打印系统调用的参数；

## 2. 实验步骤

### 2.1 增加系统调用打印

阅读syscall.c 代码。存放所有指令的的数组，猜测`SYS_fork SYS_exit` 等SYS__xxx为从0开始的连续int，在syscall.c里并没有初始化SYS__fork等常量，极有可能在syscall.h文件里。

```
107 static int (*syscalls[])(void) = {
108 [SYS_fork]    sys_fork,
109 [SYS_exit]    sys_exit,
110 [SYS_wait]    sys_wait,
111 [SYS_pipe]    sys_pipe,
112 [SYS_read]    sys_read,
113 [SYS_kill]    sys_kill,
114 [SYS_exec]    sys_exec,
115 [SYS_fstat]   sys_fstat,
116 [SYS_chdir]   sys_chdir,
117 [SYS_dup]     sys_dup,
118 [SYS_getpid]  sys_getpid,
119 [SYS_sbrk]    sys_sbrk,
120 [SYS_sleep]   sys_sleep,
121 [SYS_uptime]  sys_uptime,
122 [SYS_open]    sys_open,
123 [SYS_write]   sys_write,
124 [SYS_mknod]   sys_mknod,
125 [SYS_unlink]  sys_unlink,
126 [SYS_link]    sys_link,
127 [SYS_mkdir]   sys_mkdir,
128 [SYS_close]   sys_close,
129 };
```

打开syscall.h。在这个文件里存放了所有system call的数

1. vim syscall.h (ssh)

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
~
~
```

系统调用的核心代码，当系统调用不存在的时候为else，会打印unknown。根据变量名，猜测curproc->name为名字，num为数字。

```
void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();

  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

仿照else写法，完成if条件语句里的打印

```
131 void
132 syscall(void)
133 {
134   int num;
135   struct proc *curproc = myproc();
136
137   num = curproc->tf->eax;
138   if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
139     curproc->tf->eax = syscalls[num]();
140     cprintf("%s -> %d\n", curproc->name, num);
141   } else {
142     cprintf("%d %s: unknown sys call %d\n"
143             curproc->pid, curproc->name, num);
144     curproc->tf->eax = -1;
145   }
146 }
-- INSERT --                                          140,45        Bot
```

再次运行make qemu，发现curproc->name不是系统调用的名字。

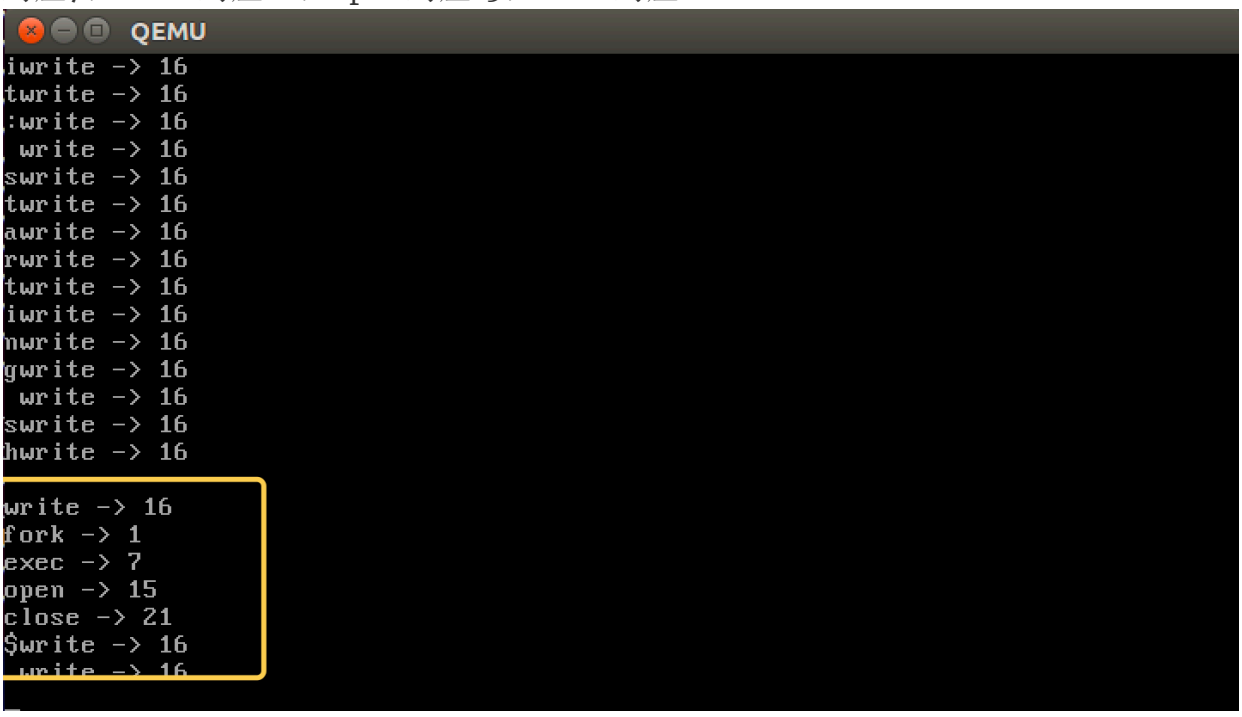
```
iinit -> 16
ninit -> 16
ginit -> 16
 init -> 16
sinit -> 16
hinit -> 16

init -> 16
init -> 1
sh -> 7
sh -> 15
sh -> 21
$sh -> 16
 sh -> 16
```

由于num是系统调用指令的编号，所以可以维护一个数组syscall_name，根据syscall.c里系统调用的编号赋值数组，采用num下标访问系统调用的名字。

```
131 char * syscalls_name[]={
132   [SYS_fork]    "fork",
133   [SYS_exit]    "exit",
134   [SYS_wait]    "wait",
135   [SYS_pipe]    "pipe",
136   [SYS_read]    "read",
137   [SYS_kill]    "kill",
138   [SYS_exec]    "exec",
139   [SYS_fstat]   "fstat",
140   [SYS_chdir]   "chdir",
141   [SYS_dup]     "dup",
142   [SYS_getpid]  "getpid",
143   [SYS_sbrk]    "sbrk",
144   [SYS_sleep]   "sleep",
145   [SYS_uptime]  "uptime",
146   [SYS_open]    "open",
147   [SYS_write]   "write",
148   [SYS_mknod]   "mknod",
149   [SYS_unlink]  "unlink",
150   [SYS_link]    "link",
151   [SYS_mkdir]   "mkdir",
152   [SYS_close]   "close",
153 };
```

```
164     cprintf("%s -> %d\n",syscalls_name[num],num);
```

再次执行make qemu，出现下图，根据syscall.h对比指令和编号，fork 对应2，exec对应7，write对应16，open对应15，close对应21。



### 2.2 打印系统调用的参数

观察syscall.c内的函数

- fetchint(uint addr, int *ip) // Fetch the int at addr from the current process.
- fetchstr(uint addr, char **pp)

这个两个可以从函数的注释中知道，作用为Fetch the int/string at addr from the current process.

- argint(int n, int *ip)
- argptr(int n, char **pp, int size)
- argstr(int n, char **pp)

以上三个的注释总结为：Fetch the nth system call argument。 并且argint里中调用了fetchint，argstring调用了fetchstr。

在fetchint中增加对于参数ip的打印，就可以打印int参数，同理在fetchstr中增加对pp参数的打印。

```
17 int
18 fetchint(uint addr, int *ip)
19 {
20   struct proc *curproc = myproc();
21
22   if(addr >= curproc->sz || addr+4 > curproc->sz)
23     return -1;
24   *ip = *(int*)(addr);
25   cprintf("int arg: %d\n",*ip);
26   return 0;
27 }
```

```
33 fetchstr(uint addr, char **pp)
34 {
35   char *s, *ep;
36   struct proc *curproc = myproc();
37
38   if(addr >= curproc->sz)
39     return -1;
40   *pp = (char*)addr;
41   cprintf("str arg: %s\n",*pp);
42   ep = (char*)curproc->sz;
43   for(s = *pp; s < ep; s++){
44     if(*s == 0)
45       return s - *pp;
46   }
47   return -1;
48 }
```

## 3.实验结果

在xv6中执行make qemu，如下图所示：