XV6 SYSTEM CALLS 作业报告

软工2班 3017218092 侯雨茜

PART 1: SYSTEM CALL TRACING

第一部分，需要跟踪系统调用，并把他们打印在终端上。

首先，观察 `syscall.h` 文件：

```
// 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
```

文件中定义了系统调用的名称和对应的序号，这些需要在终端上被显示出来。

接着，修改 `syscall.c` 文件。在文件中添加编号和系统调用名称相互对应的数组，然后在 `syscall()` 函数中添加对应的输出命令：

```c
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
#include "syscall.h"

// User code makes a system call with INT T_SYSCALL.
// System call number in %eax.
// Arguments on the stack, from the user call to the C
// library system call function. The saved user %esp points
// to a saved program counter, and then the first argument.

// Fetch the int at addr from the current process.
int
fetchint(uint addr, int *ip)
{
  struct proc *curproc = myproc();

  if(addr >= curproc->sz || addr+4 > curproc->sz)
    return -1;
  *ip = *(int*)(addr);
  return 0;
}

// Fetch the nul-terminated string at addr from the current process.
// Doesn't actually copy the string - just sets *pp to point at it.
// Returns length of string, not including nul.
int
fetchstr(uint addr, char **pp)
{
  char *s, *ep;
  struct proc *curproc = myproc();

  if(addr >= curproc->sz)
    return -1;
  *pp = (char*)addr;
  ep = (char*)curproc->sz;
  for(s = *pp; s < ep; s++){
    if(*s == 0)
      return s - *pp;
  }
  return -1;
}

// Fetch the nth 32-bit system call argument.
int
```

```
50  argint(int n, int *ip)
51  {
52    return fetchint((myproc()->tf->esp) + 4 + 4*n, ip);
53  }
54
55  // Fetch the nth word-sized system call argument as a pointer
56  // to a block of memory of size bytes.  Check that the pointer
57  // lies within the process address space.
58  int
59  argptr(int n, char **pp, int size)
60  {
61    int i;
62    struct proc *curproc = myproc();
63
64    if(argint(n, &i) < 0)
65      return -1;
66    if(size < 0 || (uint)i >= curproc->sz || (uint)i+size > curproc-
    >sz)
67      return -1;
68    *pp = (char*)i;
69    return 0;
70  }
71
72  // Fetch the nth word-sized system call argument as a string pointer.
73  // Check that the pointer is valid and the string is nul-terminated.
74  // (There is no shared writable memory, so the string can't change
75  // between this check and being used by the kernel.)
76  int
77  argstr(int n, char **pp)
78  {
79    int addr;
80    if(argint(n, &addr) < 0)
81      return -1;
82    return fetchstr(addr, pp);
83  }
84
85  extern int sys_chdir(void);
86  extern int sys_close(void);
87  extern int sys_dup(void);
88  extern int sys_exec(void);
89  extern int sys_exit(void);
90  extern int sys_fork(void);
91  extern int sys_fstat(void);
92  extern int sys_getpid(void);
93  extern int sys_kill(void);
94  extern int sys_link(void);
95  extern int sys_mkdir(void);
96  extern int sys_mknod(void);
97  extern int sys_open(void);
```

```
98  extern int sys_pipe(void);
99  extern int sys_read(void);
100 extern int sys_sbrk(void);
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106
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 };
130
131 // 用数组存放系统调用对应名字
132 static char syscalls_name[][6] = {
133     [SYS_fork]    "fork",
134     [SYS_exit]    "exit",
135     [SYS_wait]    "wait",
136     [SYS_pipe]    "pipe",
137     [SYS_read]    "read",
138     [SYS_kill]    "kill",
139     [SYS_exec]    "exec",
140     [SYS_fstat]   "fstat",
141     [SYS_chdir]   "chdir",
142     [SYS_dup]     "dup",
143     [SYS_getpid]  "getpid",
144     [SYS_sbrk]    "sbrk",
145     [SYS_sleep]   "sleep",
146     [SYS_uptime]  "uptime",
```

```
    [SYS_open]    "open",
    [SYS_write]   "write",
    [SYS_mknod]   "mknod",
    [SYS_unlink]  "unlink",
    [SYS_link]    "link",
    [SYS_mkdir]   "mkdir",
    [SYS_close]   "close",
};

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

    // 获取系统调用号
  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    //curproc->tf->eax = syscalls[num]();
      // 存储系统调用返回值
      curproc->tf->eax = syscalls[num]();
      // 打印系统调用名称和编号并换行，使用制表符对齐格式
      cprintf("\tsyscalls: %s\tid: %d\n", syscalls_name[num], num);
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

最后，在ubuntu中编译运行：

```
make qemu-nox
```

运行结果如下：

```
记录了336+1 的读入
记录了336+1 的写出
172124 bytes (172 kB, 168 KiB) copied, 0.000625019 s, 275 MB/s
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw
 -drive file=xv6.img,index=0,media=disk,format=raw -smp 1 -m 512 -snapshot
xv6...
cpu0: starting 0
sb: size 20000 nblocks 19937 ninodes 200 nlog 30 logstart 2 inodestart 32 bm
ap start 58
main-loop: WARNING: I/O thread spun for 1000 iterations
        syscalls: exec  id: 7
        syscalls: open  id: 15
        syscalls: mknod id: 17
        syscalls: open  id: 15
        syscalls: dup   id: 10
        syscalls: dup   id: 10
i       syscalls: write id: 16
n       syscalls: write id: 16
i       syscalls: write id: 16
t       syscalls: write id: 16
:       syscalls: write id: 16
        syscalls: write id: 16
s       syscalls: write id: 16
t       syscalls: write id: 16
a       syscalls: write id: 16
r       syscalls: write id: 16
t       syscalls: write id: 16
i       syscalls: write id: 16
n       syscalls: write id: 16
g       syscalls: write id: 16
        syscalls: write id: 16
s       syscalls: write id: 16
h       syscalls: write id: 16

        syscalls: write id: 16
        syscalls: fork  id: 1
        syscalls: exec  id: 7
        syscalls: open  id: 15
        syscalls: close id: 21
$       syscalls: write id: 16
        syscalls: write id: 16
```

PART 2: DATE SYSTEM CALL

第二部分，需要添加一个系统调用函数，使其可以返回UTC时间。

`lapic.c` 中定义了函数 `cmostime()` 来读取当前时间。

```
1  // qemu seems to use 24-hour GWT and the values are BCD encoded
2  void
3  cmostime(struct rtcdate *r)
4  {
```

```
  struct rtcdate t1, t2;
  int sb, bcd;

  sb = cmos_read(CMOS_STATB);

  bcd = (sb & (1 << 2)) == 0;

  // make sure CMOS doesn't modify time while we read it
  for(;;) {
    fill_rtcdate(&t1);
    if(cmos_read(CMOS_STATA) & CMOS_UIP)
        continue;
    fill_rtcdate(&t2);
    if(memcmp(&t1, &t2, sizeof(t1)) == 0)
      break;
  }

  // convert
  if(bcd) {
#define    CONV(x)     (t1.x = ((t1.x >> 4) * 10) + (t1.x & 0xf))
    CONV(second);
    CONV(minute);
    CONV(hour  );
    CONV(day   );
    CONV(month );
    CONV(year  );
#undef     CONV
  }

  *r = t1;
  r->year += 2000;
}
```

`date.h` 中定义了 `struct rtcdate` 结构体，作为 `cmostime()` 的参数。

```
struct rtcdate {
  uint second;
  uint minute;
  uint hour;
  uint day;
  uint month;
  uint year;
};
```

添加系统调用函数 `date()`：

1. 在 `syscall.h` 中增加日期的系统调用编号

```
// 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
// 日期系统调用编号
#define SYS_date   22
```

2. 在 `syscall.c` 中添加系统调用函数的外部声明

```
// Fetch the nth word-sized system call argument as a string pointer.
// Check that the pointer is valid and the string is nul-terminated.
// (There is no shared writable memory, so the string can't change
// between this check and being used by the kernel.)
int
argstr(int n, char **pp)
{
  int addr;
  if(argint(n, &addr) < 0)
    return -1;
  return fetchstr(addr, pp);
}

extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
```

```
22  extern int sys_kill(void);
23  extern int sys_link(void);
24  extern int sys_mkdir(void);
25  extern int sys_mknod(void);
26  extern int sys_open(void);
27  extern int sys_pipe(void);
28  extern int sys_read(void);
29  extern int sys_sbrk(void);
30  extern int sys_sleep(void);
31  extern int sys_unlink(void);
32  extern int sys_wait(void);
33  extern int sys_write(void);
34  extern int sys_uptime(void);
35  extern int sys_date(void);
36
37  static int (*syscalls[])(void) = {
38  [SYS_fork]    sys_fork,
39  [SYS_exit]    sys_exit,
40  [SYS_wait]    sys_wait,
41  [SYS_pipe]    sys_pipe,
42  [SYS_read]    sys_read,
43  [SYS_kill]    sys_kill,
44  [SYS_exec]    sys_exec,
45  [SYS_fstat]   sys_fstat,
46  [SYS_chdir]   sys_chdir,
47  [SYS_dup]     sys_dup,
48  [SYS_getpid]  sys_getpid,
49  [SYS_sbrk]    sys_sbrk,
50  [SYS_sleep]   sys_sleep,
51  [SYS_uptime]  sys_uptime,
52  [SYS_open]    sys_open,
53  [SYS_write]   sys_write,
54  [SYS_mknod]   sys_mknod,
55  [SYS_unlink]  sys_unlink,
56  [SYS_link]    sys_link,
57  [SYS_mkdir]   sys_mkdir,
58  [SYS_close]   sys_close,
59  [SYS_date]    SYS_date,
60  };
61
62  // 用数组存放系统调用对应名字
63  static char syscalls_name[][6] = {
64      [SYS_fork]    "fork",
65      [SYS_exit]    "exit",
66      [SYS_wait]    "wait",
67      [SYS_pipe]    "pipe",
68      [SYS_read]    "read",
69      [SYS_kill]    "kill",
70      [SYS_exec]    "exec",
```

```
    [SYS_fstat]   "fstat",
    [SYS_chdir]   "chdir",
    [SYS_dup]     "dup",
    [SYS_getpid]  "getpid",
    [SYS_sbrk]    "sbrk",
    [SYS_sleep]   "sleep",
    [SYS_uptime]  "uptime",
    [SYS_open]    "open",
    [SYS_write]   "write",
    [SYS_mknod]   "mknod",
    [SYS_unlink]  "unlink",
    [SYS_link]    "link",
    [SYS_mkdir]   "mkdir",
    [SYS_close]   "close",
    [SYS_date]    "date",
};
```

3. 在 `user.h` 中添加用户态函数的定义

```
// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int date(struct rtcdate*);  // 日期
```

4. 在 `usys.S` 中添加用户态函数的实现

```
SYSCALL(fork)
SYSCALL(exit)
```

```
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(date)
```

5. 在 `sysproc.c` 中添加系统调用函数 `sys_date` 的实现

```
// return UTC time
int
sys_date(struct rtcdate *r)
{
    if (argptr(0, (void *)&r, sizeof(*r)) < 0) {
        return -1;
    }
    cmostime(r);    // get time from cmos
    return 0;
}
```

6. 新建 `date.c` 文件，添加使用系统调用函数 `sys_date` 的方法

```
#include "types.h"
#include "user.h"
#include "date.h"

int
main(int argc, char *argv[])
{
  struct rtcdate r;

  if (date(&r)) {
    printf(2, "date failed\n");
```

```
    exit();
  }

  // your code to print the time in any format you like...
    printf(1, "%d-%d %d %d:%d:%d\n", r.month, r.day, r,year, r.hour,
r.minute, r.second);
  exit();
}
```

7. 在 `Makefile` 中添加 `UPRODS` 对应命令的定义

```
UPROGS=\
  _cat\
  _echo\
  _forktest\
  _grep\
  _init\
  _kill\
  _ln\
  _ls\
  _mkdir\
  _rm\
  _sh\
  _stressfs\
  _usertests\
  _wc\
  _zombie\
  _big\
  _date\

fs.img: mkfs README $(UPROGS)
  ./mkfs fs.img README $(UPROGS)
```

编译并运行，输入命令 `date`，日期无法以正确格式显示。推测是由于Part 1中的输出影响了时间的显示。为防止Part 1的输出影响时间的显示，将 `syscall.c` 中的一些代码注释掉：

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
#include "syscall.h"

// User code makes a system call with INT T_SYSCALL.
// System call number in %eax.
```

```
12  // Arguments on the stack, from the user call to the C
13  // library system call function. The saved user %esp points
14  // to a saved program counter, and then the first argument.
15
16  // Fetch the int at addr from the current process.
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    return 0;
26  }
27
28  // Fetch the nul-terminated string at addr from the current process.
29  // Doesn't actually copy the string - just sets *pp to point at it.
30  // Returns length of string, not including nul.
31  int
32  fetchstr(uint addr, char **pp)
33  {
34    char *s, *ep;
35    struct proc *curproc = myproc();
36
37    if(addr >= curproc->sz)
38      return -1;
39    *pp = (char*)addr;
40    ep = (char*)curproc->sz;
41    for(s = *pp; s < ep; s++){
42      if(*s == 0)
43        return s - *pp;
44    }
45    return -1;
46  }
47
48  // Fetch the nth 32-bit system call argument.
49  int
50  argint(int n, int *ip)
51  {
52    return fetchint((myproc()->tf->esp) + 4 + 4*n, ip);
53  }
54
55  // Fetch the nth word-sized system call argument as a pointer
56  // to a block of memory of size bytes.  Check that the pointer
57  // lies within the process address space.
58  int
59  argptr(int n, char **pp, int size)
60  {
```

```
61    int i;
62    struct proc *curproc = myproc();
63
64    if(argint(n, &i) < 0)
65      return -1;
66    if(size < 0 || (uint)i >= curproc->sz || (uint)i+size > curproc-
>sz)
67      return -1;
68    *pp = (char*)i;
69    return 0;
70  }
71
72  // Fetch the nth word-sized system call argument as a string pointer.
73  // Check that the pointer is valid and the string is nul-terminated.
74  // (There is no shared writable memory, so the string can't change
75  // between this check and being used by the kernel.)
76  int
77  argstr(int n, char **pp)
78  {
79    int addr;
80    if(argint(n, &addr) < 0)
81      return -1;
82    return fetchstr(addr, pp);
83  }
84
85  extern int sys_chdir(void);
86  extern int sys_close(void);
87  extern int sys_dup(void);
88  extern int sys_exec(void);
89  extern int sys_exit(void);
90  extern int sys_fork(void);
91  extern int sys_fstat(void);
92  extern int sys_getpid(void);
93  extern int sys_kill(void);
94  extern int sys_link(void);
95  extern int sys_mkdir(void);
96  extern int sys_mknod(void);
97  extern int sys_open(void);
98  extern int sys_pipe(void);
99  extern int sys_read(void);
100 extern int sys_sbrk(void);
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_date(void);
107
108 static int (*syscalls[])(void) = {
```

```
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_date]    sys_date,
};

// 用数组存放系统调用对应名字
//static char syscalls_name[][6] = {
//    [SYS_fork]    "fork",
//    [SYS_exit]    "exit",
//    [SYS_wait]    "wait",
//    [SYS_pipe]    "pipe",
//    [SYS_read]    "read",
//    [SYS_kill]    "kill",
//    [SYS_exec]    "exec",
//    [SYS_fstat]   "fstat",
//    [SYS_chdir]   "chdir",
//    [SYS_dup]     "dup",
//    [SYS_getpid]  "getpid",
//    [SYS_sbrk]    "sbrk",
//    [SYS_sleep]   "sleep",
//    [SYS_uptime]  "uptime",
//    [SYS_open]    "open",
//    [SYS_write]   "write",
//    [SYS_mknod]   "mknod",
//    [SYS_unlink]  "unlink",
//    [SYS_link]    "link",
//    [SYS_mkdir]   "mkdir",
//    [SYS_close]   "close",
//    [SYS_date]    "date",
//};
```

```
158
159 void
160 syscall(void)
161 {
162   int num;
163   struct proc *curproc = myproc();
164
165     // 获取系统调用号
166   num = curproc->tf->eax;
167   if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
168     //curproc->tf->eax = syscalls[num]();
169       // 存储系统调用返回值
170       curproc->tf->eax = syscalls[num]();
171       // 打印系统调用名称和编号并换行，使用制表符对齐格式
172       //cprintf("\tsyscalls: %s\tid: %d\n", syscalls_name[num], num);
173   } else {
174     cprintf("%d %s: unknown sys call %d\n",
175             curproc->pid, curproc->name, num);
176     curproc->tf->eax = -1;
177   }
178 }
```

编译并运行，输入命令 `date`，日期以正确的格式显示：

```
cpu0: starting 0
sb: size 20000 nblocks 19937 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap s
tart 58
init: starting sh
$ date
10-22 2019 2:0:49
$
```