xv6 system calls-part2

- 姓名：陈姝宇
- 学号：3017218119
- 班级：软工三班

xv6 system calls-part2

1. 实验内容
2. 实验步骤
3. 实验结果
4. 实验结论

1. 实验内容

向xv6添加一个新的系统调用。新的系统调用将获得当前的UTC时间并将其返回给用户程序。

2. 实验步骤

在syscall.h中增加SYS_date

```
20 #define SYS_link   19
21 #define SYS_mkdir  20
22 #define SYS_close  21
23 #define SYS_date   22
```

模仿其他sys call 指令，在syscall.c中增加相应的extern int，在syscalls[]和syscalls_name 中增加date，

```
105 extern int sys_wait(void);
106 extern int sys_write(void);
107 extern int sys_uptime(void);
108 extern int sys_date(void);
110 static int (*syscalls[])(void) = {
112 [SYS_exit]    sys_exit,
113 [SYS_wait]    sys_wait,
114 [SYS_pipe]    sys_pipe,
115 [SYS_read]    sys_read,
116 [SYS_kill]    sys_kill,
117 [SYS_exec]    sys_exec,
118 [SYS_fstat]   sys_fstat,
119 [SYS_chdir]   sys_chdir,
120 [SYS_dup]     sys_dup,
121 [SYS_getpid]  sys_getpid,
122 [SYS_sbrk]    sys_sbrk,
123 [SYS_sleep]   sys_sleep,
124 [SYS_uptime]  sys_uptime,
125 [SYS_open]    sys_open,
126 [SYS_write]   sys_write,
127 [SYS_mknod]   sys_mknod,
128 [SYS_unlink]  sys_unlink,
129 [SYS_link]    sys_link,
130 [SYS_mkdir]   sys_mkdir,
132 [SYS_date]    sys_date,
133 };
```

```
135 char * syscalls_name[]={
136   [SYS_fork]    "fork",
137   [SYS_exit]    "exit",
138   [SYS_wait]    "wait",
139   [SYS_pipe]    "pipe",
140   [SYS_read]    "read",
141   [SYS_kill]    "kill",
142   [SYS_exec]    "exec",
143   [SYS_fstat]   "fstat",
144   [SYS_chdir]   "chdir",
145   [SYS_dup]     "dup",
146   [SYS_getpid]  "getpid",
147   [SYS_sbrk]    "sbrk",
148   [SYS_sleep]   "sleep",
149   [SYS_uptime]  "uptime",
150   [SYS_open]    "open",
151   [SYS_write]   "write",
152   [SYS_mknod]   "mknod",
153   [SYS_unlink]  "unlink",
154   [SYS_link]    "link",
155   [SYS_mkdir]   "mkdir",
156   [SYS_close]   "close",
157   [SYS_date]    "date",
158 };
```

打开date.h 观察rtcdate结构体

```
1. vim date.h (ssh)
  1 struct rtcdate {
  2   uint second;
  3   uint minute;
  4   uint hour;
  5   uint day;
  6   uint month;
  7   uint year;
  8 };
~
```

打开lapic.c，观察 `cmostime()`方法，需要的参数为struct rtcdate *r

```
1. vim lapic.c (ssh)
194 // qemu seems to use 24-hour GWT and the values are BCD encoded
195 void
196 cmostime(struct rtcdate *r)
197 {
198   struct rtcdate t1, t2;
199   int sb, bcd;
200
201   sb = cmos_read(CMOS_STATB);
202
203   bcd = (sb & (1 << 2)) == 0;
204
205   // make sure CMOS doesn't modify time while we read it
206   for(;;) {
```

在提供的date.c 代码中引入了user.h 和 types.h，打开这两个文件观察。在user.h 中也有 syscall 的相关代码。所以应当在user.h中仿照结构体stat，增加date

```
 1 struct stat;
 2 struct rtcdate
 3
 4 // system calls
 5 int fork(void);
 6 int exit(void) __attribute__((noreturn));
 7 int wait(void);
 8 int pipe(int*);
 9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int
25 int uptime(void);
26 int date(struct rtcdata *);
```

```
26 int date(struct rtcdate *);
```

1. vim types.h (ssh)

```
1 typedef unsigned int   uint;
2 typedef unsigned short ushort;
3 typedef unsigned char  uchar;
4 typedef uint pde_t;
```

使用grep -n uptime *.[chS]，发现除了`syscall.c` 和 `user.h` 文件，`sysproc.c` 和 `usys.S`也涉及到系统调用。

```
➜  xv6-public git:(master) ✗ grep -n uptime *.[chS]
syscall.c:107:extern int sys_uptime(void);
syscall.c:124:[SYS_uptime]  sys_uptime,
syscall.c:149:  [SYS_uptime]  "uptime",
syscall.h:15:#define SYS_uptime 14
sysproc.c:83:sys_uptime(void)
user.h:25:int uptime(void);
usys.S:31:SYSCALL(uptime)
```

打开usys.S，加入date

```
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(date)
```

修改sysproc.c，参照函数sys_kill，先判断能否取到参数，不能则返回-1，能则使用cmostime获得时间。

```
30 sys_kill(void)
31 {
32   int pid;
33
34   if(argint(0, &pid) < 0)
35     return -1;
36   return kill(pid);
37 }
```

```
 93 int
 94 sys_date(struct rtcdate *r)
 95 {
 96   if(argptr(0,(void *)&r,sizeof(*r))<0)
 97     return -1;
 98   cmostime(r);
 99   return 0;
100 }
```

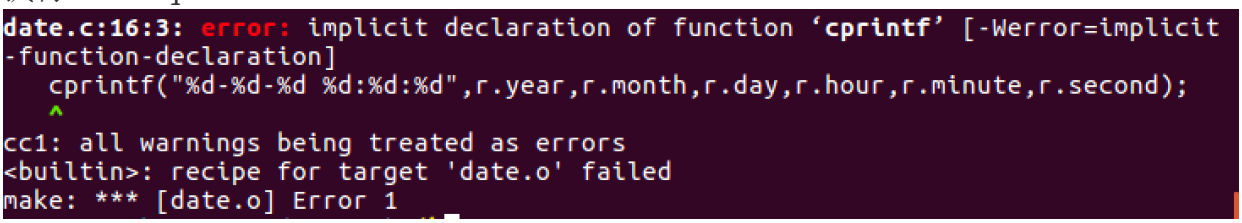
最后在date.c 中按照yyyy-MM-dd'T'HH:mm:ss.SSS Z格式打印出时间：

1. vim date.c (ssh)

```
 1 #include "types.h"
 2 #include "user.h"
 3 #include "date.h"
 4
 5 int
 6 main(int argc, char *argv[])
 7 {
 8   struct rtcdate r;
 9
10   if (date(&r)) {
11     printf(2, "date failed\n");
12     exit();
13   }
14
15   // your code to print the time in any format you like...
16   cprintf("%d-%d-%d %d:%d:%d",r.year,r.month,r.day,r.hour,r.minute,r.second);
17
18   exit();
19 }
```

在Makefile文件中将_date 加入UPROGS

执行make qemu



在date.c中增加打印库

运行，发现依然有报错

```
→ xv6-public git:(master) ✗ make qemu
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32
-Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o da
te.o date.c
In file included from date.c:4:0:
/usr/include/stdio.h:362:12: error: conflicting types for ‘printf’
 extern int printf (const char *__restrict __format, ...);
            ^
In file included from date.c:2:0:
user.h:34:6: note: previous declaration of ‘printf’ was here
 void printf(int, const char*, ...);
      ^
In file included from /usr/include/stdio.h:936:0,
                 from date.c:4:
/usr/include/bits/stdio2.h:102:1: error: conflicting types for ‘printf’
 printf (const char *__restrict __fmt, ...)
 ^
In file included from date.c:2:0:
user.h:34:6: note: previous declaration of ‘printf’ was here
 void printf(int, const char*, ...);
      ^
date.c:5:19: fatal error: conio.h: No such file or directory
compilation terminated.
<builtin>: recipe for target 'date.o' failed
make: *** [date.o] Error 1
```

根据提示，仿照前面的打印代码，改成使用printf，而不是cprintf，并去掉引入的库。

```
10   if (date(&r)) {
11     printf(2,"date failed\n");
12     exit();
13   }
14
15   // your code to print the time in any format you like...
16   printf(2,"%d-%d-%d %d:%d:%d",r.year,r.month,r.day,r.hour,r.minute,r.second);
17
```

发现qemu不能输入数字，所以将part1 改的代码注释掉，再次运行

```
$ date
2019-10-24 7:10:21$
```

返现$没有换行，输入时间的时候没有加回车，这样很丑，所以修改date.c，并再次运行

```
16   printf(2,"%d-%d-%d %d:%d:%d\n",r.year,r.month,r.day,r.hour,r.minute,r.second);
t 58
init: starting sh
$ date
2019-10-24 7:15:36
$ _
```

3. 实验结果

```
t 58
init: starting sh
$ date
2019-10-24 7:15:36
$ _
```

4. 实验结论

关于系统调用的过程，以自己写的date为例子。

sysproc.c实现系统调用的方法，大部分是根据情况处理异常，在正常情况下调用date.c 中引入的 user.h 中的系统调用方法，这些方法的具体实现是在proc.c。

```
104 //PAGEBREAK: 16
105 // proc.c
106 int             cpuid(void);
107 void            exit(void);
108 int             fork(void);
109 int             growproc(int);
110 int             kill(int);
111 struct cpu*     mycpu(void);
112 struct proc*    myproc();
113 void            pinit(void);
114 void            procdump(void);
115 void            scheduler(void) __attribute__((noreturn));
116 void            sched(void);
117 void            setproc(struct proc*);
118 void            sleep(void*, struct spinlock*);
119 void            userinit(void);
120 int             wait(void);
121 void            wakeup(void*);
122 void            yield(void);
```

在`sysproc.c` 中引入的`defs.h`中有`cmostime`方法，所以可以在实现date的方法中调用cmostime获得 UTC时间，cmostime的具体实现在lapic.c中。

```
1 #include "types.h"
2 #include "x86.h"
3 #include "defs.h"
4 #include "date.h"
5 #include "param.h"
6 #include "memlayout.h"
7 #include "mmu.h"
8 #include "proc.h"
9
```

```
➜  xv6-public git:(master) ✗ grep -n cmostime *
defs.h:76:void        cmostime(struct rtcdate *r);
Binary file kernel matches
kernel.asm:7332:801027d0 <cmostime>:
kernel.asm:7337:cmostime(struct rtcdate *r)
kernel.asm:7512:80102875:       78 89                   js     80102800 <cmostime+0x30>
kernel.asm:7664:801028ee:       0f 85 0c ff ff ff       jne    80102800 <cmostime+0x30>
kernel.asm:7671:801028f8:       75 78                   jne    80102972 <cmostime+0x1a2>
kernel.asm:16008:  cmostime(r);
kernel.asm:16011:80105640:      e8 8b d1 ff ff          call   801027d0 <cmostime>
kernel.asm:16026:  cmostime(r);
kernel.sym:277:801027d0 cmostime
lapic.c:196:cmostime(struct rtcdate *r)
Binary file lapic.o matches
sysproc.c:98:  cmostime(r);
Binary file sysproc.o matches
Binary file xv6.img matches
```

1. vim lapic.c (ssh)

```
1 // The local APIC manages internal (non-I/O) interrupts.
2 // See Chapter 8 & Appendix C of Intel processor manual volume 3.
3
4 #include "param.h"
5 #include "types.h"
6 #include "defs.h"
7 #include "date.h"
```

在date.c 中调用先声明一个rtcdate结构体，调用user.h 中的date方法。先判断获取时间是否成功，没有成功则打印失败并结束程序，如果获取时间成功，就打印出时间。

1. vim date.c (ssh)

```
1 #include "types.h"
2 #include "user.h"
3 #include "date.h"
4
5 int
6 main(int argc, char *argv[])
7 {
8   struct rtcdate r;
9
10  if (date(&r)) {
11    printf(2,"date failed\n");
12    exit();
13  }
14
15  // your code to print the time in any format you like...
16  printf(2,"%d-%d-%d %d:%d:%d\n",r.year,r.month,r.day,r.hour,r.minute,r.second);
17
18  exit();
19 }
```

利用grep想看date方法在哪儿实现的，并没有同名方法，猜测应该是由sysproc.c 中的sys_date方法实现。

```
➜  xv6-public git:(master) ✗ grep -r "date(struct rtcdate" *


kernel.asm:fill_rtcdate(struct rtcdate *r)
kernel.asm:fill_rtcdate(struct rtcdate *r)
kernel.asm:sys_date(struct rtcdate *r)
kernel.asm:sys_date(struct rtcdate *r)
lapic.c:fill_rtcdate(struct rtcdate *r)
sysproc.c:sys_date(struct rtcdate *r)
user.h:int date(struct rtcdate *);
```

sys_date方法先调用argptr方法判断传入的指针是否位于进程地址空间内。，如果不合理就返回-1，合理就调用cmostime获取UTC时间。

```
 93 int
 94 sys_date(struct rtcdate *r)
 95 {
 96   if(argptr(0,(void *)&r,sizeof(*r))<0)
 97     return -1;
 98   cmostime(r);
 99   return 0;
100 }
```