

操作系统原理

实 验 报 告

学 院 智能与计算学部
年 级 2019 级
班 级 留学生班
学 号 6319000359
姓 名 张明君

2020 年 12 月 5 日

天津大学

操作系统原理实验报告

题目: xv6 system calls

学院名称	智能与计算学部
专 业	计算机科学与技术
学生姓名	张明君
学 号	6319000359
年 级	2019 级
班 级	留学生班
时 间	2020. 12. 05

目 录

实验名称	1
实验目的	1
实验内容	1
实验步骤与分析	3
实验结论及心得体会	11

Homework: xv6 system calls

1. 实验目的

- **System call tracing:** Your first task is to modify the xv6 kernel to print out a line for each system call invocation.
- **Date System call:** second task is to add a new system call to xv6.

2. 实验内容

Part One: System call tracing

Your first task is to modify the xv6 kernel to print out a line for each system call invocation. It is enough to print the name of the system call and the return value; you don't need to print the system call arguments.

When you're done, you should see output like this when booting xv6:

```
...
fork -> 2
exec -> 0
open -> 3
close -> 0
$write -> 1
  write -> 1
```

That's init forking and execing sh, sh making sure only two file descriptors are open, and sh writing the \$ prompt. (Note: the output of the shell and the system call trace are intermixed, because the shell uses the write syscall to print its output.)

Hint: modify the syscall() function in syscall.c.

Optional challenge: print the system call arguments.

Part Two: Date system call

Your second task is to add a new system call to xv6. The main point of the exercise is for you to see some of the different pieces of the system call machinery. Your new system call will get the current UTC time and return it to the user program. You may want to use the helper function, `cmostime()` (defined in `lapic.c`), to read the real time clock. `date.h` contains the definition of the `struct rtcdate` struct, which you will provide as an argument to `cmostime()` as a pointer.

You should create a user-level program that calls your new date system call; here's some source you should put in `date.c`:

```
#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...

    exit();
}

```

In order to make your new `date` program available to run from the xv6 shell, add `_date` to the `UPROGS` definition in `Makefile`.

Your strategy for making a `date` system call should be to clone all of the pieces of code that are specific to some existing system call, for example the “uptime” system call. You should `grep` for `uptime` in all the source files, using `grep -n uptime *.c`.

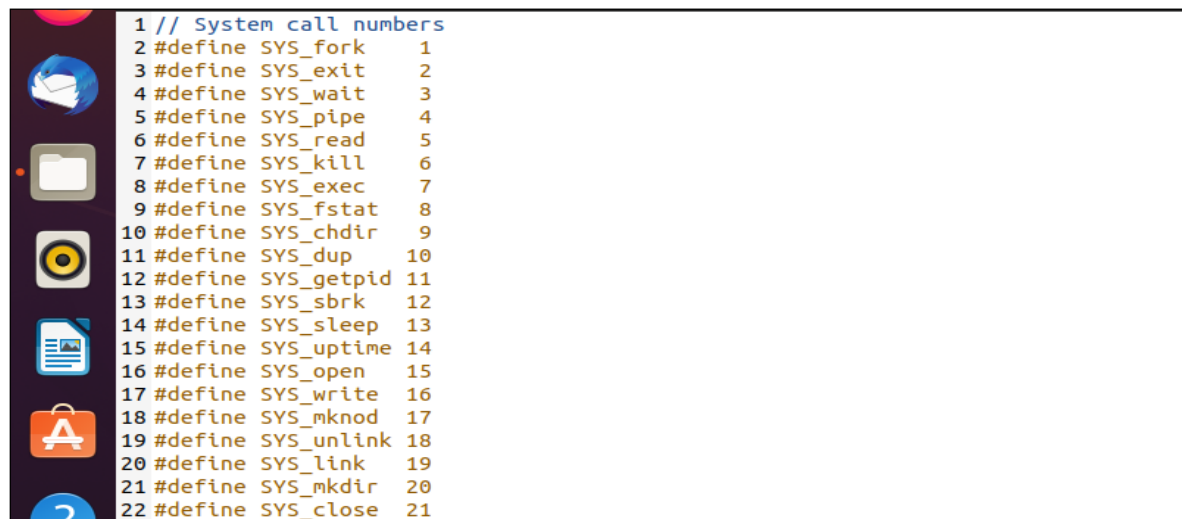
When you’re done, typing `date` to an xv6 shell prompt should print the current UTC time. Write down a few words of explanation for each of the files you had to modify in the process of creating your `date` system call.

Optional challenge: add a `dup2()` system call and modify the shell to use it.

3. 实验步骤与分析（要细化如何实现的思路或流程图）

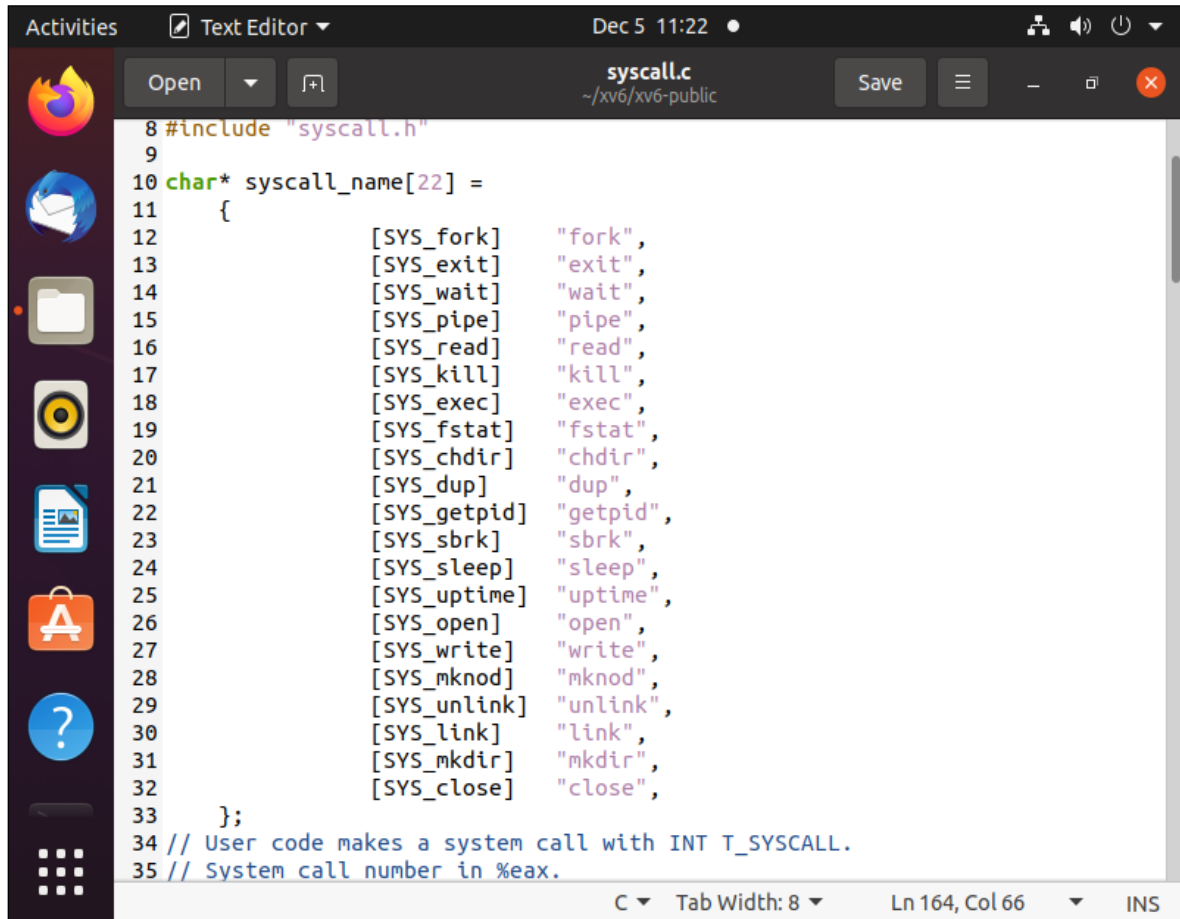
Part One: System call tracing

1/ First of all, we need to check the `syscall.h` file, which defines the name and corresponding serial number of the system call. What we want to display on the terminal is the corresponding name and number here. The `syscall.h` file below:



2/Enter the `syscall.c` file to modify it.

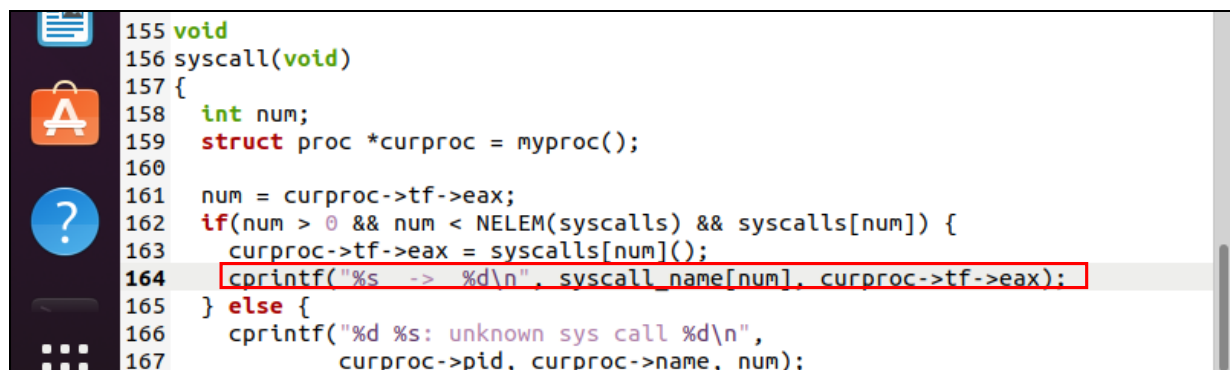
First, Add the array corresponding to the number and system call name at the beginning of the file. The code below:



```
8 #include "syscall.h"
9
10 char* syscall_name[22] =
11 {
12     [SYS_fork]    "fork",
13     [SYS_exit]    "exit",
14     [SYS_wait]    "wait",
15     [SYS_pipe]    "pipe",
16     [SYS_read]    "read",
17     [SYS_kill]    "kill",
18     [SYS_exec]    "exec",
19     [SYS_fstat]   "fstat",
20     [SYS_chdir]   "chdir",
21     [SYS_dup]     "dup",
22     [SYS_getpid]  "getpid",
23     [SYS_sbrk]    "sbrk",
24     [SYS_sleep]   "sleep",
25     [SYS_uptime]  "uptime",
26     [SYS_open]    "open",
27     [SYS_write]   "write",
28     [SYS_mknod]   "mknod",
29     [SYS_unlink]  "unlink",
30     [SYS_link]    "link",
31     [SYS_mkdir]   "mkdir",
32     [SYS_close]   "close",
33 };
34 // User code makes a system call with INT T_SYSCALL.
35 // System call number in %eax.
```

Figure 1: Add name to syscall.c

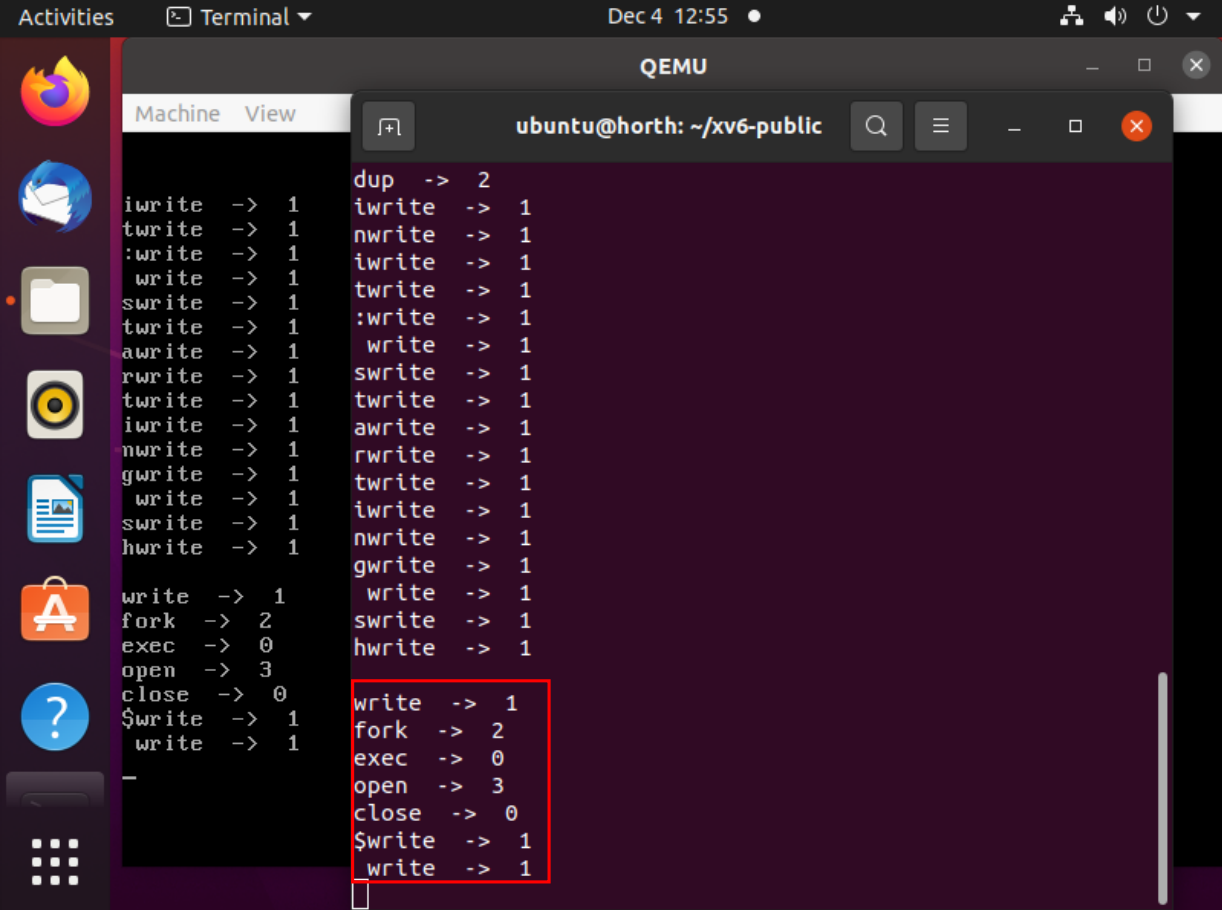
3/Then modify the corresponding `syscall` function by adding a `cprintf` statement in it, as follows:



```
155 void
156 syscall(void)
157 {
158     int num;
159     struct proc *curproc = myproc();
160
161     num = curproc->tf->eax;
162     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
163         curproc->tf->eax = syscalls[num]();
164         cprintf("%s -> %d\n", syscall_name[num], curproc->tf->eax);
165     } else {
166         cprintf("%d %s: unknown sys call %d\n",
167             curproc->pid, curproc->name, num);
168     }
```

Figure 2: Add print to call the name

4/Then run `make qemu` in terminal to produce the following results:



```
Machine View
ubuntu@horth: ~/xv6-public

iwrite -> 1
twrite -> 1
:write -> 1
write -> 1
swrite -> 1
twrite -> 1
awrite -> 1
rwrite -> 1
twrite -> 1
iwrite -> 1
nwrite -> 1
gwrite -> 1
write -> 1
swrite -> 1
hwrite -> 1

write -> 1
fork -> 2
exec -> 0
open -> 3
close -> 0
$write -> 1
write -> 1

dup -> 2
iwrite -> 1
nwrite -> 1
iwrite -> 1
twrite -> 1
:write -> 1
write -> 1
swrite -> 1
twrite -> 1
awrite -> 1
rwrite -> 1
twrite -> 1
iwrite -> 1
nwrite -> 1
gwrite -> 1
write -> 1
swrite -> 1
hwrite -> 1

write -> 1
fork -> 2
exec -> 0
open -> 3
close -> 0
$write -> 1
_write -> 1
```

Figure 3: Final answer

Part Two: Date system call

1/ First, we have to run `grep -n uptime *. [chS]` in terminal to see where we have to add the date as uptime codes ,So we have to follow the uptime to complete date in every codes.

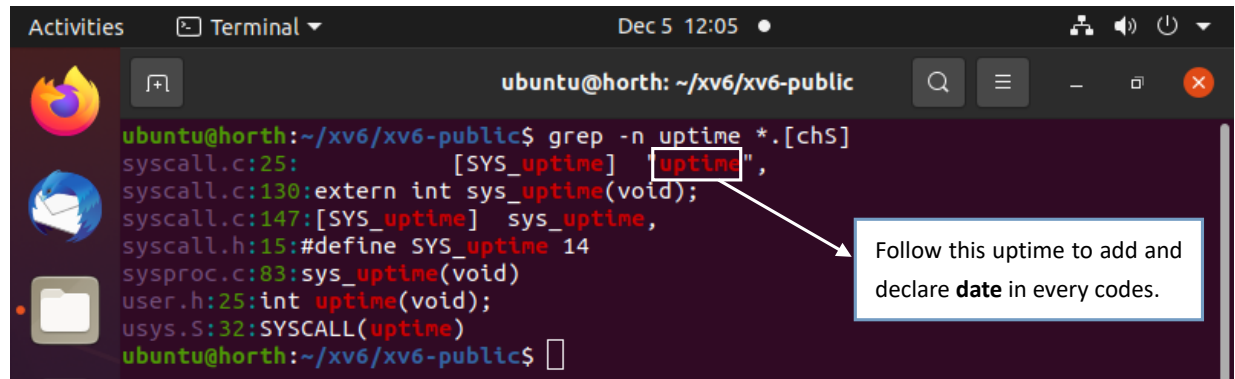


Figure 4: Find location of files

2/ Then add the number of the system call to `syscall.h`.

```
#define SYS_date 22
```

3/ Then add the declaration of the system call function in `syscall.c`.
Three statements need to be added in three places. The code is as follows:

```
[SYS_date] "date",
```

```
extern int sys_date(void);
```

```
[SYS_date] sys_date,
```

4/ Add the definition of the date function in `user.h`.

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

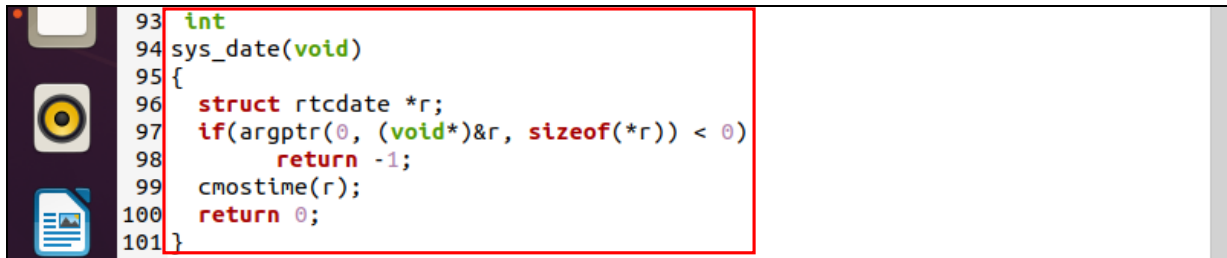
5/ In the file `usys.S` we have to add the function in the last of the line.

```
SYSCALL(date)
```

6/ Add the definition of the command corresponding to UPROGS in the `makefile` file.

```
_zombie\
//we have to add it here
_date\
```


7/ Implementation of adding system call function in **sysproc.c**.

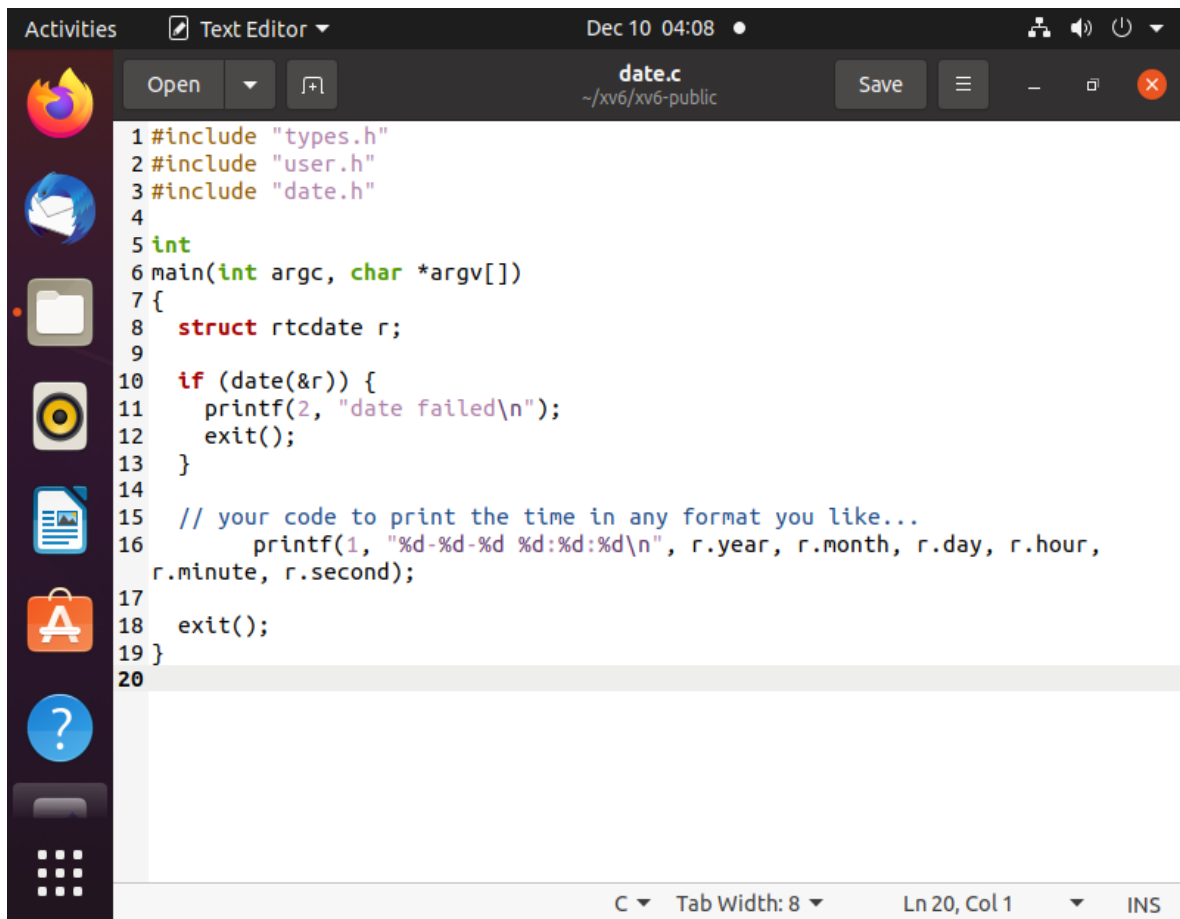


```
93 int
94 sys_date(void)
95 {
96     struct rtcdate *r;
97     if(argptr(0, (void*)&r, sizeof(*r)) < 0)
98         return -1;
99     cmostime(r);
100     return 0;
101 }
```

Figure 5: Add some functions in sysproc.c

8/ Make a new **date.c** file and then add them to xv6 file.

date.c code below:



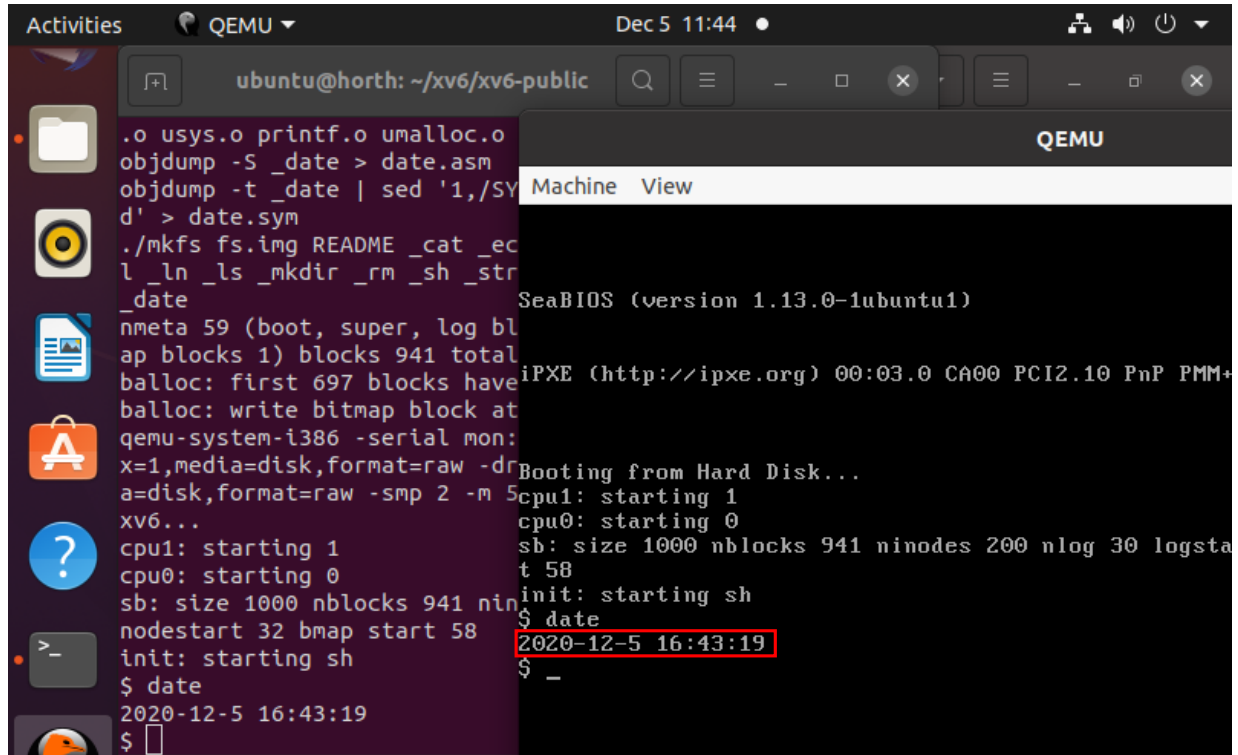
```
1 #include "types.h"
2 #include "user.h"
3 #include "date.h"
4
5 int
6 main(int argc, char *argv[])
7 {
8     struct rtcdate r;
9
10    if (date(&r)) {
11        printf(2, "date failed\n");
12        exit();
13    }
14
15    // your code to print the time in any format you like...
16    printf(1, "%d-%d-%d %d:%d:%d\n", r.year, r.month, r.day, r.hour,
17           r.minute, r.second);
18    exit();
19 }
20
```

Figure 6: Date.c file

9/ In order to prevent the output from the previous task from affecting this result, we need to comment out the code that we already did in **syscall.c** in the part one by just add

“//” before the code to unread it.

10/ Finally, enter **make qemu** in terminal to compile and run, and enter the command **date** to get the following results:



```
.o usys.o printf.o umalloc.o
objdump -S _date > date.asm
objdump -t _date | sed '1,/SY
d' > date.sym
./mkfs fs.img README _cat _ec
l _ln _ls _mkdir _rm _sh _str
_date
nmeta 59 (boot, super, log bl
ap blocks 1) blocks 941 total
ballocc: first 697 blocks have
ballocc: write bitmap block at
qemu-system-i386 -serial mon:
x=1,media=disk,format=raw -dr
a=disk,format=raw -smp 2 -m 5
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 nin
t 58
init: starting sh
$ date
2020-12-5 16:43:19
$
```

Figure 7: Final answer

4. 实验结论与心得体会

In the system call tracing part I think that it's the easiest one in this experiments. We have to modify the xv6 kernel to print out a line for each system call invocation, To do that we have to add some codes to the files and run it in the terminal it'll show you the answer.

In the second part Date system call it's not too easy until we read the hint that they gave you. Especially the trick that we have to find the location of the files that we need to change. It's very useful and easy to find the location that I wanted to change. And another tip is that we have to unread some statements in the first part because if we not unread it, it'll be interrupted when we run the qemu in the terminal. So there're many trick that you have to know before doing it.