Homework: xv6 system calls

刘恒星 2022229044

March 8, 2023

1 Part One: System call tracing

第一个任务是要让 kernel 在每一次系统调用的时候输出调用的函数名和返回值。并且提示去修改 syscall.c 的 syscall() 函数

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

可以看出，系统通过 $syscall[num]()$ 来使用系统调用，然后将返回值存回 eax 中。num 会按照规定好的宏定义去映射到预定义的数组下标，通过 $staticint(*syscalls[])(void)$ 我们可以知道调用了什么系统调用，并且按照一定顺序将系统调用的函数名存放在一个数组中，最后可以让他输出调用的函数名和返回值。

```
static char *syacall_name[] = {
    [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] = "sys_date",
};

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

效果如下：

```
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
exec -> 0
open -> -1
mknod -> 0
open -> 0
dup -> 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
```

2 Part Two: Date system call

第二个任务主要是添加一个新的系统调用。任务书中给出了这个调用的源代码的一部分，完善之后如下

```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...
    printf(1, "%04d-%02d-%02d %02d:%02d:%d\n", r.year, r.month, r.day, r.hour, r.minute,
        r.day);

    exit();
```

```
}
```

观察 rtcdate 的结构如下

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

可以看出这是一个存放时间的结构体，所以将输出语句写好即可。
接下来是将这个指令变成系统调用，根据任务书做如下步骤

1. 在 makefile 中的 UPRGOS 添加 $_date$
2. 使用 $grep -nuptime *.[chS]$ 指令仿照 uptime 来补全 date 缺失的代码

需要注意的是，当我们完成 *sysproc.c* 的 $sys_date()$ 的时候，可以使用 $cmosdate()$ 函数来获取当前时间，代码如下

```
int sys_date(void)
{
   struct rtcdate *date;
   if (argptr(0, (void *)&date, sizeof(*date)) < 0)
      return -1;
   cmostime(date);
   return 0;
}
```

效果如下：

```
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ date
2023-3-8  7:44:8
$
```