**System calls**

一、实验目的

修改 xv6 以添加系统调用

二、实验内容

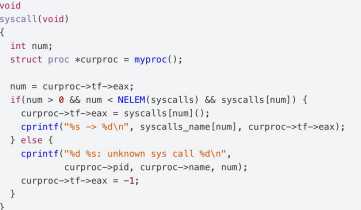
[Homework: xv6 system calls (mit.edu)](https://pdos.csail.mit.edu/6.828/2018/homework/xv6-syscall.html)

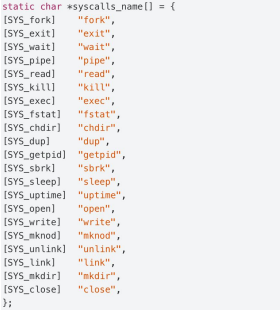
三、实验步骤与实验结论

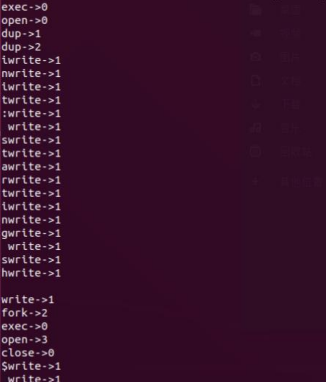
第一部分：系统调用跟踪

修改xv6的源码，打印出每一个每一个系统掉调用的流程。只要打印出system call的名字以及system call的返回值。所以，在syscall.c文件里面添加syscalls\_name数组，并修改syscall

eax寄存器保存的是需要调用的system call的下标，所以num = curproc->tf->eax;这个是获得需要调用哪个system call。然后经过一些安全检查后,使用num来从syscalls当中获得对应system call，syscalls里面都是函数指针，所以syscalls[num]()相当于直接执行函数,返回值放在eax寄存器(x86 calling convention)当中。为了获得system call的名字，我们首先需要也建一个数组，和上面的syscalls







第二部分：日期系统调用

(1) 在mkaefile 文件的UPROGS中添加\_date

(2) 新建文件date.c

(3) 在 user.h 中添加 int date(struct rtcdate\*);

(4) 在文件 usys.S 中添加 SYSCALL(date)

(5) 在 syscall.h 中添加#define SYS\_date 22

(6) 在 syscall.c 中添加[SYS\_date] sys\_date,

(7) 在 sysproc.c 中添加

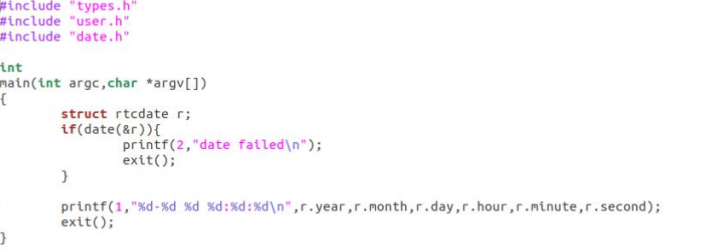
注意：在trap.c的trap()中调用了syscall()。 最重要的一点就是在usys.S中加入一行SYSCALL(date)。当用户在调用date()时，会自动陷入并执行对应的sys\_date()。后在makefile对应的UPROGS添加\_date即可

date.c文件加入时间格式输出

argptr函数：这个函数是用来抓取系统调用的参数的。换言之，我们在date函数中，有一个结构体参数struct rtcdate，因此这里就可以通过argptr来获得传递进来的参数，然后接下来利用cmostime函数，最终让rtcdate得到时间数据。于是在之后切换回用户态时，此时我们的date.c文件中的main函数即可进行使用。于是我们就完成了这个系统调用的代码工作。

当调用dup函数时，内核在进程中创建一个新的文件描述符，此描述符是当前可用文件描述符的最小数值，这个文件描述符指向oldfd所拥有的文件表项。

dup2和dup的区别就是可以用newfd参数指定新描述符的数值，如果newfd已经打开，则先将其关闭。如果newfd等于oldfd，则dup2返回newfd, 而不关闭它。dup2函数返回的新文件描述符同样与参数oldfd共享同一文件表项。在sysfile.c文件下，模仿sys\_dup()可写出以下函数。







四、反思总结

在第二部分的实践中，最开始出来的结果会带有第一部分的过程。将第一部分的代码删去之后，就可以得到正常的结果

proc结构体里的name是整个线程的名字，不是函数调用的函数名称，所以我们不能用p->name，而要自己定义一个数组