哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验设计报告

开课学期：2022 年秋季

课程名称：操作系统

实验名称：系统调用

实验性质：课内实验

实验时间：9.28 地点：T2507

学生班级：6

学生学号：200110631

学生姓名：张景昊

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2022 年 9 月

一、 回答问题

1. 阅读 kernel/syscall.c，试解释函数 syscall()如何根据系统调用号调用对应的系统调用处理函数（例如 sys_fork）？syscall()将具体系统调用的返回值存放在哪里？

答：

1. syscall.h 文件将系统调用函数与作为系统调用号的特定整型变量进行映射
2. syscall.c 文件中，声明了 extern 函数接口和 syscall 数组。extern 接口对各系统调用函数进行了声明，以便在其他文件中进行具体实现；syscall 数组存储所有的系统调用函数指针，指针对应的数组下标为函数对应的系统调用号
3. syscall()函数根据 syscall.c 中特定系统调用号对应的数组下标，调用对应的系统调用函数指针，以此实现调用对应的系统调用处理函数

系统调用的返回值存放在：

trapframe 中的 a0 寄存器

2. 阅读 kernel/syscall.c，哪些函数用于传递系统调用参数？试解释 argraw()函数的含义。

答：

用于传递系统调用参数的函数有：

argraw(), argint(), argaddr(), argstr()

argraw()函数的含义：
声明一个指向当前进程 PCB 的指针 p；根据输入的形参 n，将指针指向对应寄存器 a1-a5，以此获取寄存器中的值并返回，实现参数的传递。若出现异常则调用 panic 函数，返回-1

3. 阅读 kernel/proc.c 和 proc.h，进程控制块存储在哪个数组中？进程控制块中哪个成员指示了进程的状态？一共有哪些状态？

答：

进程控制块存储在 proc.c 中的 proc[NPROC]数组中

进程控制块的 p->state 指示进程的状态

共有 UNUSED、SEPPING、RUNNABLE、RUNNING、ZOMBIE 五种状态

4. 阅读 kernel/kalloc.c，哪个结构体中的哪个成员可以指示空闲的内存页？Xv6 中的一个页有多少字节？

答：

指示空闲内存页的成员：kmem 结构体中的 freelist

XV6 中的一个页有 4096 个字节

5. 阅读 kernel/vm.c，试解释 copyout()函数各个参数的含义。

答：
pagetable_t pagetable：进程页表
uint64 dstva：用户态的目标地址
*char *src：数据源地址*
uint64 len：数据大小

二、 实验详细设计

1. 系统调用 trace：

（1）对应用户端函数参数：mask 标识要追踪的系统调用号

（2）功能：根据 mask 追踪特定的系统调用；在处理完该系统调用后，打印特定信息

（3）具体实现：

• 首先在 MakeFile 中的 UPROGS 下新增$U/_trace，随后在系统调用的各相关部分（user.h、usys.pl、syscall.h、syscall.c）进行关于 trace 的声明；

•在 sysproc.c 中定义系统调用函数 uint64 sys_trace(void)，并在 PCB 中增加 int 型变量变量 trace_mask 来保存系统调用 trace 的形参 mask；

• 考虑到子进程的问题，需要修改 proc.c 文件中的 fork()函数以保证子进程能够开启 trace 并继承进程的 mask

• 最后，在 syscall.c 文件中添加一个存储系统调用名称的字符串数组；在 syscall()函数添加特定存储语句，利用该数组即可实现输出题目要求的信息。

2. 系统调用 sysinfo:

（1）对应用户端函数参数：void

（2）功能：对已经完成定义的结构体 sysinfo 的各成员（剩余内存字节、未使用进程数、可用的文件描述符数量）填入对应值

（3）具体实现：

• 首先在 MakeFile 中的 UPROGS 下新增$U/_sysinfotest，随后在系统调用的各相关部分（user.h、usys.pl、syscall.h、syscall.c）进行关于 sysinfo 的声明；

• 在 kalloc.c 和 proc.c 两个文件中，分别实现三个数据收集函数以实现功能：

get_freemem:空闲内存由 kmem.freelist 链表存储。因此只需遍历列表，计数并乘以页表大小即可得到所需数据

get_nproc:进程存放于 proc[NPROC]数组中只需遍历数组并记录状态为

UNUSED 的进程数即可

get_freefd:文件描述符对应的是 PCB 块中的 ofile 数组下标。因此只需遍历该数组并记录数组中值为 0 的元素个数即可。

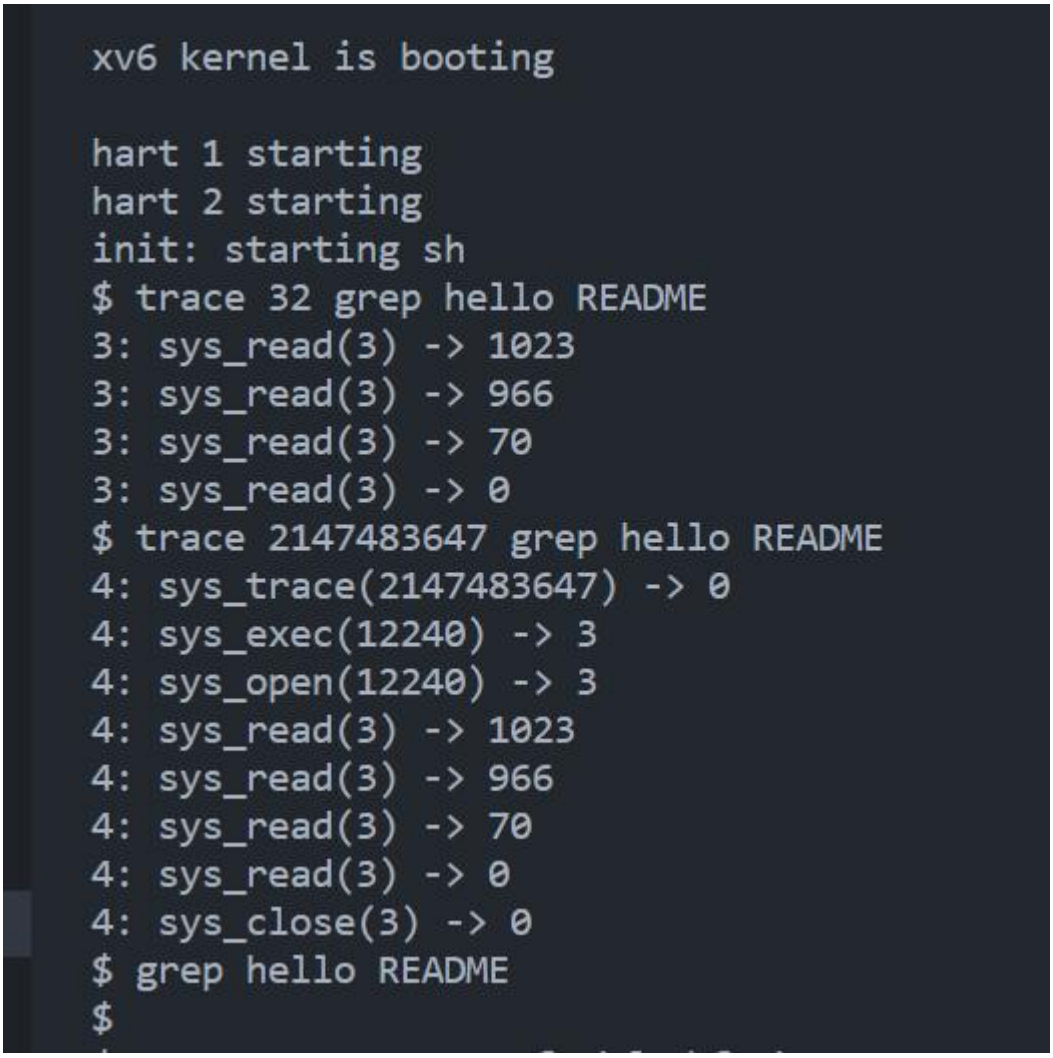
前两个函数涉及到利用指针对链表进行操作，因此需要加锁来保证遍历能够得到正确的结果。

• 在实现了三个工具函数后，参考 sys_fstat 函数，利用它们实现具体的 sys_sysinfo 系统调用函数；其中利用 copyout()函数将内核数据传输到用户态

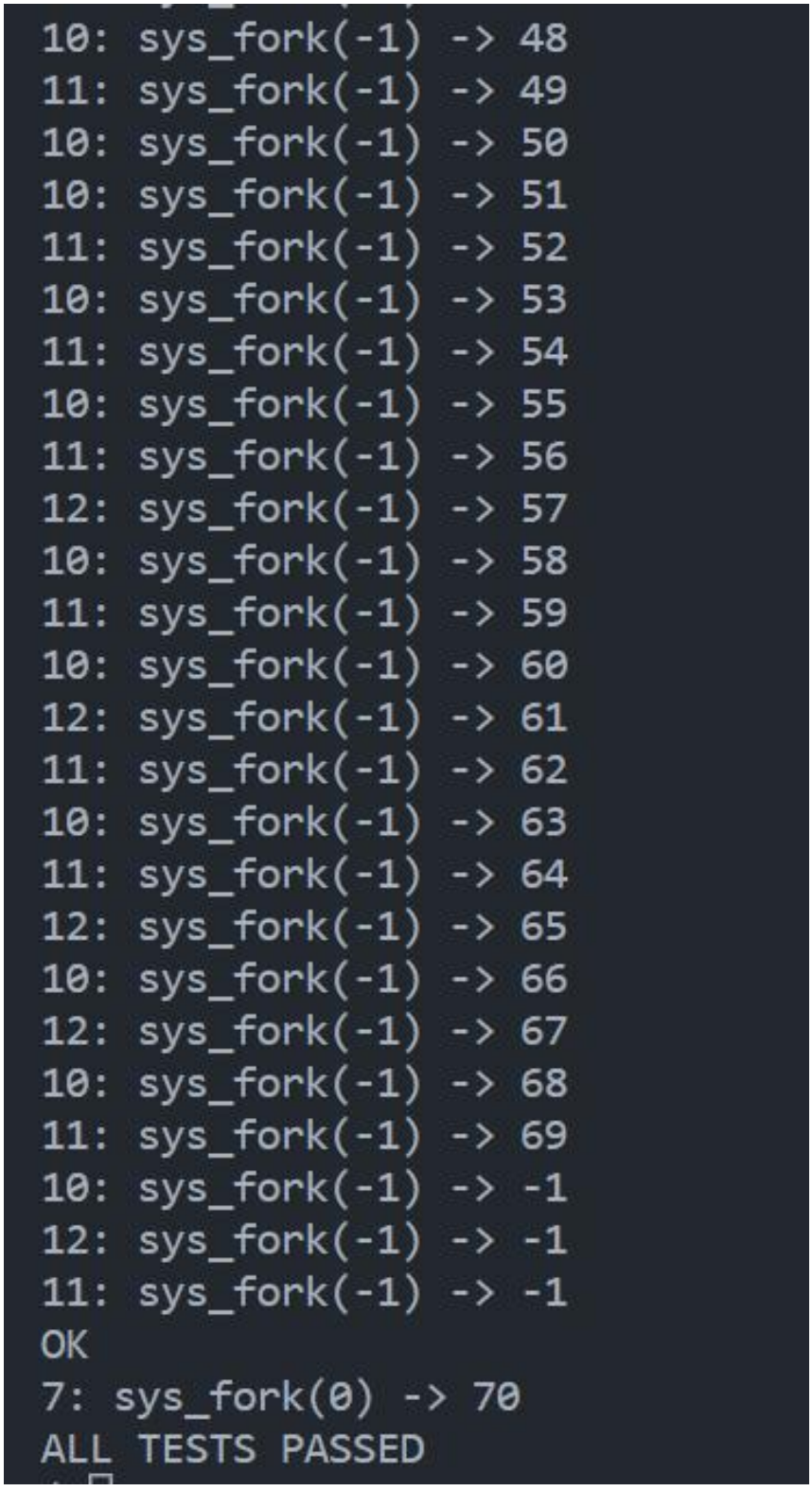
在实现过程中，出现了一个及其奇怪却又有趣的 bug：已经定义好的 sysinfo.h 头文件的三个成员的数据类型标识符 uint64 突然报错“找不到定义”且不知原因，间接影响了编译。学生在和老师同学交流很久后仍然无法解决，遂采用了最原始的方法：手动添加头文件。因而，在学生编写函数修改过的某些文件顶部，仍有头文件的痕迹。私以为可能是编译过程中机器出现了问题，导致头文件的引入顺序错误，因而导致找不到定义报错

三、 实验结果截图

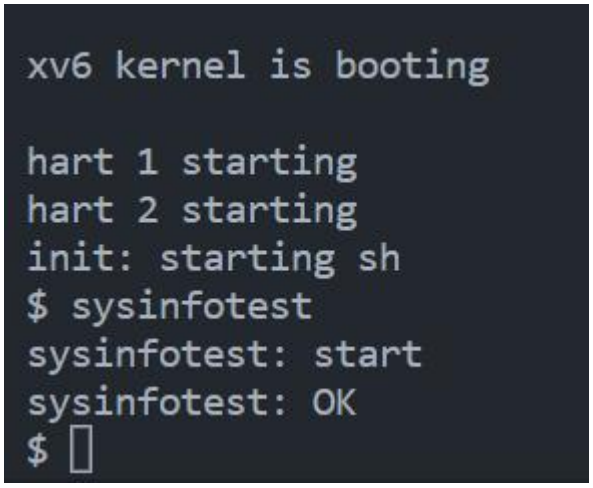
根据实验指导书首先对系统调用 trace 进行检查，运行结果如下



```
$ trace 2 usertests forkforkfork
usertests starting
7: sys_fork(0) -> 8
test forkforkfork: 7: sys_fork(1) -> 9
9: sys_fork(-1) -> 10
10: sys_fork(-1) -> 11
10: sys_fork(-1) -> 12
11: sys_fork(-1) -> 13
10: sys_fork(-1) -> 14
11: sys_fork(-1) -> 15
10: sys_fork(-1) -> 16
11: sys_fork(-1) -> 17
10: sys_fork(-1) -> 18
12: sys_fork(-1) -> 19
10: sys_fork(-1) -> 20
11: sys_fork(-1) -> 21
10: sys_fork(-1) -> 22
11: sys_fork(-1) -> 23
10: sys_fork(-1) -> 24
12: sys_fork(-1) -> 25
10: sys_fork(-1) -> 26
11: sys_fork(-1) -> 27
10: sys_fork(-1) -> 28
11: sys_fork(-1) -> 29
10: sys_fork(-1) -> 30
11: sys_fork(-1) -> 31
10: sys_fork(-1) -> 32
11: sys_fork(-1) -> 33
11: sys_fork(-1) -> 34
11: sys_fork(-1) -> 35
11: sys_fork(-1) -> 36
10: sys_fork(-1) -> 37
11: sys_fork(-1) -> 38
12: sys_fork(-1) -> 39
10: sys_fork(-1) -> 40
11: sys_fork(-1) -> 41
10: sys_fork(-1) -> 42
11: sys_fork(-1) -> 43
10: sys_fork(-1) -> 44
11: sys_fork(-1) -> 45
10: sys_fork(-1) -> 46
11: sys_fork(-1) -> 47
10: sys_fork(-1) -> 48
```


10: sys_fork(-1) -> 48
11: sys_fork(-1) -> 49
10: sys_fork(-1) -> 50
10: sys_fork(-1) -> 51
11: sys_fork(-1) -> 52
10: sys_fork(-1) -> 53
11: sys_fork(-1) -> 54
10: sys_fork(-1) -> 55
11: sys_fork(-1) -> 56
12: sys_fork(-1) -> 57
10: sys_fork(-1) -> 58
11: sys_fork(-1) -> 59
10: sys_fork(-1) -> 60
12: sys_fork(-1) -> 61
11: sys_fork(-1) -> 62
10: sys_fork(-1) -> 63
11: sys_fork(-1) -> 64
12: sys_fork(-1) -> 65
10: sys_fork(-1) -> 66
12: sys_fork(-1) -> 67
10: sys_fork(-1) -> 68
11: sys_fork(-1) -> 69
10: sys_fork(-1) -> -1
12: sys_fork(-1) -> -1
11: sys_fork(-1) -> -1
OK
7: sys_fork(0) -> 70
ALL TESTS PASSED

根据实验指导书，运行 sysinfotest 函数对 sysinfo 系统调用进行检查，运行结果如下：



在命令行使用 make grade 命令对两个系统调用进行检查，结果如下：