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：9.30 地点：T2210

学生班级：20 级 08 班

学生学号：200210231

学生姓名：王木一

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？

在 kernel/syscall.c 中定义了一个静态的函数指针数组 syscalls[],其中的元素就是每个系统调用函数的地址，每个元素（系统调用函数）的下标对应的是 kernel/syscall.h 中定义的系统调用号。使用时直接用系统调用号索引即可调用对应的处理函数。

返回值存放在当前进程 p 的 trapframe 的 a0 号寄存器中，p->trapframe->a0 = sys_($name)()

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。

Kernel/syscall.c 中用于传递系统调用参数的有：

- int argstr(int n, char *buf, int max)
- int argaddr(int n, uint64 *ip)
- int argint(int n, int *ip)
- static uint64 argraw(int n)
- int fetchstr(uint64 addr, char *buf, int max)
- int fetchaddr(uint64 addr, uint64 *ip)

static uint64 argraw(int n)函数：

从当前进程 p 的 trapframe 的 an 号寄存器（即 p->trapframe->an）获取传入的参数，这里返回的值为 uint64 类型，具体使用时转换为哪种类型由调用此函数的函数（如 argint()）决定（故此函数被命名为“raw”，因为其返回的是原始数据）。

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？

Xv6 的进程控制块 PCB 用结构体 struct proc（位于 kermel/proc.h）表示，进程控制快存储在结构体数组 proc（被定义为 struct proc proc[NPROC]）（位于 kernel/proc.c）中

PCB（即结构体 struct proc）中的 state 成员指示（state 的类型为枚举类型 enum procstate）

根据 enum procstate 定义，一个进程有 5 种状态，分别为{UNUSED, SLEEPING, RUNNABLE, RUNNING, ZOMBIE}

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？

结构体 kmem（struct kmem）中的 freelist 成员（结构体链表 struct run *freelist）指示空闲的内存页。链表的长度即空闲页表的页数。

根据宏 PGSIZE（kernel/riscv.h）的定义，每页共 4096 字节。

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

由于某些函数使用指针传输数据，而内核页表和用户页表不同，故需要 copyout()用于从 kernel 向 user 拷贝数据（相反的，也有 copyin()函数）

```
int copyout(pagetable_t pagetable, uint64 dstva, char *src, uint64 len)
```

pagetable: 用于检索用户目标地址的页表（用户页表）

dstva: 用户提供的虚拟地址（用于存储返回值的目标地址）

src: 待拷贝数据位于内核的地址

len: 待拷贝数据的长度（byte）

二、 实验详细设计

注意不要照搬实验指导书上的内容，请根据你自己的设计方案来填写

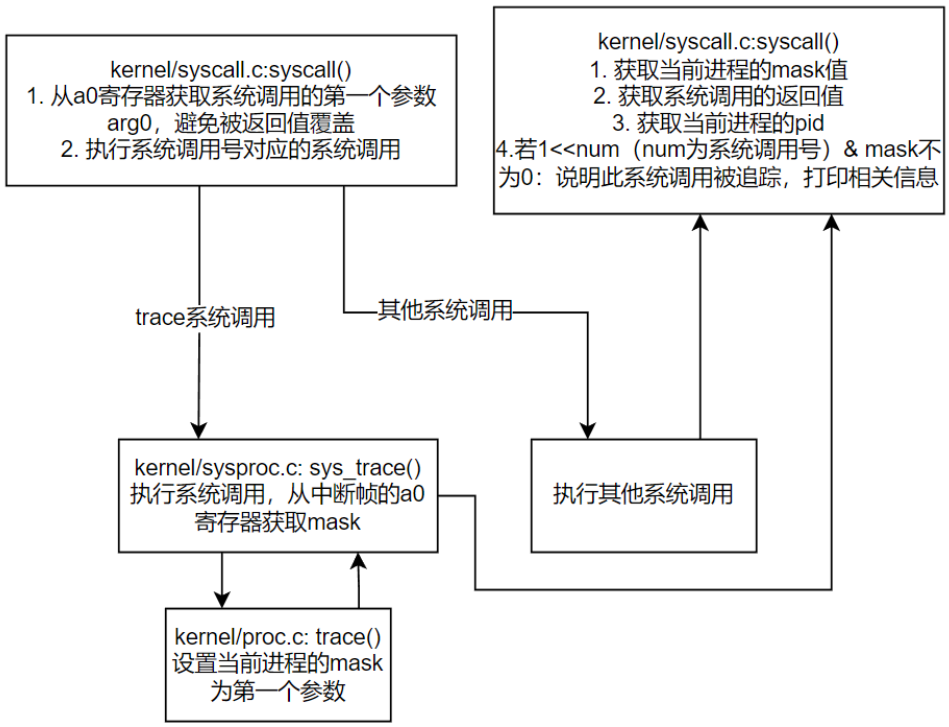
2.1 trace

• 前序工作：

1. 在 PCB（即 struct proc(kernel/proc.h)中，增加元素 mask，用于标志是否需要 trace，以及需要 trace 哪些系统调用
2. 在 fork()(kernel/proc.c)函数中增加子进程继承父进程 mask 的逻辑，以便子进程也能进行 trace。同时 mask 初值设为 0，此步在 procinit()（kernel/proc.c）函数中实现。

• 核心逻辑

下图为进入内核态后，从 syscall()开始的执行逻辑：



trace 系统调用的核心功能是设置当前进程的 mask 值。每次系统调用返回后（syscall.c）都会利用系统调用号和当前进程的 mask 进行比对，以得出此系统调用是否被追踪，是否需要打印相关信息。

部分代码：

```
//set mask for syscall TRACE
int
trace(uint64 mask){
  struct proc *p = myproc();

  p->mask = mask;
  return 0;
```

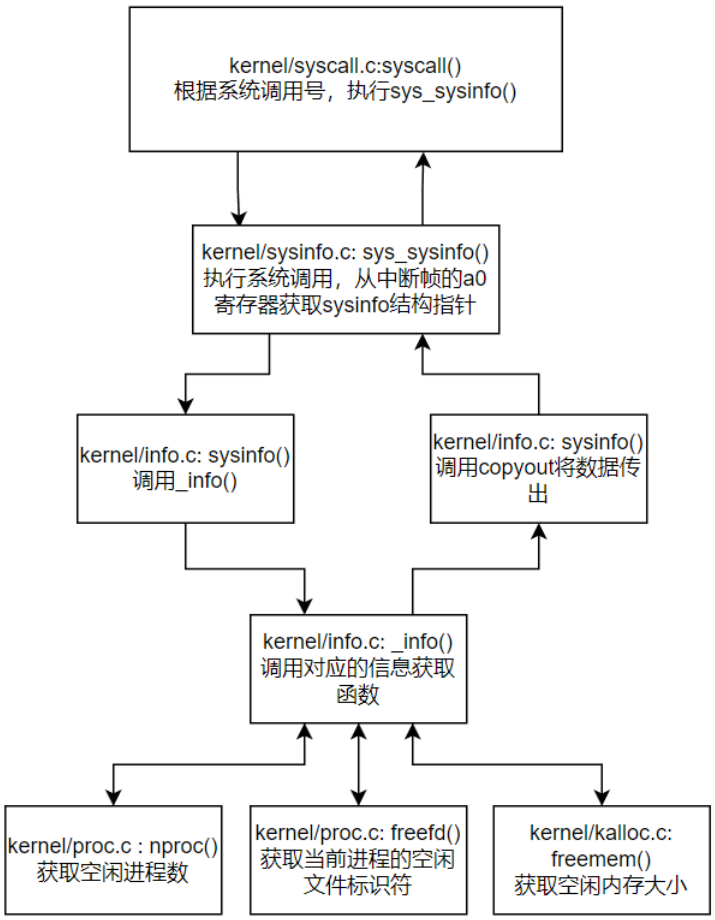
```
}
    //print trace infomation
    mask = p->mask;
    return_value = p->trapframe->a0;
    pid = p->pid;
    mask_try = 1 << num;
    if(mask_try & mask){
      printf("%d: %s(%d) -> %d\n", pid, syscall_names[num], arg0,
return_value);
    }
```

• 后续工作

设置 Makefile, kernel/defs.h, kernel/syscall.h, user/user.h, user/usys.pl

2.2 sysinfo

• 核心逻辑



1. 空闲进程数：遍历结构体数组 proc，计算其中进程状态为 UNUSED 的数目
2. 空闲文件标识符：PCB（即结构体 struct proc）用指针数组 ofile（元素为 struct file*）

保存已打开的文件结构体指针。用此数组大小（NOFILE）减去其中不为 0 的元素个数即为空闲的文件标识符数目

3. 空闲内存大小：kernel/kalloc.c 中结构体 kmem（struct kmem）中的 freelist 成员（结构体链表 struct run *freelist））指示空闲的内存页。链表的长度即空闲页表的页数。页数乘以每页大小（PGSIZE）即为空闲内存大小

部分代码：

```
//get the number of free file descriptors of current process
uint64
freefd(void){
  struct proc *p = myproc();
  // struct file **f;
  int num = 0;
  for(int i = 0; i < NOFILE; i++){
    if(p->ofile[i]){
      num++;
    }
  }
  return NOFILE - num;
};

//get the number of UNUSED processes
uint64
nproc(void){
  struct proc *p;
  int num = 0;
  for(p = proc; p < &proc[NPROC]; p++) {
    if(p->state == UNUSED)
      num++;
  }
  return num;
};
//get how many bytes are free in memory
uint64
freemem(void){
  int i = 0;
  struct run *r;
  r = kmem.freelist;
  while(r){
    i++;
    r = r->next;
  }
  return i * PGSIZE;
}
```

• 后续工作

设置 Makefile, kernel/defs.h, kernel/syscall.h, user/user.h, user/usys.pl

三、 实验结果截图

请填写