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

# 实验设计报告

开课学期：2021 秋

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：4 地点：T2608

学生班级：19 级计算机 10 班

学生学号：190111005

学生姓名：倪洪达

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2018 年 12 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：syscall 中有一个表单存放着各个系统调用号对应的函数，当接收到系统调用号的时候，就根据调用号在表单中查询对应的函数。返回值被存放在 a0 寄存器中。

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：argint()、argstr()和 argaddr()都可用于传递系统调用参数。argraw()可以去读寄存器 a0 到 a5 的值，这里面不同的寄存器分别对应不同的函数调用。

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：PCB 存储在 proc 数组中，proc -> state 指示了进程状态。一共有 UNUSED、SLEEPING、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：结构体 kmem 的 freelist 可以指示空闲的内存页。Xv6 中，一个页有 PGSIZE=4096 个字节。

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

   答：第一个参数是待复制字符串的页表号，第二个参数是页内地址，第三个参数是待复制的字符串的首地址，第四个参数是待复制的字符串的长度。

# 二、 实验详细设计

1. trace

   首先需要在一切包含系统调用名称的地方加上 trace 相关的声明，如系统调用号和函数表等。接着需要在 syscall.c 中加入对应的系统调用分发的逻辑，需要打印出当前 a0 寄存器的值和执行系统调用后 a0 寄存器的值，再用一个数组存储系统调用名称：

```
num = p->trapframe->a7;
int i = p->trapframe->a0;
if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
```

```
    p->trapframe->a0 = syscalls[num]();
  } else {
    printf("%d %s: unknown sys call %d\n",
            p->pid, p->name, num);
    p->trapframe->a0 = -1;
  }
  if (p->mask & (1 << num))
      printf("%d: sys_%s(%d) -> %d\n", p -> pid, calls[num - 1], i, p -> trapframe
-> a0);
```

其中数组 calls 也在 syscall.c 中定义：

```
static char *calls[32] = {
  "fork",
  "exit",
  "wait",
  "pipe",
  "read",
  "kill",
  "exec",
  "fstat",
  "chdir",
  "dup",
  "getpid",
  "sbrk",
  "sleep",
  "uptime",
  "open",
  "write",
  "mknod",
  "unlink",
  "link",
  "mkdir",
  "close",
  "trace",
  "sysinfo"
};
```

另外需要实现 sys_trace()中的对应逻辑：

```
sys_trace(void)
{
  int mask;
  if (argint(0, &mask) < 0) return -1;
  struct proc *p = myproc();
  p -> mask = mask;
  return 0;
}
```

要点在于使 trace 记住进程的 mask。另外 fork 函数需要修改，需要添加一句 np->mask = p->mask 从而将 mask 的值传递下去。

2. sysinfo

首先和 trace 一样，在各个包含系统调用的地方添加声明，然后在 sysproc 中添加 sys_sysinfo 函数定义：

```
uint64
sys_sysinfo(void)
{
  struct sysinfo s;
  uint64 ip; //user pointer to struct sysinfo
  struct proc *p = myproc();
  s.freemem = freemem();
  s.nproc = nproc();
  s.freefd = freefd();
  if (argaddr(0, &ip) < 0) return -1;
  if (copyout(p -> pagetable, ip, (char *)&s, sizeof(s)) < 0) return -1;
  return 0;
}
```

这里仿照 fstat 相关逻辑，使用 argaddr 函数以及 copyout 函数，将结构体内容从内核地址空间复制到用户地址空间。另外传递待实现的三个函数的值：

① freemem：

```
uint64 freemem(void)
{
  struct run *r;
  acquire(&kmem.lock);
  r = kmem.freelist;
  release(&kmem.lock);
  uint64 n = 0;
  while (r)
  {
    r = r -> next;
    n += PGSIZE;
  }
  return n;
}
```

使用指针 r 按页遍历剩余的内存空间，再用变量 n 进行累加，最终返回即可。

② nproc：

```
uint64 nproc(void)
{
  struct proc *p;
  uint64 n = 0;
  for (p = proc; p < &proc[NPROC]; p++)
  {
    acquire(&p -> lock);
```

```
    if(p -> state == UNUSED) n++;
    release(&p -> lock);
  }
  return n;
}
```

就遍历进程控制块中所有进程，遇到 UNUSED 状态的就计数变量+1，最终返回即可。

③ freefd：

```
uint64 freefd(void)
{
  struct proc *p = myproc();
  uint64 n = 0;
  int i = 0;
  while (i != NOFILE)
  {
    acquire(&p -> lock);
    if (!p -> ofile[i]) ++n;
    release(&p -> lock);
    ++i;
  }
  return n;
}
```

实际上就是统计当前进程的 ofile 值为 0 的个数，注意锁的使用即可。

## 三、 实验结果截图