Operating System Lab 2 Report

by PB18000227 艾语晨

添加Linux系统调用

大概描述实验过程

增加系统调用需要修改相关文件，和具体的函数实现

就是按照实验文档所说的，先看一下后面的背景知识，并借助上一次习题课的讲解，最终完成

展示实验结果

```
[/usr]# ./test
Give me a string:
123
in sys_print_val: 123
[/usr]#
```

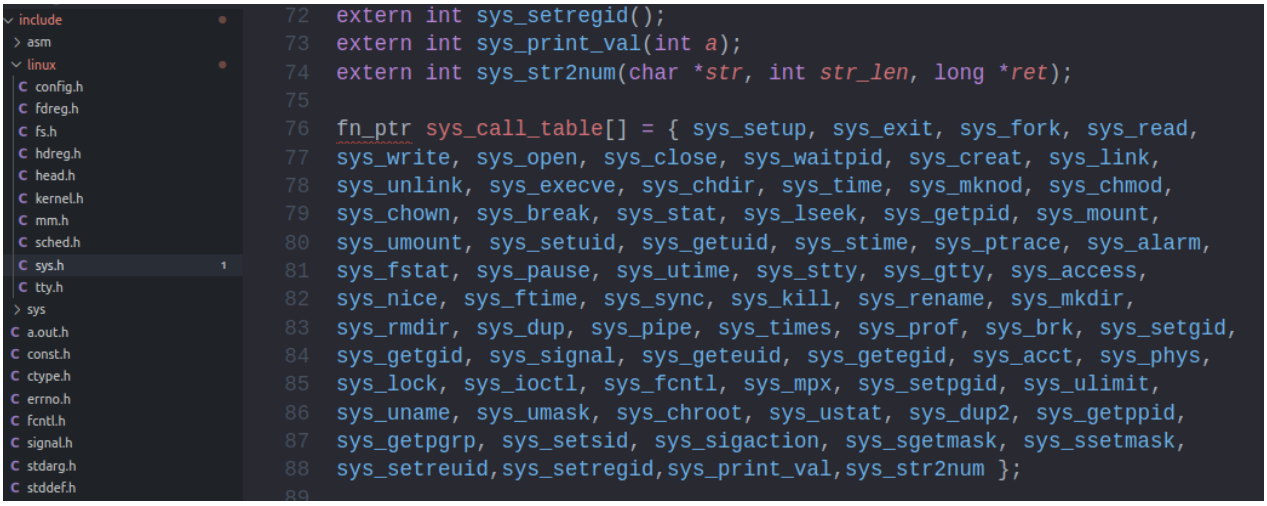
实验代码（改变部分）

`unistd.h`，左边的是include文件夹下，右边的是文件系统里面

```
130  #define __NR_setreuid   70
131  #define __NR_setregid   71
132  #define __NR_print_val  72
133  #define __NR_str2num    73
134
```

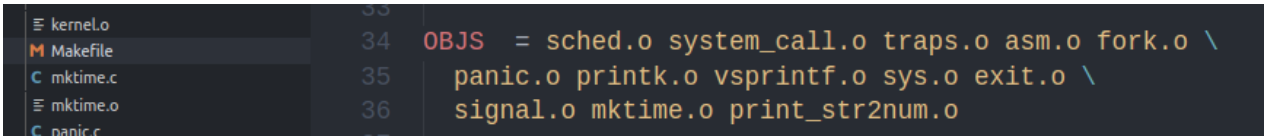
```
130  #define __NR_setreuid   70
131  #define __NR_setregid   71
132  #define __NR_print_val  72
133  #define __NR_str2num    73
134
```

`sys.h`



`system_call.s`

`Makefile`




```
../include/asm/system.h ../include/asm/segment.h ../include/asm/io.h
vsprintf.s vsprintf.o: vsprintf.c ../include/stdarg.h ../include/string.h
print_str2num.s print_str2num.o: print_str2num.c ../include/asm/segment.h

```

`print_str2num.c`

print_str2num.c

kernel > print_str2num.c > __LIBRARY__

```c
#define __LIBRARY__
#include <linux/kernel.h>
#include <asm/segment.h>

int sys_print_val(int a)
{
    printk("in sys_print_val: %d", a);
}

int sys_str2num(char *str, int str_len, long *ret)
{
    /* *ret = (long)atoi(str); */
    long sum = 0;
    char temp;
    for (int i = 0; i < str_len; i++)
    {
        temp = get_fs_byte(str + i);
        sum = sum * 10 + temp - '0';
    }
    put_fs_long(sum, ret);
}

```

回答问题

- 简要描述如何在Linux-0.11添加一个系统调用

> Answer :

1. 修改**system_call.s**中的系统调用数量
2. 在**sys.h**、**unistd.h**中增加相应的声明、函数指针表内容
3. 在kernel目录下实现相应的函数，并修改makefile

- 系统是如何通过系统调用号索引到具体的调用函数的

> Answer :

在**unistd.h**中有系统调用编号，`_syscallx`函数根据这个编号到**sys.h**中的函数指针表找到相应函数原型，然后在kernel目录下找到具体的实现

- 在Linux-0.11中，系统最多支持几个参数？有什么方法可以超过这个限制吗？

> Answer :

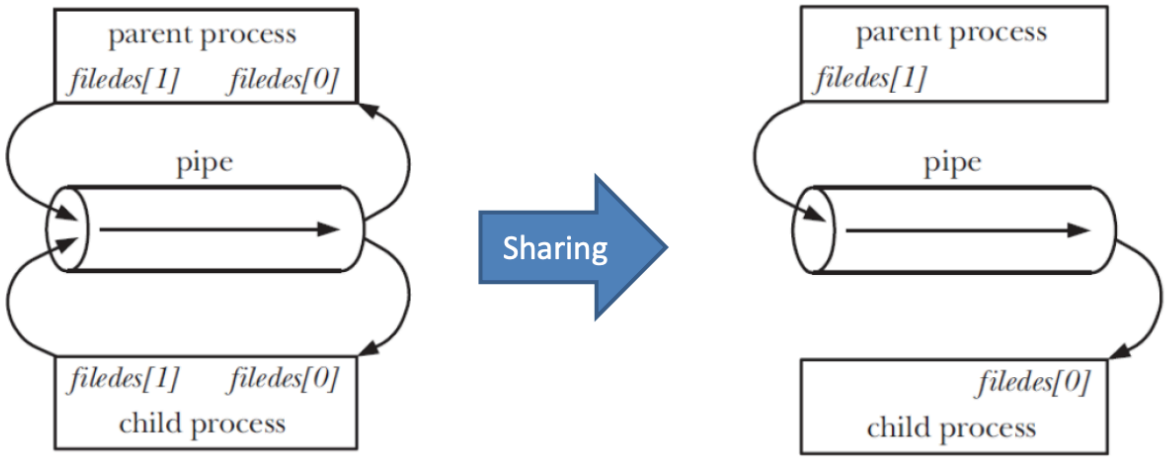
最多支持3个参数。因为在**unistd.h**里面只定义到了`_syscall3()`。超过限制的方式可以是通过修改**unistd.h**文件，定义新的`syscallx()`函数

熟悉Linux下常见的系统调用函数

大概描述代码添加过程、思路

`os_open()`

用于pipe的开启工作，原理如下图（取自李老师PPT，ch5part1;Page37）



简单来说思路是生成匿名管道（第一次close关闭没有用的管道一端），重定向STDOUT/STDIN，然后解除管道原有的端口，而PARENT一端就相对简单，只是正常写/读数据即可

`os_system`

通过`strtok`函数进行字符串分割，来分解指令字符串，在执行各命令

```
char *strtok(char *__restrict__ __s, const char *__restrict__ __delim)
Divide S into tokens separated by characters in
DELIM.
```

`main`

分需要pipe和不需要两种

需要pipe：

1. cmd1->buf：开启管道，通过read函数写进缓冲区
2. buf->cmd2：开启管道，通过write函数从缓冲区写出

不需要pipe：直接调用os_system函数即可

展示实验结果