# Lab6实验报告

## 思考题

### Thinking 6.1

简单修改样例中的代码即可。

```
case 0: /* 子进程 - 作为管道的写者 */
    close(fildes[0]); /* 关闭不用的读端 */
    write(fildes[1], "Hello world\n", 12); /* 向管道中写数据 */
    close(fildes[1]); /* 写入结束，关闭写端 */
    exit(EXIT_SUCCESS);

default: /* 父进程 - 作为管道的读者 */
    close(fildes[1]); /* 关闭不用的写端 */
    read(fildes[0], buf, 100);/* 从管道中读数据 */
    printf("father-process read:%s",buf);
    close(fildes[0]); /* 读取结束，关闭读端 */
    exit(EXIT_SUCCESS);
```

### Thinking 6.2

与fork函数类似，可能的问题还是出现在增加fd与pipe的引用次数时被中断打断而导致的没有同步增加的问题。

在进行映射之前，f[0],f[1]与pipe的引用次数为1,1,2.分别对应读，写，和管道整体的引用数目。在执行dup函数的过程中，先增加f[0]的引用次数，再增加pipe的引用次数。如果在增加f[0]的引用次数后，该进程遭遇了时钟中断而被调度，另一个进程如果调用`_pipe_is_closed`，就会发现f[0]与pipe的引用次数相同，会认为写端已经关闭，从而发生预想之外的情况。

### Thinking 6.3

在实验的操作系统下，非原子操作主要来源于时钟中断引发进程切换。而在完成系统调用的过程中，系统处于内核态，时钟中断会被屏蔽，因此系统调用不会被打断，系统调用也就保持为原子操作。

### Thinking 6.4

- 可以，只需先解除fd的映射，后解除pipe的映射，就可以满足pipe的pageref大于fd的pageref的不等式的恒成立条件，也就能保证程序运行的正确性。
- 会，与Thinking 6.2类似。

### Thinking 6.5

在处理.bss段时，会调用`load_icode_mapper()`函数。而该函数在`src==NULL`时会为其分配一页物理页面并向其中填充内容为0，故.bss段的内容在此时被自动填充为0。

## Thinking 6.6

在`user`的`init.c`中，包含如下代码：

```
if ((r = opencons()) != 0) {
        user_panic("opencons: %d", r);
    }
    // stdout
    if ((r = dup(0, 1)) < 0) {
        user_panic("dup: %d", r);
    }
```

在该段代码中，首先会调用`opencons()`打开终端，此时`fd`号为0，作为标准输入，而在`dup()`中将0号fd dup至1号fd中作为标准输出。

## Thinking 6.7

在MOS中用到的命令是外置命令。而LInux的`cd`指令是改变当前的工作目录，如果是fork一个子shell来执行命令，改变的就只是子shell的工作目录，没有完成该命令。

## Thinking 6.8

可以观察到两次spawn，分别对应`ls.b`与`cat.b`。

四次进程销毁，具体输出如下：

```
$  ls.b | cat.b > motd
[00002803] pipecreate
[00003805] destroying 00003805
[00003805] free env 00003805
i am killed ...
[00004006] destroying 00004006
[00004006] free env 00004006
i am killed ...
[00003004] destroying 00003004
[00003004] free env 00003004
i am killed ...
[00002803] destroying 00002803
[00002803] free env 00002803
i am killed ...
```

# 实验难点

## 管道的原理

整体上，管道是由两个文件描述符来进行控制的。一个对应的是读端，一个对应的是写端。而fd0与fd1的虚拟地址同样映射到一页物理内存，即为管道的具体内容所在。父子进程在关闭不需要的端口后，利用对应的文件描述符及其编号对管道进行操作。

### 管道关闭的正确判断

由于MOS采用的时间片轮转进程调度算法，在对相关的引用次数进行更改时，有可能被中途打断导致没有同步更新p[0]或p[1]与pipe的引用次数，最终造成另一进程在被调度后利用`_pipe_is_closed`时会误认为读/写端已经关闭，从而发生错误。因此，在完成相关操作时，需要灵活调整对`fd`与`pipe`的映射操作的顺序，保证两者引用次数大小关系的正确判定，从而维持对管道关闭的正确判断。

### Shell的相关函数

具体包含对spawn函数的撰写和对cmd命令的解析，虽然在有提示与在Lab1时完成printk时的经验下，代码完成相对轻易。但要真正完全读懂相关的操作，仍然需要一定的时间。

## 实验体会

作为整个Mos系统的收官之战，管道与shell的内容相比于lab4的系统调用&fork和lab5的文件系统来说并不算特别多，但是理解的难度却不比Lab5文件系统复杂的函数低。类似于后面的几个lab，本lab完成时需要熟练掌握前序的知识，例如系统调用，对文件描述符与文件的理解等，是个相当综合的单元。