Lab 6 实验报告

15061200 李明轩

一，思考题

> **Thinking 6.1** 示例代码中，父进程操作管道的写端，子进程操作管道的读端。如果现在想让父进程作为“读者”，代码应当如何修改？ ■

答：把 case 0 改成 case 1 即可

```
case 0:  /* Child - reads from pipe */
    close(fildes[1]);                       /* Write end is unused */
    read(fildes[0], buf, 100);              /* Get data from pipe */
    printf("child-process read:%s",buf);    /* Print the data */
    close(fildes[0]);                       /* Finished with pipe */
    exit(EXIT_SUCCESS);


default:  /* Parent - writes to pipe */
    close(fildes[0]);                       /* Read end is unused */
    write(fildes[1], "Hello world\n", 12);  /* Write data on pipe */
    close(fildes[1]);                       /* Child will see EOF */
    exit(EXIT_SUCCESS);
}
```

> **Thinking 6.2** 上面这种不同步修改 pp_ref 而导致的进程竞争问题在 user/fd.c 中的 dup 函数中也存在。请结合代码模仿上述情景，分析一下我们的 dup 函数中为什么会出现预想之外的情况？ ■

答：原版函数中先将新旧文件描述符映射到了同一个物理页面上，这样会造成在文件描述符映射和文件内容映射之间，如果产生了中断，就会导致无法访问新文件而出错（因为只有新的文件描述符而没有内容）。

> **Thinking 6.3** 阅读上述材料并思考：为什么系统调用一定是原子操作呢？如果你觉得不是所有的系统调用都是原子操作，请给出反例。希望能结合相关代码进行分析。 ■

答：个人认为，系统调用是否是原子操作，换句话说就是陷入内核之后还有没有更高的特权级。在我们的简化操作系统中，一旦陷入内核，中断使能就变为 0，也即不能被打断，从而实现原子操作。但是有的系统会有多级优先级（感谢郭致远大神指点），如树莓派中内置的可以支持四级特权级别，也就是说系统调用也可以再一次被中断。因此，系统调用是不是原子操作，是由不同的系统和硬件支持来决定的，需要看前提条件。

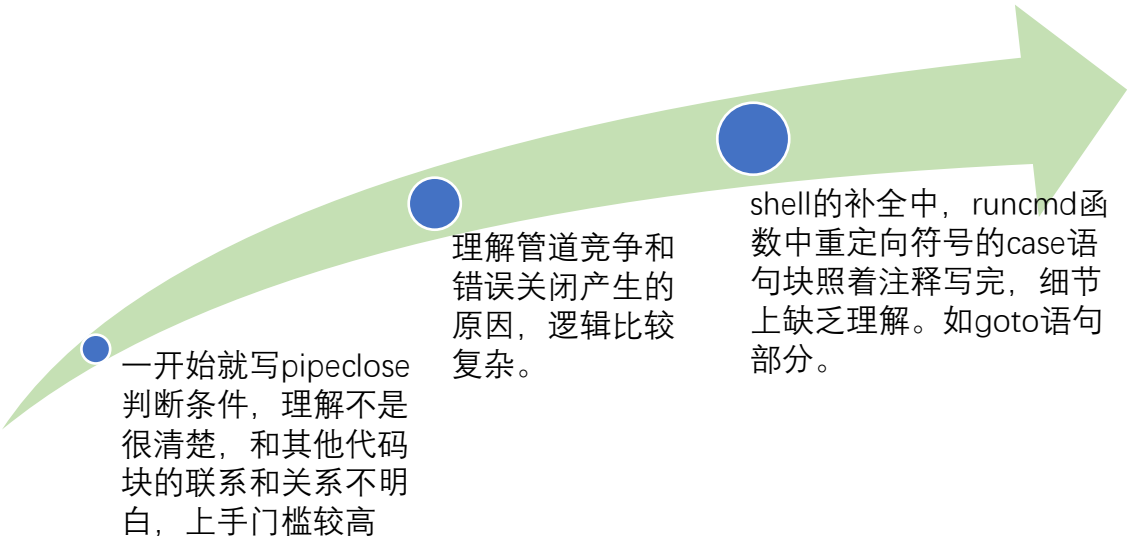
> **Thinking 6.4** 仔细阅读上面这段话，并思考下列问题
>
> - 按照上述说法控制 **pipeclose** 中 fd 和 pipe unmap 的顺序，是否可以解决上述场景的进程竞争问题？给出你的分析过程。
> - 我们只分析了 close 时的情形，那么对于 dup 中出现的情况又该如何解决？请模仿上述材料写写你的理解。
>
> ■

答：

可以，因为管道文件引用次数和 p[0]，p[1]中的某一个值相等的核心条件就是，pipe 引用次数减少速度大于 p[0]或 p[1]，这就意味着一旦父子进程中在 pipe 引用解除而 p 没有解除的时候，进程切换会导致 pipe 引用次数和 p[0]，p[1]中的某一个值相等。因此，只要让 p[0]，p[1]引用次数先于 pipe 降低即可避免切换带来的相等。

只要将 dup 函数中两块分别负责复制文件描述符和重映射文件内容的代码块换个顺序即可。

二，难点示意图



三，感想与总结

最后一次操作系统实验，算是对于前面所有课程内容的集大成者。从管道通信到简易 shell，这个制作过程也完全契合我们前面几个 lab 的顺序，从内存分配到进程调度和通信。虽然 lab6 整体理解难度没有 lab5 那么深奥，但 lab6 如果仅仅是限制于完成作业，就会丢失很多探索细节的乐趣。比如谁在调用 piperead，Dev 结构体又是什么（所有设备都有的基础方法的函数指针，如 piperead，pipewrite 等）。

唯一比较坑的是由于原版代码中数组没有初始化，导致 ls 和 cat 一直不能正常运行，

直到在 ls.c 和 fprintf.c 中添加了 bzero 之后才正常输出。

四，残留难点

为什么 shell 可以执行.b 结尾的文件，而不是一些直接运行的 C 语言程序(.out)？