**Lab6实验报告**

19373135 田旗舰

**一、实验思考题**

**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？**

交换case0和default的内容。

**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？**

dup函数将一个文件描述符对应的内容映射到另一个文件描述符，由于对文件描述符的引用次数和pipe的引用次数修改存在先后顺序，因此如果在其之间发生时钟中断，跳转到另一进程调用\_pipeisclosed函数，则可能会出现判断错误。

**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**

在实验操作系统中，系统调用陷入内核态时会屏蔽中断，因此系统调用一定是原子操作，不会被打断。而在Unix中系统调用不一定是原子操作，如write系统调用可以被打断。

**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，在 fd.c 中有一个 dup 函数，用于复制文件内容。试想，如果要复制的是一个管道，那么是否会出现与 close 类似的问题？请模仿上述材料写写你的理解。**

可以解决，由于pipe 的引用次数总比 fd 要高，所以先解除fd的映射，再解除pipe的映射，使得fd的引用次数减1先于pipe的引用次数减1，所以不会出现fd与pipe引用次数相同的误判。

会出现，所以，同理，由于pipe 的引用次数总比 fd 要高，应当先映射pipe，使pipe的引用次数加1先于fd。

**Thinking 6.5**

**bss 在 ELF 中并不占空间，但 ELF 加载进内存后，bss 段的数据占据了空间，并且初始值都是 0。请回答你设计的函数是如何实现上面这点的？**

MemSize大于FileSize的部分即为bss段，分配页面时将其初始化为0，使得bss的数据初值为0。

**Thinking 6.6**

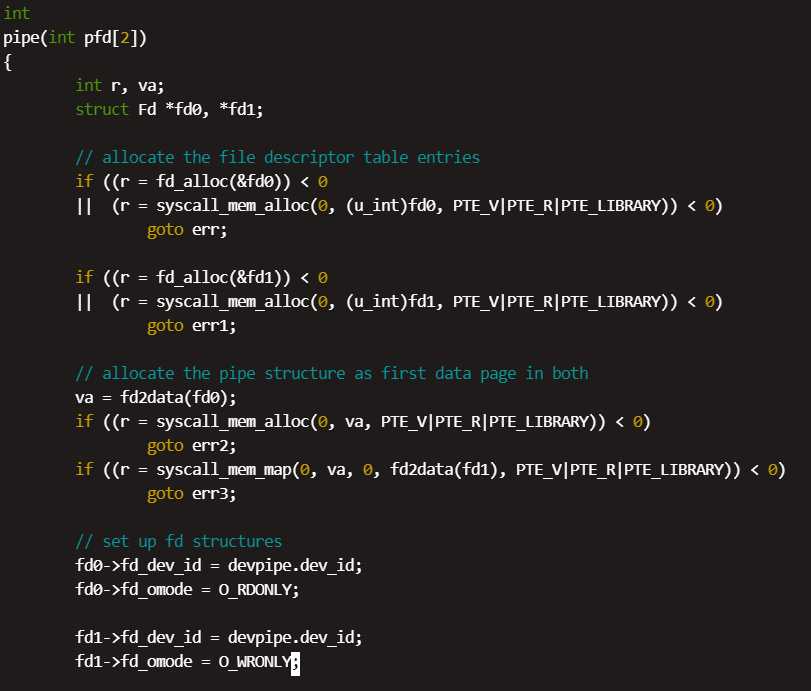
**为什么我们的 \*.b 的 text 段偏移值都是一样的，为固定值？**

链接器设置为固定值。

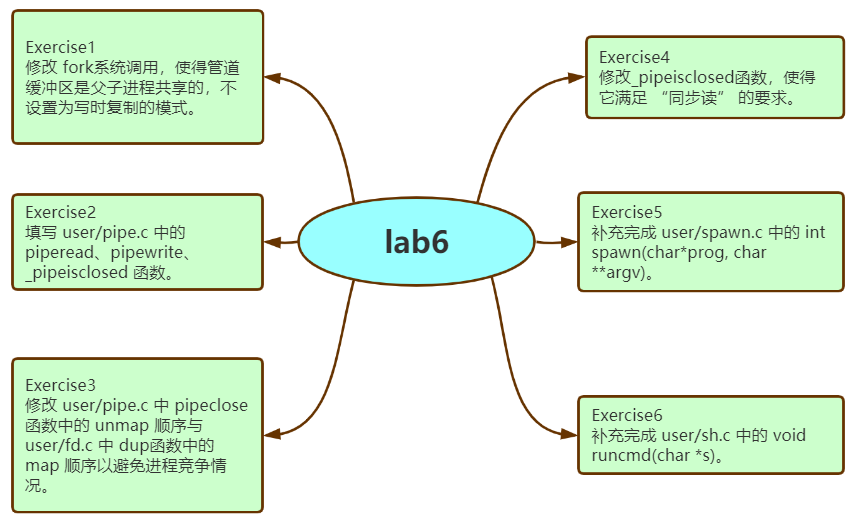
**Thinking 6.7**

**在哪步，0 和 1 被 “安排” 为标准输入和标准输出？请分析代码执行流程，给出答案。**

在创建pipe时安排的，在pipe.c的pipe函数中



**二、实验难点图示**

****

本次实验主要分为两个部分：管道和shell。其中前4个exercise与管道相关，后2个exercise与shell相关，相比较而言管道部分较为简单，只需要注意进程切换时产生的错误判断管道关闭以及同步读的的问题，而shell部分难度较高，不仅阅读的代码量较大，需要填写的部分提示也较少，需要建立在理解的基础上进行。

具体而言，本次实验的难度如下：

1.管道中进程竞争问题的处理，包括调整unmap的顺序以及实现\_pipeisclosed中同步的要求。

2.shell部分读取并加载elf文件，由于时间久远有些遗忘，需要重新熟悉elf文件格式，而加载elf文件的部分与load\_icode\_mapper函数十分相似，注意处理MemSize大于FileSize的那一部分。

**三、体会与感想**

本次实验难度尚可，大致与lab5相同，总共花费了大概2整天的时间，可能是熟能生巧，debug的时间有所减少。

随着lab6的结束，OS正式的实验部分也告一段落。这6次实验帮助我建立起了整个操作系统的概念，了解了其中部分重要机制的实现，也算是有了较大的收获。但回顾整个小操作系统，还有相当一部分的代码对我来说较为陌生。提供相应的硬件接口，从0开始编写一个操作系统对我来说还不太现实，但或许也正是操作系统的复杂性，才有了操作系统这门课程，也才需要我们更加深入地去学习操作系统。“路漫漫其修远兮,吾将上下而求索”，接下来还有挑战性任务，希望能够顺利完成。

**四、指导书反馈**

似乎指导书中关于实验正确结果的图片与实际有些不一致的地方，建议更新。

**五、残留难点**

虽然顺利通过了测评，但是会出现莫名奇妙的bug，比如刚刚退出仿真后再立即运行，有时shell无法启动，但也不是每次都能复现，不知是何种原因所致。