OS —— Lab6实验报告

22373407 王飞阳

思考题

Thinking 6.1

交换`case 0`和`default`的内容即可：

```
#include <stdlib.h>
#include <unistd.h>

int fildes[2];
char buf[100];
int status;
int main() {
    status = pipe(fildes);
    if (status == -1) {
        printf("error\n");
    }
    switch (fork()) {
        case -1:
            break;
        case 0:
            close(fildes[0]);
            write(fildes[1], "Hello world\n", 12);
            close(fildes[1]);
            exit(EXIT_SUCCESS);
        default:
            close(fildes[1]);
            read(fildes[0], buf, 100);
            printf("parent-process read:%s", buf);
            close(fildes[0]);
            exit(EXIT_SUCCESS);
    }
}
```

Thinking 6.2

dup 函数的功能是将文件描述符 oldfdnum 所对应的内容映射到文件描述符 newfdnum 中，会将 oldfdnum 和 pipe 的引用次数都增加 1，将 newfdnum 的引用次数变为 oldfdnum 的引用次数。

当我们将一个管道的读/写端对应的文件描述符（记作 fd[0]）映射到另一个文件描述符。在进行映射之前，f[0]，f[1] 与 pipe 的引用次数分别为 1, 1, 2。按照 dup 函数的执行顺序，会先将 fd[0] 引用次数 +1，再将 pipe 引用次数 +1，如果 fd[0] 的引用次数 +1 后恰好发生了一次时钟中断，进程切换后，另一进程调用 _pipeisclosed 函数判断管道写端是否关闭，此时`pageref(fd[0]) = pageref(pipe) = 2`，所以会误认为写端关闭，从而出现判断错误。

Thinking 6.3

在`kern/entry.S`中，有：

```
#include <asm/asm.h>
#include <stackframe.h>

.section .text.tlb_miss_entry
tlb_miss_entry:
    j       exc_gen_entry

.section .text.exc_gen_entry
exc_gen_entry:
    SAVE_ALL
    mfc0    t0, CP0_STATUS
    and     t0, t0, ~(STATUS_UM | STATUS_EXL | STATUS_IE)
    mtc0    t0, CP0_STATUS
    mfc0    t0, CP0_CAUSE
    andi    t0, 0x7c
    lw      t0, exception_handlers(t0)
    jr      t0
```

要使处理器处于内核模式并启用异常重入，我们取消设置UM和EXL，取消设置IE以全局禁用中断。

故在进行系统调用时，系统陷入内核，会关闭时钟中断，以保证系统调用不会被打断，因此系统调用都是原子操作。

Thinking 6.4

1. 可以解决。如果程序正常运行，pipe 的 pageref 是要大于 fd 的，在执行 unmap 操作时，优先解除 fd 的映射，这样就可保证严格大于关系恒成立，即使发生了时钟中断，也不会出现运行错误。
2. 会出现同样的问题。但同样的道理，在 dup 使引用次数增加时，先将 pipe 的引用次数增加，这样就可保证严格大于关系恒成立，保证不会出现两者相等的情况，即使发生了时钟中断，也不会出现运行错误。

Thinking 6.5

1. 通过分配新的文件描述符、使用文件系统进程间通信（fsipc）准备它，并将文件内容映射到内存来打开文件。
2. 填写了load_icode 函数，实现了ELF 可执行文件中读取数据并加载到内存空间，其中通过调用 elf_load_seg 函数来加载各个程序段。填写 load_icode_mapper 回调函数，在内核态下加载 ELF 数据到内存空间。
3. 处理 bss 段的函数是 lab3 中的 load_icode_mapper。在这个函数中，我们要对 bss 段进行内存分配，但不进行初始化。当 `bin_size < sgsize` 时，会将空位填 0，在这段过程中为 bss 段的数据全部赋上了默认值 0。

Thinking 6.6

在 `user/init.c` 文件中的 `main` 函数当中有下面的部分。

```
// stdin should be 0, because no file descriptors are open yet
    if ((r = opencons()) != 0) {
        user_panic("opencons: %d", r);
    }
// stdout
    if ((r = dup(0, 1)) < 0) {
        user_panic("dup: %d", r);
    }
```

它将0映射在1上，相当于就是把控制台的输入输出缓冲区当做管道。

Thinking 6.7

1. 在 MOS 中，我们用到的 shell 命令是外部命令，需要 fork 一个子 shell 来执行命令。
2. Linux 的 cd 指令是改变当前的工作目录，如果在子 shell 中执行，则改变的是子 shell 的工作目录，无法改变当前 shell 的工作目录。

所有能对当前 shell的环境作出改变的命令都必须是内部命令

Thinking 6.8

两次 spawn，结果如下所示。

```
$ ls.b | cat.b > motd
[00001c03] pipecreate
[00001c03] SPAWN: ls.b
serve_open 00001c03 ffff000 0x0
serve_open 00002404 ffff000 0x1
[00002404] SPAWN: cat.b
serve_open 00002404 ffff000 0x0
```

四次进程销毁，结果如下所示。

```
[00002c05] destroying 00002c05
[00002c05] free env 00002c05
i am killed ...
[00003406] destroying 00003406
[00003406] free env 00003406
i am killed ...
[00002404] destroying 00002404
[00002404] free env 00002404
i am killed ...
[00001c03] destroying 00001c03
[00001c03] free env 00001c03
i am killed ...
```

难点分析

管道

在本次实验中，第一部分主要是对管道（pipe）的实现。我们通过`intpipe(intfd[2])`函数创建管道，`fd[0]`对应读端，`fd[1]`对应写端。对两个文件描述符的操作是十分重要，是管道实现的基础。

查询管道是否关闭的函数为`static int _pipe_is_closed(struct Fd *fd, struct Pipe *p)`，其主要原理为：

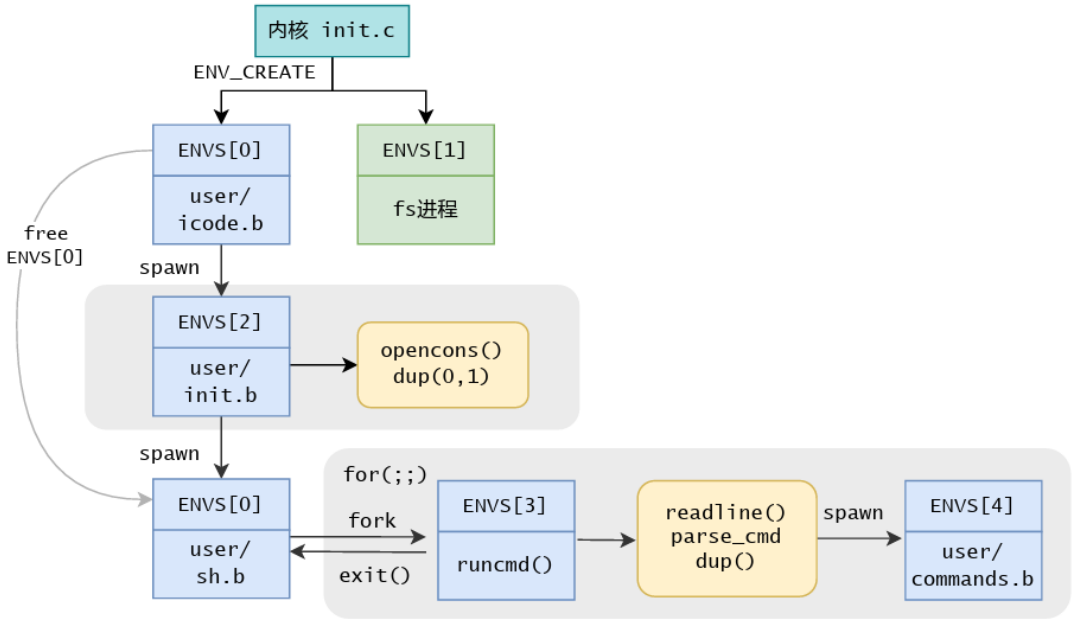
> 对于每一个匿名管道 而言，我们分配了三页空间：一页是读数据的文件描述符rfd，一页是写数据的文件描述符 wfd， 剩下一页是被两个文件描述符共享的管道数据缓冲区 pipe。既然管道数据缓冲区是被两个文件 描述符所共享的，我们很直观地就能得到一个结论：如果有1个读者，1个写者，那么管道将被 引用2次。pageref 函数能得到页的引用次数，所以有下面这个等式成立：
> `pageref(rfd) + pageref(wfd) = pageref(pipe)`

管道的读写。管道的读写的实现主要是通过维护`Pipe`结构体中`p_rpos`和`p_wpos`实现。一个管道有`PIPE_SIZE(32 Byte)`大小的缓冲区。这个`PIPE_SIZE`大小的缓冲区发挥的作用类似于环形缓冲区，所以下一个要读或写的位置`i`实际上是`i%PIPE_SIZE`。

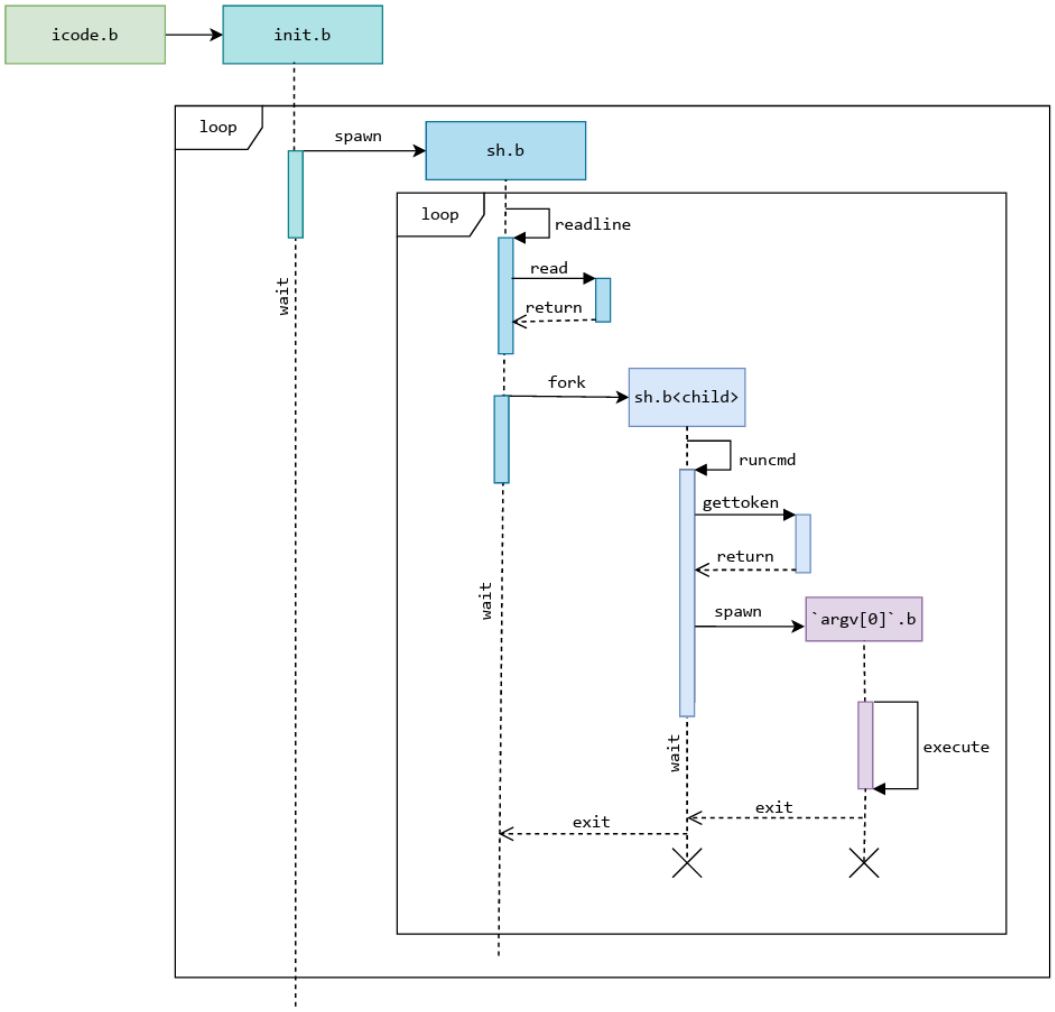
通过`p_rpos >= p_wpos`判断管道是否为空，通过`p_wpos p_rpos < PIPE_SIZE`判断管道是否满溢。

shell

shell的启动执行过程如下图所示：



其主要是通过`spawn`调用文件系统中的可执行文件并执行，`parsecmd`函数进行解析，`runcmd`函数执行解析出来的每一条命令。难点在于对shell整个过程中相关函数调用的理解：



实验体会

Lab6 的内容相对较少，但理解上的难度较高，很多函数都有难以下手的感觉，在弄懂函数原理的过程中，花费了很多的功夫。这是整个OS实验课程最后一个Lab了，OS 实验课到这里也要结束了。

在整个实验课期间，在课上和课下我都遇到了挺多困难，特别是课上实验的extra部分做得都不尽人意。但是总而言之，也算是完成了一个小型的操作系统，第一次了解到内核和计算机较底层的知识。这门课不仅丰富了我对操作系统的认识，也加深了我对C语言的理解和掌握，受益匪浅。

完结撒花！！！