Lab: Simple xv6 shell

Git链接：[Jeery1/xv6-6.S081: 2019版的xv6实验实现](https://github.com/Jeery1/xv6-6.S081/tree/main)

**实验目的：**

基础功能实现：

支持带参数的命令执行（如 xv6s hello there）

处理输入/输出重定向（如 file from READ）

实现双命令管道（如 cmd1 | cmd2）

交互规范：

使用 @ 作为提示符（区别于系统默认的 $）

不依赖内存分配（禁用 malloc），仅使用栈/全局变量

允许合理的固定限制（如命令长度、参数数量等）

**实验步骤：**

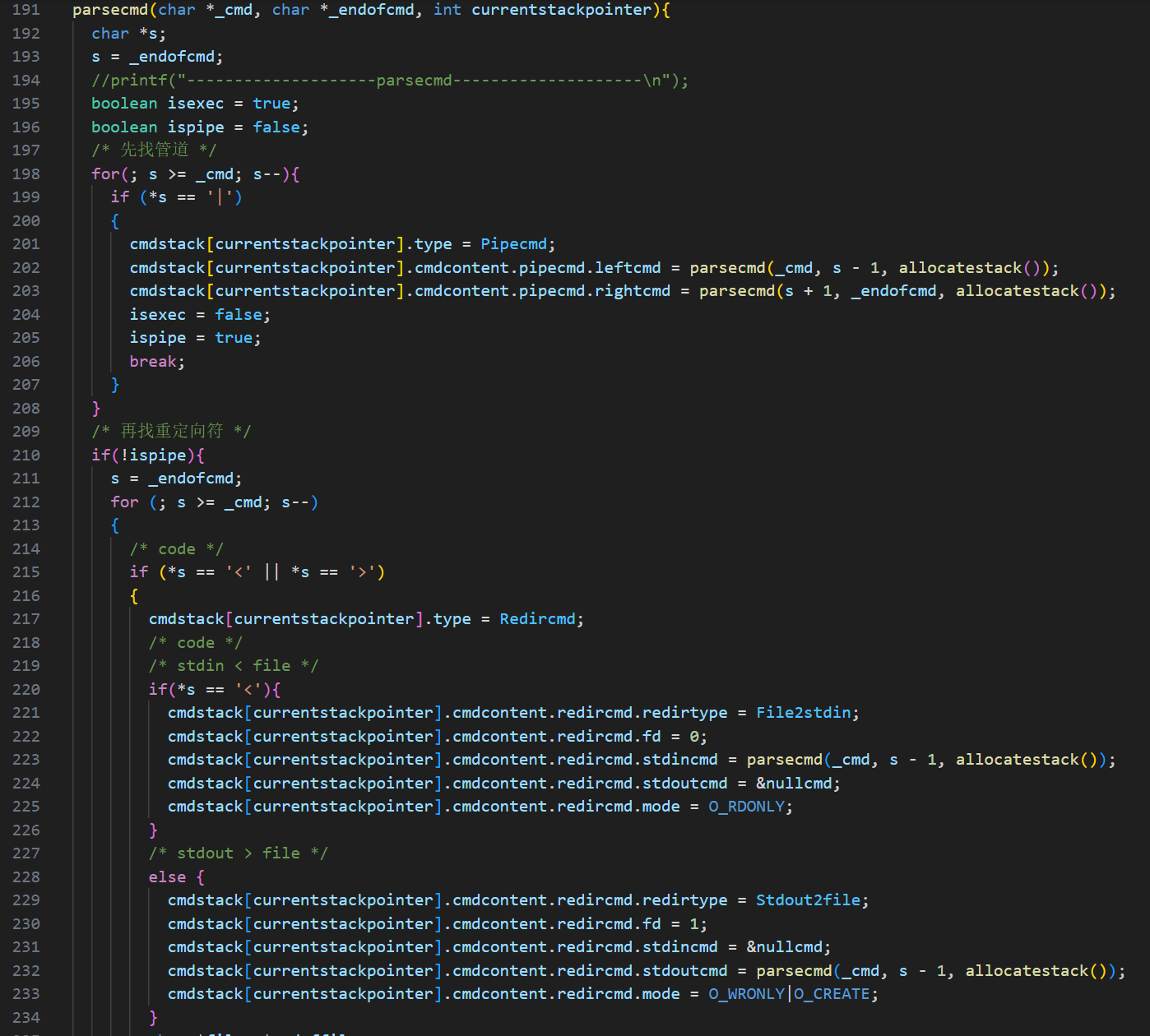
在user文件夹中新建一个nsh.c文件，编写实现nsh指令相关的代码。然后修改makefile文件，将nsh.c加入。然后执行make clean和make qemu命令，进入qemu环境，验证nsh命令的准确性。

**代码分析：**

前端解析：parsecmd()构建抽象语法树

后端执行：evaluate()递归解释执行

中间表示：通过cmd结构体连接两者





Parsecmd函数，该函数是 Shell 实现中最为核心的命令解析器

输入：

\_cmd：指向待解析字符串起始位置

\_endofcmd：指向字符串结束位置

currentstackpointer：当前命令栈位置

输出：构建好的命令节点指针

递归下降解析 + 右结合性处理

采用从右向左扫描的优先级处理：

最高优先级：管道符 |

for(; s >= \_cmd; s--) if (\*s == '|') { ... }

次高优先级：重定向符 < >

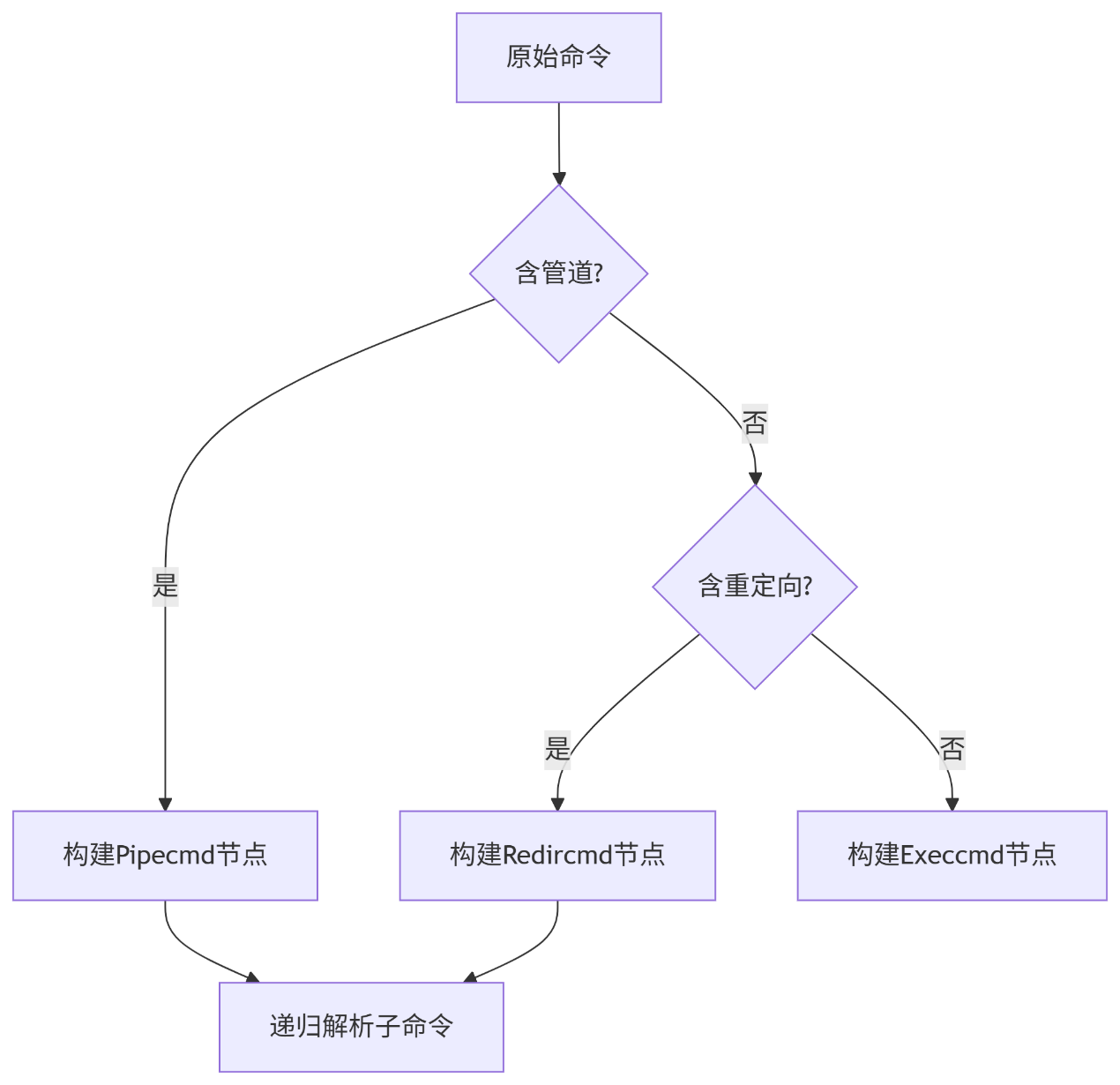
if(!ispipe) for(; s >= \_cmd; s--) if (\*s == '<' || \*s == '>') { ... }

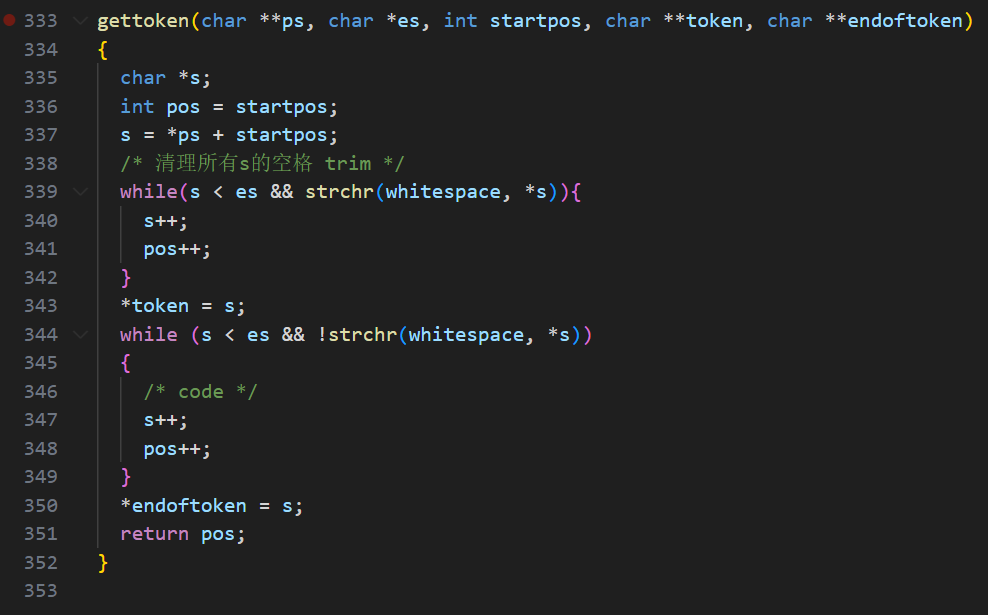
默认情况：普通命令参数

if(isexec) { */\* 处理空格分隔的参数 \*/* }

管道命令处理：将管道左右两侧分别作为子命令解析，并通过allocatestack()获取新节点空间。

普通命令处理：通过gettoken分割空格分隔的参数，使用预分配的tokens数组避免动态分配

流程图：  




gettoken函数，这是 Shell 实现中关键的词法分析函数

核心功能：从字符串中提取下一个token（非空白字符序列）

参数说明：

ps：二级指针，指向原始字符串的当前位置（会更新）

es：字符串结束位置

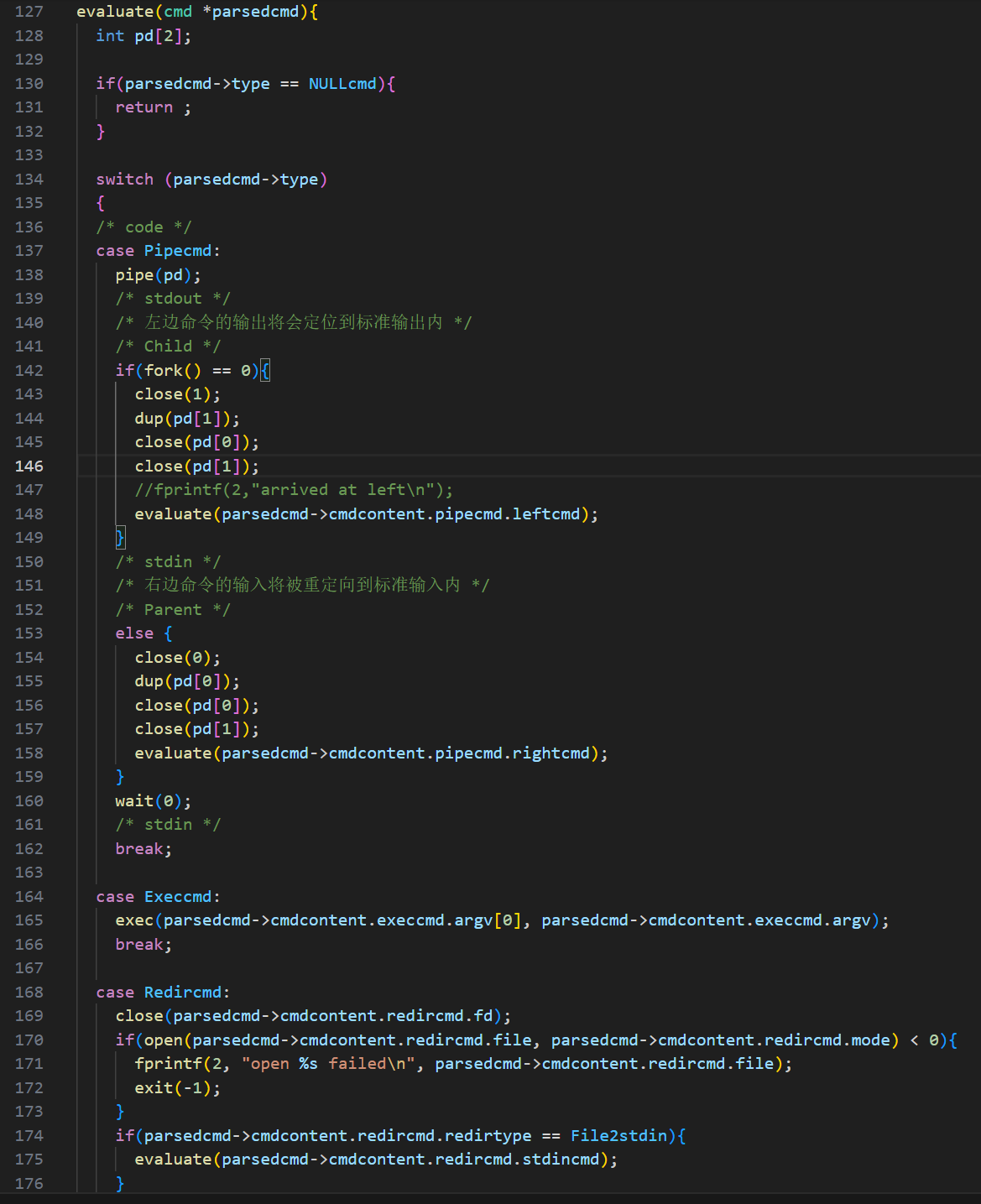
startpos：起始扫描位置

token：返回token起始地址

endoftoken：返回token结束位置

返回值：新的扫描位置（供下次调用使用）

首先跳过所有空格，将所有非空白字符位置存入\*token，遇到遇到任意空白字符或字符串结尾就终止扫描。



Evaluate函数，这是 Shell 实现中负责命令执行的核心函数

输入：解析后的命令抽象语法树（AST）

管道命令执行：case Pipecmd:

严格遵循关闭-复制-关闭模式，兄弟进程通过管道通信，wait保证管道两端顺序执行，实现同步控制。

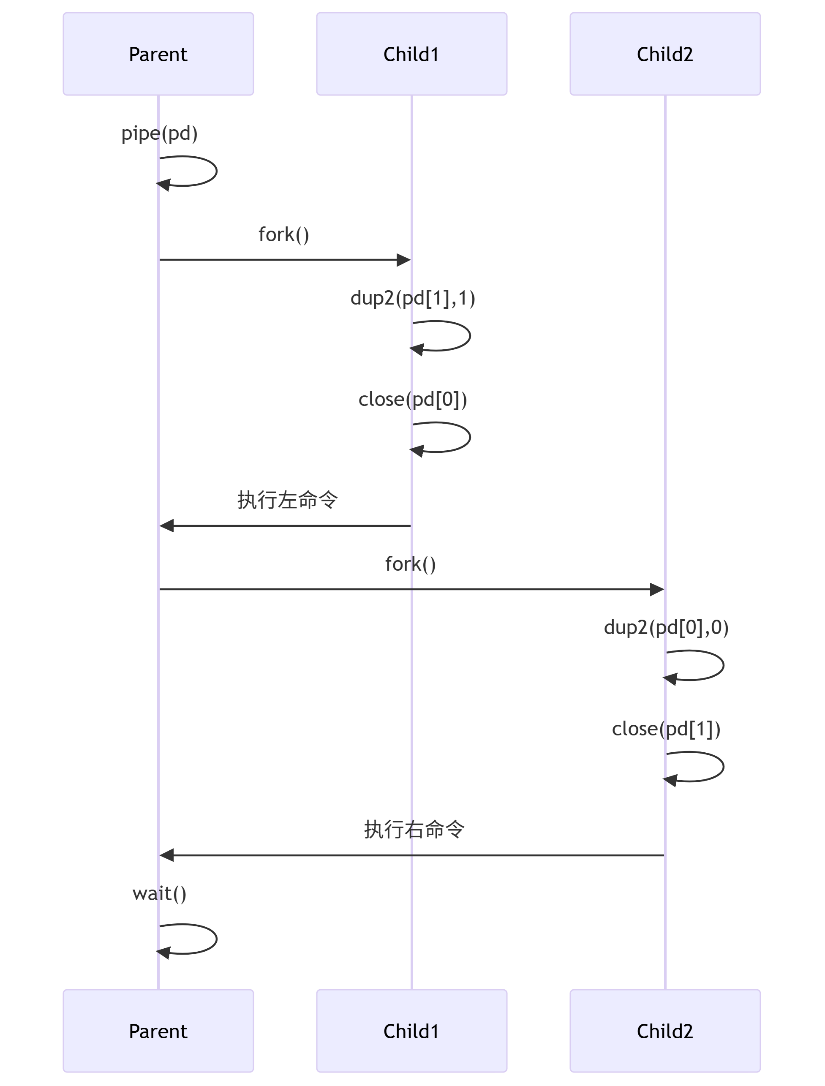
普通命令执行：case Execcmd:

执行机制：参数传递：直接使用解析好的argv数组 进程替换：当前进程被目标程序完全替换

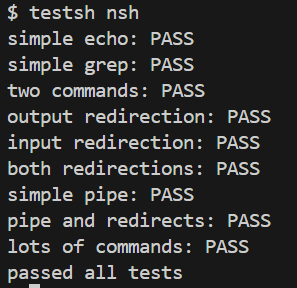
错误处理：隐含在exec系统调用中，失败时自动终止进程 由父进程通过wait获取状态（在main函数中处理）

重定向命令执行：文件描述符魔术：依赖最小可用fd分配 规则原子性操作：close+open确保重定向可靠 错误处理：文件打开失败时立即报错退出

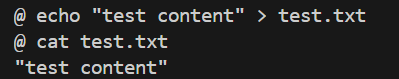
流程图：



**实验现象：**



通过了自带的测试用例



输出重定向



管道

**实验中遇到的问题和解决办法：**

问题描述：

在解析用户输入的命令时，按空格分割参数，但发现当输入类似ls -l时，argv[0]正确存储"ls"，但argv[1]却存储了"-l\n"（包含换行符），导致命令执行失败。

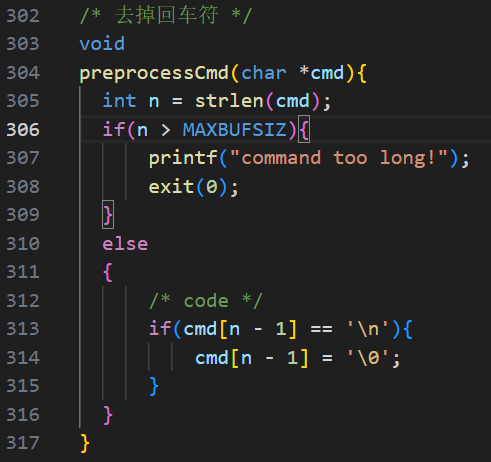
调试过程：

打印argv数组，发现argv[1]末尾带有\n。

检查输入缓冲区，发现fgets读取的输入包含换行符。

解决办法：

在分割参数前，先去除输入末尾的换行符：



然后再分割参数，确保参数正确存储。  
问题描述：

在实现cmd1 | cmd2功能时，发现程序卡死，无法正常执行。

调试过程：

检查fork()和pipe()调用，确认管道创建成功。

发现父进程没有正确关闭管道文件描述符，导致子进程read()阻塞。

原因分析：

父进程在fork()后没有关闭管道的写入端（pd[1]），导致cmd2的read()一直等待数据，而cmd1的write()可能已经结束，造成死锁。

解决办法：

在父进程中正确关闭管道两端，并在子进程中关闭不需要的文件描述符：



**实验心得：**

必须及时关闭不需要的文件描述符，否则可能导致资源泄漏或死锁。

wait()的使用很重要，避免僵尸进程。

exec失败时应打印错误信息并退出，避免子进程继续执行错误逻辑。

使用printf打印变量和文件描述符状态，有助于定位问题。

通过本次实验，深入理解了 shell 的基本工作原理，特别是管道和重定向的实现机制。