OSLab1—Shell

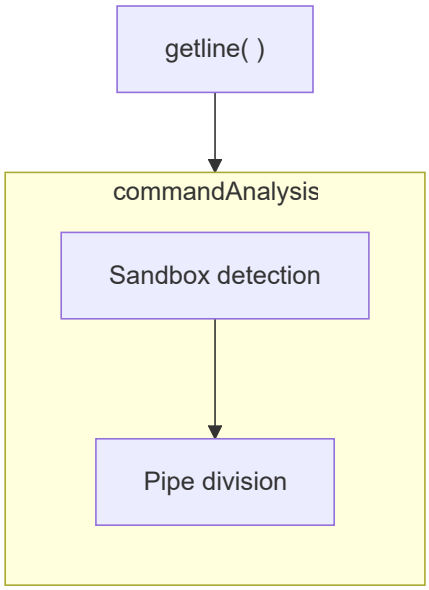
引言

本篇实验报告将简要介绍整体的代码框架，同时简述调试的一些经验。

整体框架

指令预处理

面对输入指令，需要对指令进行预处理，转换为特定的数据结构之后，传递给后续函数执行。预处理的基本流程为：



接着介绍两个部分所调用的主要函数，以及最终存储的数据结构

- Sandbox detection：通过检查输入开头是否有sandbox，决定是否执行
  - loadRules：访问指定的rule.txt，并提取deny参数，存储在数据结构RuleSet中

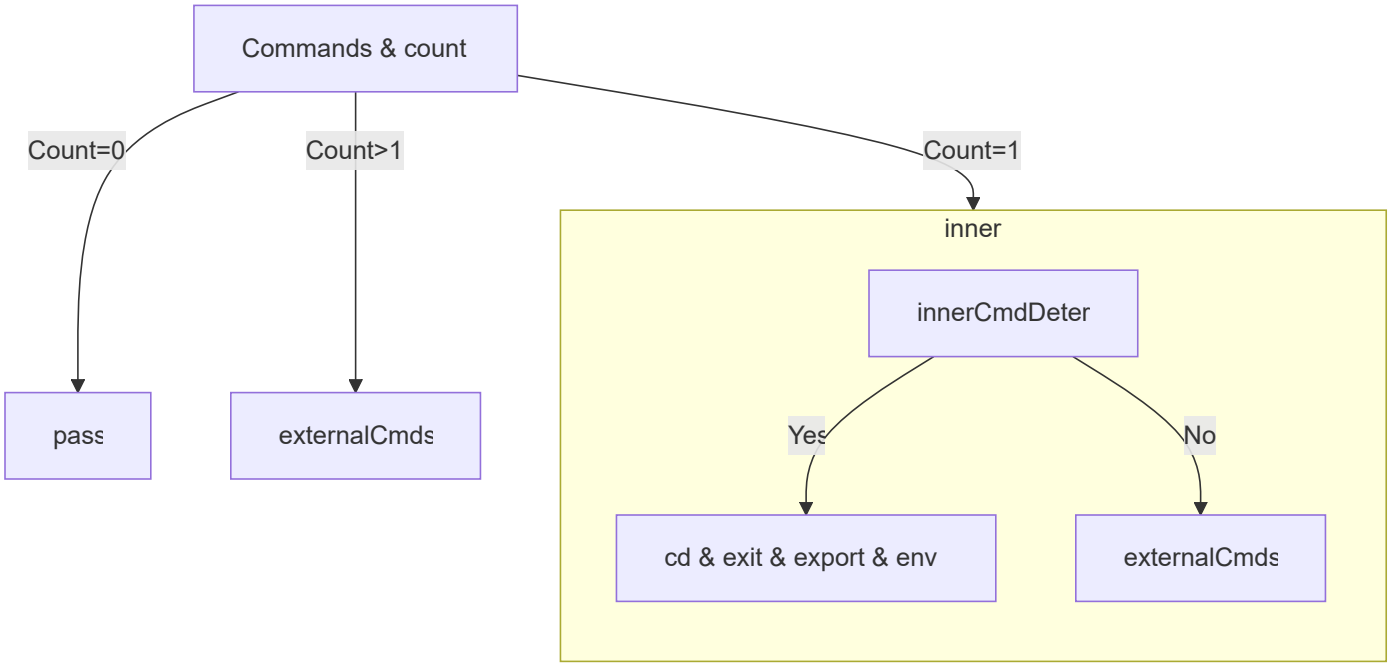
```
typedef struct {
    int arg_index;
    char arg_value_str[256];
    long arg_value_int;
    int is_string;
} SyscallArgs;

typedef struct {
    char syscall_name[32];
    int args_count;
    SyscallArgs args[6];
} SyscallRule;

typedef struct {
    SyscallRule rules[MAX_RULES];
```

```
    int rule_count;
} RuleSet;
```

- Pipe division：通过计算输入中“ | ”的数量，从而判断出一共需要执行多少指令，为commands数组分配足够的空间，存储**元指令。**
  - extractCmds：从输入Input中划分出**元指令**并保存在commands中

指令执行

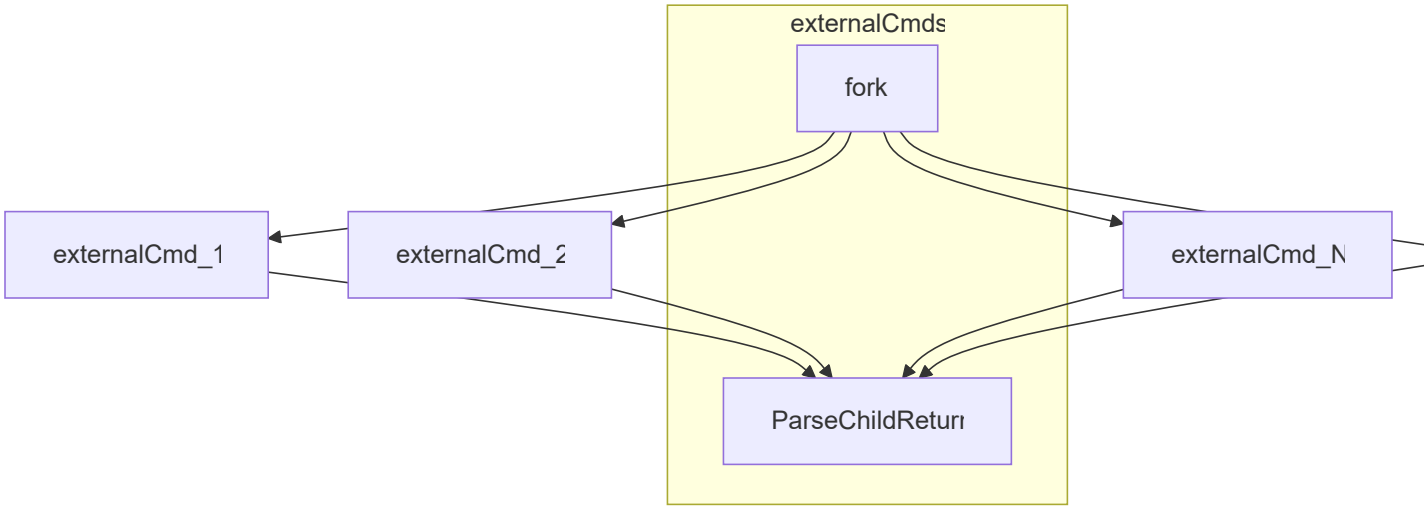
关于指令执行，可以分成几种情况：



后面将解释**为什么要将count=1和count>1分开讨论。**

externalCmds

这个函数接受commands和count，fork出多个子进程分别执行**元指令**，而自己负责监视子进程执行时的syscall，exit code，以及报错处理，具体为：



- ParseChildReturn：根据是否使用sandbox分成两种情况

    - 使用sandbox：使用ptrace监视所有子进程的syscall，一旦有子进程syscall，就会被侦测到并对黑名单进行对比，一旦匹配成功，杀死子进程。子进程结束时，还需要检测exit code，判断是否需要进行报错。
    - 不使用sandbox：不使用ptrace，当子进程结束时，检测exit code，判断是否需要进行报错。
- 将count=1和count>1分开来讨论：
    - externalCmds会fork出子进程来执行任务，而对于内置指令而言，必须要在主进程中执行，所以在逻辑上，不能够直接套用externalCmds，而是得单独判断并执行。

报错处理

为了调试方便，我在原有print系列上，又自己构建了新的报错函数errorProcess( )，所有的函数执行时，遇到错误都需要执行errorProcess，而不是直接执行print系列：

```
void errorProcess(char* pos,enum ERROR error,...)
{
    //
    //  To process all error
    //
    // printf("Sometime catch error: %s!\\n", pos);
    switch(error){
        case SYNTAXINVALID:
            print_invalid_syntax();
            break;
        case COMMANDNOTFOUND:
            print_command_not_found();
            break;
        case EXECERROR:
            print_execution_error();
            break;
        case SYSCALLBLOCK: {
            va_list args;
            va_start(args, error);
            char* syscall_name = va_arg(args, char*);
            int count = va_arg(args, int);
            struct user_regs_struct regs = va_arg(args, struct user_regs_struct);
            pid_t pid = va_arg(args, pid_t);

            dispatchSyscallArgs(syscall_name, regs, count, pid);
            va_end(args);
            break;
        }
        default:
            // Left for explore
            break;
    }
}
```

- 注意看第一行，注释掉的一行printf。整体函数执行时，遇到错误，不仅会传入错误的类型，还会传入报错的位置，而errorProcess第一行的printf可以输出报错位置，方便调试。而真正使用时，直接注释掉即可。

其他

以上我就介绍这么多，当然如你所见，整体的代码量绝不止这么多，但是其他都是一些细节。所以我这里就不一一介绍。此外，本bash的所有规格都是按照课程lab要求，所以和真实的bash还会有出入（比如我们输入“hello”，真实bash不会报错，但是我的bash会”Command Not Found“）

Todo

- 关于Pipe detection，检测机制不够健全，比如被双引号括起来的 | ，按理不应该算的，但是还是计算在内了。这里应该加入状态机检测。
- 关于sandbox，对于rule.txt文件的规则格式而言，过于严苛了（精确到空格），后面需要优化提取机制。