Shell Lab 实验报告

石依凡 2020202264

一、总体完成情况

本实验共有六个待填写部分：

- eval 函数：接受由 main 函数中初步处理的命令行字符串，并通过 builtin_cmd 和 parse_line 函数将其处理成一系列命令行参数形式（char * argv[]）。然后，根据命令内容创建进程（组）、管理进程（组）。
- builtin_cmd 函数：处理内置命令（built-in commands）。这些命令是终端 shell（*tsh*）内置命令——fg、bg、jobs、quit 等，不需要创建新进程，因此单独拿出。
- do_bgfg 函数：完成 fg、bg 这两个内置命令，实现前台（*fgjob_cmd*）和后台（*bgjob_cmd &*）任务管理、切换。
- waitfg 函数：在 shell 前台存在任务时，挂起 shell 使其不能继续接受命令行输入，并将完整屏幕交接给前台任务。在前台任务终止（terminate）或挂起（stop）后，释放进程并删除任务（如果前台任务被终止）或修改任务状态（如果前台任务被挂起）。
- sigchld_handler 函数：获取 SIGCHLD 信号，并处理。
- sigint_handler 函数：获取 SIGINT 信号，并将其发送给当时的前台任务（若有）。
- sigtstp_handler 函数：获取 SIGSTP 信号，并将其发送给当时的前台任务（若有）。

在整个 *tsh* shell 运行期间，这六个函数均起到了预期的作用。在使用终端 shell 脚本（trace 文件）检测后，实验完成度良好，可以完成预期功能：**正确终止、创建并管理前台&后台进程（组）、处理信号、并进行简单的错误（异常）输入处理**。

二、*tsh* shell 使用文档

tsh shell 运行后，屏幕将出现命令提示符 `tsh>`，此时即可输入命令。

输入命令分为两类：内置命令和创建进程命令：

- 内置命令
  - fg：本命令将任务 ID 是 *jobid* 或 PID 是 *pid* 的进程改为前台状态，并继续运行（如果该进程被挂起）。语法为 fg %*jobid* 或 fg *pid*，例如 fg %1、fg 15223。
  - bg：本命令将进程改为后台运行状态，如果进程被挂起，则对进程组发送 SIGCONT 信号使其继续运行。语法规则同“fg”命令。
  - jobs：本命令不接受其他参数，将在屏幕上显示目前仍然在运行或被挂起的由 *tsh* shell 创建的前台&后台进程。显示格式：[*jobid*] (*pid*) *status tsh_command*。其中，*status* 是该任务的状态：后台运行（Running）、前台运行（Foreground）、挂起（Stopped）；*tsh_command* 是通过 *tsh* shell 创建进程时所键入的完整命令。例如：“[1] (15223) Stopped ./myspin 5 &”。
- 创建进程命令：用户需要键入该进程二进制具有执行权限的文件的绝对路径或相对路径，相对路径为相对 *tsh* shell 二进制文件的路径。然后，在其后输入该进程需要的参数，并用分隔符隔开。最后，键入回车结束命令。

三、实现细节

（一）命令处理

程序通过已经写好的parseline函数，将命令分割为命令行参数char * argv[]，并针对 argv[0]的内容进行分支处理，将其归类为内置命令&创建进程、前台任务&后台任务。

如果是内置命令，则调用 builtin_cmd；否则，首先使用 fork 创建子进程，使用 setpgid 修改其进程组 ID，然后子进程用 execvp 将程序路径和命令行参数传入，父进程将根据前台、后台状态情况来选择挂起等待（waitfg）或继续运行。

为防止对 jobs 进行更改时，*tsh* shell 终端被某些信号中断，从而造成不可逆的影响，笔者使用了 sigprocmask 函数将无关信号暂时阻断，并在接收信号后，将屏蔽取消，继续接收其他信号。

（二）任务整理

任务整理主要是对进程前台&后台状态、前台任务运行的管理。

关于前台&后台状态管理，*tsh* shell 通过对内置命令解析，判定其命令含义，并正确将参数分类为任务 ID 和 PID，然后对任务列表中对应的进程发送 SIGCONT 信号，将可能挂起的进程恢复，并修改任务列表的任务状态。

关于前台任务运行管理，*tsh* shell 使用 WUNTRACED 参数的 waitpid 函数将 shell 进程挂起，并等待直至子进程被挂起或终止。然后，通过对 status 参数的解析，如果是正常退出（WIFEXITED），则直接删除任务；如果是被某些信号中断而终止（WIFSTOPPED），则屏幕提示任务异常被信号（WTERMSIG）中断退出，然后删除任务；如果被信号中断而挂起（WIFSTOPPED），则屏幕提示任务异常被信号（WSTOPSIG）中断挂起，然后修改任务状态为挂起（ST）。

为防止对 jobs 进行删除或更改时，*tsh* shell 终端被某些信号中断，从而造成不可逆的影响，笔者使用了 sigprocmask 函数将无关信号暂时阻断，并在接收信号后，将屏蔽取消，继续接收其他信号。

（三）信号管理

信号管理主要为对 SIGCHLD、SIGINT 和 SIGTSTP 信号的接收和处理。

对于 SIGINT 和 SIGTSTP 信号的处理，*tsh* shell 先检索是否有前台任务正在运行，如果没有则忽略信号，否则将对应的信号发送至对应的进程组（避免仅仅终止单个进程而造成错误的意图理解并产生僵尸进程）。

对于 SIGCHLD 信号，当程序捕捉到 SIGCHLD 信号位时，使用 while 循环（防止多个子进程（组）同时引发信号）和“WNOHANG | WUNTRACED”选项的 waitpid 函数将未来得及回收的所有进程（组）均回收处理。然后，根据子进程状态（终止或挂起）相应地修改或删除任务状态。

为防止对 jobs 进行删除或更改时，*tsh* shell 终端被某些信号中断，从而造成不可逆的影响，笔者使用了 sigprocmask 函数将无关信号暂时阻断，并在接收信号后，将屏蔽取消，继续接收其他信号。

四、问题与改进

本实验基本实现了一个基本功能性的 Unix Shell，并经过了测验。存在的不

足有：

1) 对当前工作路径的管理欠缺。在使用“cd”、“ls”、“pwd”等操作系统提供的命令时，无法确切定位到“cd”等命令修改的工作目录。
2) 未实现管道、输入输出重定向等实用功能。
3) 对信号的处理较为简单，只考虑到常见的几个信号处理。
4) 对错误处理不够细致，如未细化因异常无法通过 exec 更新子进程内容的原因。