Shell Lab

实验目标

通过编写一个简单的支持作业控制（job control）的 Unix shell 程序来熟悉进程控制和信号的概念。

实验步骤

1. 登陆服务器
    - 地址：ics.ayaya.in
    - 用户名与密码：与上学期一致
2. 使用 `cp ~/../shlab-handout.tar ~/` 将实验文件复制到自己的用户目录下
3. 使用 `tar -xvf shlab-handout.tar` 解压，得到以下文件：
    - `tsh.c`：需要完成的代码
    - `tshref`：参考二进制文件，你的 shell 程序应与其表现一致
    - `trace*.txt`：测试 shell 程序正确性的输入文件
    - `sdriver.pl`：测试 shell 程序正确性的脚本
    - `tshref.out`：测试文件的期望输出
    - `my*.c`：测试时被调用的小程序
4. 完成实验代码。`tsh.c` 中包含了一些已经实现好的函数，你需要在此基础之上完成下列函数：
    - `eval`：解析并解释命令行的主要部分
    - `builtin_cmd`：识别并解释内建命令（`quit`，`fg`，`bg` 以及 `jobs`）
    - `do_bgfg`：实现内建命令 `bg` 和 `fg`
    - `waitfg`：等待一个前台作业结束
    - `sigchld_handler`：捕捉 `SIGCHLD` 信号
    - `sigint_handler`：捕捉 `SIGINT`（`ctrl-c`）信号
    - `sigtstp_handler`：捕捉 `SIGTSTP`（`ctrl-z`）信号

    每一次你修改了 `tsh.c` 之后都应该使用 `make` 重新编译它，可以使用 `./tsh` 来运行你的 shell。

实验说明

什么是 Unix shell?

一个 shell 是一个交互式的命令行解释器，代表用户运行程序。一个 shell 会重复打印一个提示符，从 `stdin` 等待命令，然后根据命令内容执行一些操作。

命令行是由空格分隔的 ASCII 字符组成的序列。命令行的第一个词要么是内建命令的名字，要么是一个可执行文件的路径名。剩下的词是命令行参数。如果第一个词是内建命令，shell 会在当前进程立即执行命令；否则，shell 会 fork 一个子进程，然后在子进程的上下文中加载并运行程序。由解释一条命令行创建的子进程被称作作业（job）。一般来说，一个作业可以包含由 Unix 管道（pipes）连接的多个子进程。

如果命令行以符号「&」结束，那么作业会在后台运行，意味着 shell 在打印提示符并等待下一条命令行之前不会等待这一作业结束。否则，任务会在前台运行，意味着在等待下一条命令行之前 shell 会等待这一作业结束。因此，在任何时候，至多只有一项作业在前台运行。不过，后台可以运行任意数量的作业。

例如，输入

```
tsh> jobs
```

会让 shell 执行内建 `jobs` 命令。输入

```
tsh> /bin/ls -l -d
```

会让 `ls` 程序在前台执行。一般来讲，shell 保证在程序开始执行主函数 `int main(int argc, char *argv[])` 时，参数 `argc` 和 `argv` 的值如下：

- `argc == 3`
- `argv[0] == "/bin/ls"`
- `argv[1] == "-l"`
- `argv[2] == "-d"`

或者，输入

```
tsh> /bin/ls -l -d &
```

会让 `ls` 程序在后台运行。

Unix shell 支持作业控制（job control），可以允许用户在前台和后台之间移动作业，并改变一个作业中进程的状态（running，stopped 或者 terminated）。输入 `ctrl-c` 会给前台作业地每个进程一个 `SIGINT` 信号，`SIGINT` 信号的默认行为是终止进程。类似地，输入 `ctrl-z` 会给前台作业的每个进程一个 `SIGTSTP` 信号，`SIGTSTP` 信号的默认行为是将一个进程置入 stopped 状态，直到其接收到 `SIGCONT` 信号被唤醒。Unix shell 也提供了不同的内建命令来支持作业控制。例如：

- `jobs`：列出处于 running 和 stopped 状态的后台作业
- `bg <job>`：将一个 stopped 的后台作业状态改为 running
- `fg <job>`：将一个 stopped 或 running 状态的后台作业切换到前台并置为 running
- `kill <job>`：终止一个作业

更多关于 shell 和 terminal 的内容可以参考：

- Unix shell
- Terminal emulator

`tsh` 的特性

你的 `tsh` 应该具有以下特性：

- 提示符应该是字符串 「tsh> 」
- 用户输入的命令行由一个 `name` 和若干个参数构成，所有这些都以空格分隔。如果 `name` 是一个内建命令，`tsh` 应该立即处理它然后等待下一条命令。否则，`tsh` 应该假设 `name` 是一个可执行文件的路径，将其加载并运行在一个子进程中（在这里，作业指这个子进程）
- tsh不需要支持管道（`|`）或者I/O重定向（`<` 和 `>`）
- 输入 `ctrl-c`（`ctrl-z`）应该给前台作业发送 `SIGINT`（`SIGTSTP`）信号，并且也要发送给这个作业的所有后继（例如它 fork 出的任何子进程）。如果没有前台作业，这些信号没有任何作用
- 如果命令行以「&」结尾，`tsh` 应该将作业运行在后台，否则，应该运行在前台

- 每一个作业用一个进程 ID（PID）或一个作业 ID（JID）标识，它们都是正整数，JID 是 `tsh` 内部标识。JID 在命令行中应该用「%」作为前缀。例如，「%5」标识 JID 5，「5」标识 PID 5（已经提供了用于操作作业列表的函数）
- `tsh` 应该支持以下内建命令：
  - `quit` 命令终止 shell
  - `jobs` 命令列出所有后台作业
  - `bg <job>` 命令通过发送 `SIGCONT` 信号重启 `<job>`，之后将其在后台运行。`<job>` 可以是 PID 或 JID
  - `fg <job>` 命令通过发送 `SIGCONT` 信号重启 `<job>`，之后将其在前台运行。`<job>` 可以是 PID 或 JID
- `tsh` 需要回收所有僵尸状态（zombie）的子进程。如果任何作业因为其接收到了一个没有捕获的信号而终止，那么 `tsh` 应该识别到这一事件并使用作业的PID 和信号描述打印一条信息

我们提供了一些工具来帮助你检查你的工作：

- `tshref` 是一个该任务的参考可执行文件。如果你对你的 shell 应该有什么样的行为有任何疑问，可以执行该程序解决。你的 shell 程序应该与参考程序有相同的输出（除了 PID 这类每次运行都会改变的值）。
- `sdriver.pl` 是一个以子进程运行一个 shell 的脚本，它会按照 trace 文件的指示发送命令和信号，捕获并显示 shell 的输出。我们提供了 16 个 trace 文件（`trace{01-16}.txt`）用来测试 shell 的正确性。
  - 使用 `-h` 参数获取 `sdriver.pl` 的使用方法
  - 使用 `./sdriver.pl -t trace01.txt -s ./tsh -a "-p"` 或者 `make test01` 使用 `trace01.txt` 测试你的 shell（-a “-p“ 参数让你的shell不要打印提示符）
  - 使用 `./sdriver.pl -t trace01.txt -s ./tshref -a "-p"` 或者 `make rtest01` 使用 `trace01.txt` 运行 `tshref`

提示

- 仔细阅读课本第八章
- 使用 trace 文件指导 shell 开发。从 `trace01.txt` 开始，确保你的 shell 能产生与参考完全相同的输出之后，再开始 `trace02.txt`，等等
- `waitpid`，`kill`，`fork`，`execve`，`setpgid` 以及 `sigprocmask` 等函数将会非常有用。`waitpid` 的 `WUNTRACED` 和 `WNOHANG` 等选项也非常有用
- 当你实现你的信号处理函数时，确保将 `SIGINT` 和 `SIGTSTP` 信号发送给整个前台进程组，使用 `-pid` 而不是 `pid` 作为 `kill` 函数的参数
- 当你的 shell 卡住时，可以使用 `ctrl-d` 强行退出程序
- 一个值得考虑的地方是如何在 `waitfg` 和 `sigchld_handler` 函数之间分配任务，我们推荐以下做法：
  - 在 `waitfg` 中，使用 `sleep` 或其他功能相似的函数并在其附近忙等待
  - 在 `sigchld_handler` 中，只调用一次 `waitpid`

  当然，其他方法也是可行的
- 在 `eval` 中，父进程必须在 fork 子进程之前使用 `sigprocmask` 屏蔽 `SIGCHLD` 信号，然后在通过 `addjob` 将子进程添加到作业列表之后再一次使用 `sigprocmask` 解除屏蔽。因为子进程继承了父进程的屏蔽列表，子进程必须确保在执行新程序之前解除屏蔽 `SIGCHLD` 信号。父进程需要使用这种方法来屏蔽 `SIGCHLD` 信号，以此避免子进程在父进程调用 `addjob` 之前被 `sigchld_handler` 回收（因此被从作业列表中移出）产生的竞争情况

- `more`，`less`，`vi` 和 `emacs` 等程序会对终端设置做一些奇怪的事，不要在你的 shell 里运行这些程序。可以使用简单的基于文本的程序，例如 `/bin/ls`，`/bin/ps` 和 `/bin/echo` 等
- 当你从标准 Unix shell 中运行你的 shell 时，你的 shell 会被运行在前台进程组中。如果你的 shell 创建了一个子进程，默认情况下该子进程也会是这个前台进程组的成员。因为输入 `ctrl-c` 会对前台进程组的每一个进程都发送 `SIGINT` 信号，所以它会给你的 shell 和其创建的每一个进程都发送 `SIGINT` 信号，这显然是不正确的，可以考虑使用 `setpgid` 重新分配进程组 ID

评分标准

你需要提交**实验报告**和**代码**。

- 代码（70%）
  - 正确性（测试文件与下发文件一致）
  - 代码风格
- 报告（30%）

思维扩展

- 怎样在 shell 中实现管道和输入输出重定向功能?