Shell Lab 实验指导

实验概述

编写一个简单的支持作业控制（Job Control）的**Unix Shell**。主要涉及到基于信号的进程间通信等核心技术。

那么什么是Unix Shell? 🐚

在计算机科学中，Shell俗称“壳”，和“核”（或内核，Kernel）等组件共同组成计算机系统的框架。一个 Shell 是一个交互式的命令行解释器，能够代表用户运行程序。即，**Shell接受用户命令，并帮助用户调用相应的应用程序**。

Unix Shell中的命令分为内建命令和一般命令。对于内建命令（例如，`jobs`），Shell进程会根据这个命令的要求直接执行对应的操作（例如，打印出当前管理的作业列表）。对于一般命令，Shell进程会`fork`出一个子进程，并在子进程的上下文中`exec`加载并运行对应的程序。

Unix Shell支持的一般命令的格式一般如下：

```
可执行文件路径名 + 可选的程序选项 + 可选的“&”符号
```

程序选项会作为参数传给被调用程序的`main`函数。如果命令以“&”结束，那么作业会在后台（Background）运行，意味着Shell不会等待这一作业结束便立刻打印命令提示符（Prompt）并等待下一条命令；否则，任务会在前台（Foreground）运行，Shell会等待作业结束再打印命令提示符和等待下一条命令。因此，在任意时候，至多只有一项作业在前台运行，但是后台可以同时有多个作业在运行。

例如，输入下面的命令（`tsh>`是Shell产生的命令提示符）：

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

会让Shell`fork`出一个子进程`ls`，并将参数传给它的主函数`int main(int argc, char * argv[])`，即：

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

并且，这个作业会在后台运行，Shell会立即打印命令提示符 `tsh>` 并等待下一条命令。

同时，Unix Shell支持作业控制（Job Control），允许用户使用下面的命令在前台和后台间移动作业，或者改变一个运行中作业的状态（Running、Stopped、Terminated）：

- `fg`：将一个Stopped或者Running状态的后台作业切换至前台，并转化为Running状态
- `bg`：将一个Stopped状态的后台作业状态改为Running
- `kill`：向一个作业发送信号，可以改变其状态
- 键盘输入 `ctrl-c`：给前台作业的每一个进程发送 `SIGINT` 信号
- 键盘输入 `ctrl-z`：给前台作业的每一个进程发送 `SIGSTP` 信号

更多关于Unix Shell的内容可以参考：

- Unix shell
- Terminal emulator
- 计算机壳层

你的Shell：`tsh`

你设计的 `tsh` 应当具有以下特性：

- 命令提示符应当是字符串“tsh>”。
- 用户输入的命令行由一个可执行文件路径名（或者内建命令名）和若干个参数构成，所有这些都应当以空格分隔。作业可以由“&”指定运行在后台。
- 可选支持管道（Pipes）或者I/O重定向（Redirections）。
- 键盘输入 `ctrl-c` 或者 `ctrl-z` 时，应当给当前的前台作业及其所有后继（如它 `fork` 出的所有子进程）发送信号。如果没有前台进程，这些信号将没有任何作用。
- 每一个作业对应一个 `tsh` 内部的作业识别号（Job ID） `JID`，它们都是正整数。`JID` 用于命令行中时应当使用“%”作为前缀，否则 `tsh` 将认为这个数字是进程的 `PID`。
- 需要支持下面这些内建命令：
  - `quit`：终止 `tsh`
  - `jobs`：列出所有后台作业
  - `bg <job>` 和 `fg <job>`：改变 `<job>` 的状态。 `<job>` 可以由 `JID` 或者 `PID` 进行标识。

- 需要回收所用僵尸（Zombie）状态的子进程。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 `tsh` 应该产生什么样的行为并不确定，可以执行该程序解决。你的 `tsh` 应当与参考程序有基本一致的输出。
- `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（01～16）用来测试 shell 的正确性。
  - 使用 `-h` 参数获取使用方法
  - 使用 `./sdriver.pl -t trace01.txt -s ./tsh -a "-p"` 或者 `make test01` 来使用 `trace01.txt` 测试你的Shell（ `-a "-p"` 参数让你的Shell不要打印命令提示符）
  - 使用 `./sdriver.pl -t trace01.txt -s ./tshref -a "-p"` 或者 `make rtest01.txt` 运行参考样例

实验步骤

1. 登陆服务器（ics.ruc.rvalue.moe），用户名与密码与 `ICS I` 一致。如果无法登陆请联系助教。
2. 使用 `cp ~/../shlab-handout.tar ~` 将实验handout复制到自己的用户目录下。
3. 在用户文件夹中，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应当与其表现一致
   - `trace*.txt`：测试输入
   - `sdriver.pl`：测试脚本
   - `tshref.out`：测试文件的期望输出
   - `my*.c`：测试时将被调用的小程序的源代码
4. 完成实验代码。 `tsh.c` 中包含了一些已经实现好的函数，你需要在此基础上完成：
   - `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
   - `do_bgfg`：实现内建命令 `bg` 和 `fg`
   - `waitfg`：等待一个前台作业的结束

    - `sig*_handler`：信号处理
5. 使用 `make` 编译。
6. 使用 `./tsh` 来手动运行你的Shell进行测试，或使用提供的脚本运行测试。

非常温馨的提示

- 请仔细听课，并认真完成老师布置的其他作业。否则，这个实验可能会让你一筹莫展。
- 可以参考下发的英文指导 `shlab.pdf`。
- 操作系统提供的 `waitpid`、`kill`、`fork`、`execve`、`setpgid` 以及 `sigprocmask` 等库函数可能会非常有用。
- 请仔细阅读 `kill` 和 `waitpid` 函数的使用方法。
- 当你的Shell卡住时，可以使用 `ctrl-d` 强制退出程序。
- 请充分考虑各种可能出现的情况，并妥善使用 `sigprocmask` 屏蔽信号。
- `more`、`less`、`vi`、`emacs` 等程序会对终端设置做一些奇怪的事情，最好不要在你的 `tsh` 中运行。建议使用的测试程序包括：`/bin/ls`、`/bin/ps`、`/bin/echo`。
- 因为你的 `tsh` 不是一个真的Shell，只是一个被Unix Shell调用的程序，因此当你运行你的Shell的时候，它会被运行在前台进程组中。如果你的Shell创建了一个子进程，那么默认情况下这个子进程也会是这个进程组里的成员。因此，默认情况下，当你使用 `ctrl-c` 等方式时，Unix Shell会给你的Shell和这个前台进程组的所有子进程发送信号，这显然是不正确的。你应当考虑如何解决这个问题。

非常重要的提示

- 代码禁止抄袭或者提供给别人抄袭！我们会对本学期所有的实验和作业进行查重。一旦发现存在极其严重的互相抄袭现象或者抄袭互联网上代码的情况，将会严重地影响到你此次实验的成绩，甚至会严重地影响到你的【所有实验和作业的总平时成绩】。
- 你需要保护好自己的作业。OBE平台和服务器请及时修改密码。同时禁止将自己的文件夹设置为所有人可见！本学期课程中，因未保管好作业而造成的一切责任由个人承担。
- 如果部分参考了互联网上的代码，请最好在实验报告中指出。

评分标准

你需要在OBE平台上提交**实验报告和代码**。请打包为一个zip或tar文件，内含PDF格式报告和 `tsh.c`。

- 代码（60%）
    - 正确性
    - 代码风格
- 实验报告（40%）

思维拓展

你可以思考：如何在Shell中实现管道和I/O重定向功能?