计算机系统基础 2018 春季学期

Shell Lab: Writing Your Own Unix Shell

1、介绍

本实验的目的是使你熟悉进程控制和信号处理的概念。你需要实现一个简单的支持作业控制的 Unix shell 程序。

2、下载作业

你的实验所需材料包含在名为 shlab-handout.tar 的 Linux 压缩文件中。登录 Linux 机器然后操作以下命令：

- linux> tar xvf shlab-handout.tar 解压缩该 tar 文件。
- linux> make 编译和链接一些测试例程。
- 在 tsh.c 文件的最开始的注释中写上你的学号，例如

```
/* $begin tshref-ans */
/*
 * tsh - A tiny shell program with job control
 *
 * 2014202110021
 */
```

在 tsh.c 文件中可以看到一个简单的 Unix shell 的功能框架。为了帮助你开始工作，我们已经实现了一些简单的功能。你的任务是完成下面列出的这些空函数。为了帮助你验证，我们还列出了参考实现中每个函数大概的行数（其中包括一定的注释）。

- eval：主函数，分析和解释命令行［70 行］
- builtin_cmd：识别和解释内置命令：quit，fg，bg 和 jobs［25 行］
- do_bgfg：实现内置命令 bg 和 fg［50 行］
- waitfg：等待一个前台作业结束［20 行］

- sigchld_handler：捕获 SIGCHILD 信号［80 行］
- sigint_handler：捕获 SIGINT（ctrl-c）信号［15 行］
- sigtstp_handler：捕获 SIGTSTP（ctrl-z）信号［15 行］

每次修改了 tsh.c 以后，都要用 make 重新编译它。要运行你的 shell，就在命令行上输入：

```
linux>  ./tsh
tsh> [在此输入你的 shell 命令]
```

3、Unix shell 概述

shell 是一个交互式命令行解释器，代表用户运行程序。shell 会不断输出提示符，等待 stdin 上的命令行输入，然后按照命令行内容的指示执行某些操作。

命令行是由空格分隔的一系列 ASCII 文本组成。命令行中的第一个单词是内置命令的名称或可执行文件的路径名，其余的单词是命令行参数。如果第一个单词是内置命令，那么 shell 会立即在当前进程中执行该命令。否则，这个词被认为是可执行程序的路径名。在这种情况下，shell 会 fork 一个子进程，然后在子进程中加载和运行该程序。为解释一条命令行而创建的子进程统称为作业，通常，一个作业可以由多个子进程组成，子进程之间通过 Unix pipe（管道）连接。

如果命令行以＆符号结尾，则作业将在后台运行，这意味着在打印提示符或者输入下一个命令行之前，shell 不用等待上一个作业终止。否则，作业在前台运行，这意味着 shell 会等待上一个作业终止，然后才能等待下一个命令行。因此，在任何时候，最多只有一项工作可以在前台运行。但是，后台可以运行任意数量的作业。

比如，敲下面命令：

```
tsh> jobs
```

shell 会执行内置 job 命令。输入

```
tsh> /bin/ls -l –d
```

在前台运行 ls 程序。通常 shell 会确保在程序开始执行主程序

```
int main（int argc，char * argv []）
```

时，argc 和 argv 参数具有以下值：

- argc == 3,

- argv[0] == ‘‘/bin/ls’’
- argv[1]== ‘‘-l’’ ,
- argv[2]== ‘‘-d’’ .

或者，输入命令行

tsh> / bin / ls -l –d &

将会在后台运行 ls 程序。

Unix shell 支持*作业控制（job control）*的概念，它允许用户在后台和前台之间来回切换作业，并改变作业中进程的进程状态（运行，暂停或终止）。键入 ctrl-c 会向前台作业中的每个进程发送 SIGINT 信号。SIGINT 的默认操作是终止进程。类似地，键入 ctrl-z 会向前台作业中的每个进程发送 SIGTSTP 信号。SIGTSTP 的默认操作是将进程置于暂停状态，直到它接收到 SIGCONT 信号被唤醒。Unix shell 还提供支持作业控制的各种内置命令。比如：

- job：列出正在运行和已暂停的后台作业。
- bg <job>：将暂停的后台作业更改为运行状态的后台作业。
- fg <job>：将暂停或正在运行的后台作业更改为在前台运行。
- kill <job>：终止工作。

4、tsh 规范

你的 tsh shell 应该具有以下功能：

- 提示符应该是字符串 “tsh>” 。
- 用户键入的命令行应包含一个 name 和零个或多个参数，所有参数均由一个或多个空格分隔。如果 name 是一个内置命令，那么 tsh 应该立即处理它并等待下一个命令行。否则，tsh 应该假设 name 是可执行文件的路径，它在一个新的子进程的上下文中加载并运行（在此上下文中，术语 job 指的就是这个新的子进程）。
- tsh 不需要支持管道（|）或 I/O 重定向（<和>）。
- 键入 ctrl-c（ctrl-z）应该使 SIGINT（SIGTSTP）信号发送到当前前台作业以及该作业的任何后代（例如，它产生的任何子进程）。如果没有前台作业运行，那么

信号不会产生影响。

- 如果命令行以＆符号结束，则 tsh 应在后台运行该作业。否则应该在前台运行该作业。
- 每个作业都可以通过进程 ID（PID）或作业 ID（JID）来标识，JID 是由 tsh 分配的正整数。 应在命令行上用前缀'％'表示 JID。例如， “％5” 表示 JID 5， “5” 表示 PID 5（我们为你提供了管理作业列表所需的所有例程。）
- tsh 应支持以下内置命令：
    - quit 终止 shell。
    - jobs 列出所有后台作业。
    - bg <job> 通过发送 SIGCONT 信号重新启动<job>，然后在后台运行它。<job>参数可以是 PID 或 JID。
    - fg <job> 通过发送一个 SIGCONT 信号重新启动<job>，然后在前台运行它。<job>参数可以是 PID 或 JID。
- tsh 应回收所有的僵尸子进程。如果任何作业因收到未捕获的信号而终止，则 tsh 应识别此事件并打印带有该作业的 PID 和有问题信号的描述信息。

5、检查工作

我们提供了一些工具来帮助你检查你的工作。

参考解决方案。Linux 可执行文件 tshref 是 shell 的参考解决方案。 运行这个程序来解决有关你的 shell 应该如何运行的任何问题。你的 shell 应该得到与参考解决方案相同的输出（当然，PID 每次运行都会发生改变）。

Shell 驱动。 sdriver.pl 程序将 shell 作为子进程执行，根据 trace 文件的指示向其发送命令和信号，捕获并显示 shell 的输出。

使用-h 参数查看 sdriver.pl 的用法：

```
unix> ./sdriver.pl -h
Usage: sdriver.pl [-hv] -t <trace> -s <shellprog> -a <args>
Options:
-h Print this message
```

-v Be more verbose

-t <trace> Trace file

-s <shell> Shell program to test

-a <args> Shell arguments

-g Generate output for autograder

我们还提供了 16 个 trace 文件（trace{01-16}.txt）来测试你的 shell 程序的正确性。较低编号的跟踪文件运行非常简单的测试，较高编号的执行相对复杂的测试。

你可以使用例如跟踪文件 trace01.txt 在你的 shell 上运行 shell 驱动程序，方法是键入以下命令：

unix> ./sdriver.pl -t trace01.txt -s ./tsh -a "-p"

（-a “-p” 参数是让 shell 不要产生提示），或

unix> make test01

类似地，为了将结果与参考 shell 进行比较，可以通过输入以下命令来在参考 shell 上运行 trace 驱动程序：

unix> ./sdriver.pl -t trace01.txt -s ./tshref -a "-p"　　或

unix> make rtest01

tshref.out 给出了参考实现 tshref 在所有 trace 上的输出，供你参考。这比自己手动在所有 trace 文件上运行 shell 驱动程序要方便一些。

trace 文件的好处在于，如果你以交互方式运行 shell，你将获得相同的输出（除了标识该 trace 文件的初始注释）。比如：

bass> make test15

./sdriver.pl -t trace15.txt -s ./tsh -a "-p"

#

trace15.txt - Putting it all together

#

tsh> ./bogus

./bogus: Command not found.

tsh> ./myspin 10

Job (9721) terminated by signal 2

tsh> ./myspin 3 &

[1] (9723) ./myspin 3 &

tsh> ./myspin 4 &

[2] (9725) ./myspin 4 &

tsh> jobs

[1] (9723) Running ./myspin 3 &

[2] (9725) Running ./myspin 4 &

tsh> fg %1

Job [1] (9723) stopped by signal 20

tsh> jobs

[1] (9723) Stopped ./myspin 3 &

[2] (9725) Running ./myspin 4 &

tsh> bg %3

%3: No such job

tsh> bg %1

[1] (9723) ./myspin 3 &

tsh> jobs

[1] (9723) Running ./myspin 3 &

[2] (9725) Running ./myspin 4 &

tsh> fg %1

tsh> quit

bass>

6、提示

- 阅读教材第 8 章（异常控制流）的每一个字。
- 用 trace 文件来引导你的 shell 的开发。从 trace01.txt 开始，确保你的 shell 产生与参考 shell 相同的输出。然后继续执行 trace 文件 trace02.txt，依此类推。

- waitpid，kill，fork，execve，setpgid 和 sigprocmask 函数会很有用。waitpid 函数的 WUNTRACED 和 WNOHANG 选项也是有用的。
- 当实现信号处理程序时，请确保将 SIGINT 和 SIGTSTP 信号发送到整个前台进程组，在 kill 函数的参数中使用 “-pid” 而不是 “pid” 。sdriver.pl 程序会检测这个错误。
- 作业中棘手的部分之一是决定 waitfg 和 sigchld_handler 函数之间的工作分配。我们建议采用以下方法：
    - 在 waitfg 中，写一个包含 sleep 函数的忙循环。
    - 在 sigchld_handler 中，只需调用 waitpid 一次。

  虽然其他解决方案也是可能的，例如在 waitfg 和 sigchld_handler 中都调用 waitpid，但这可能会非常复杂。在处理程序中完成所有回收更简单。
- 在 eval 中，在 fork 子进程前，父进程必须使用 sigprocmask 阻塞 SIGCHLD 信号，然后通过调用 addjob 将子节点添加到作业列表后，再次使用 sigprocmask 解除阻塞这些信号。由于子进程继承了他们父进程的 blokced 向量，子进程在执行新程序之前，必须确保解除阻塞 SIGCHLD 信号。

  父进程需要以这种方式阻塞 SIGCHLD 信号，以避免在父节点调用 addjob 之前由 sigchld_handler 回收子进程的竞争（并因此从作业列表中删除）。
- 诸如 more，less，vi 和 emacs 这样的程序会在终端设置中做出奇怪的事情。不要从你的 shell 运行这些程序。只使用简单的基于文本的程序，如 /bin/ls，/bin/ps 和/bin/echo。
- 当你从标准 Unix shell 运行你的 shell 时，你的 shell 正在前台进程组中运行。 如果你的 shell 创建一个子进程，默认情况下，这个子进程也是前台

进程组的成员。由于键入 ctrl-c 会向前台组中的每个进程发送 SIGINT，因此键入 ctrl-c 将向你的 shell 以及 shell 创建的每个子进程发送 SIGINT，这显然是不正确的。

解决方法如下：在 fork 之后，execve 之前，子进程应调用 setpgid（0，0），该进程将子进程放入一个新的进程组中，该进程组的组 ID 等于子进程的 PID。这可以确保在前台进程组中只有一个进程（即你的 shell）。 当你键入 ctrl-c 时，shell 捕获得到的 SIGINT，然后将其转发给适当的前台作业（或者更确切地说，包含前台作业的进程组）。

7、评估

根据以下分布计算您的分数，最多不超过 90 分：

80 分正确性：有 16 个跟踪文件，每个 5 分。

20 分格式。 我们希望你的程序有好的备注（10 分），并检查每个系统调用的返回值（10 分）。

你的 shell 程序将使用实验目录中包含的相同 shell 驱动程序和 trance 文件在 Linux 机器上进行正确性测试，你的 shell 程序应该与参考 shell 运行 trace 文件产生相同的输出，只有两个例外：

- PID 可以不同。
- trace11.txt，trace12.txt 和 trace13.txt 中的/bin/ps 命令每次运行输出将会不同。但是/bin/ps 命令输出中任何 mysplit 进程的运行状态应该是相同的。

8、提交

- 一定要在 tsh.c 中写入你的学号
- 将 tsh.c 重命名为 tsh-学号.c
- 将你的 tsh.c 中的如下七个函数单独拷贝至一个文件 tsh-short-学号.c

中：

- void eval(char *cmdline);
- int builtin_cmd(char **argv);
- void do_bgfg(char **argv);
- void waitfg(pid_t pid);
- void sigchld_handler(int sig);
- void sigtstp_handler(int sig);
- void sigint_handler(int sig);

- 将 tsh-学号.c 和 tsh-short-学号.c 发送至邮箱：icscswhu@163.com，截止时间：2018 年 6 月 30 日。