CS 213,2002 年秋
季实验室作业 L5:编写您自己的 Unix Shell 分配:10 月 24 日,截止时间:10 月 31 日星期四晚上 11:59

Harry Bovik (bovik@cs.cmu.edu) 是这项任务的负责人。

介绍

本作业的目的是更加熟悉过程控制和信号发送的概念。您将通过编写一个支持作业控制的简单 Unix shell 程序来完成此任务。

后勤

您可以在最多两人的小组中解决此作业中的问题。唯一的“上交”将是电子的。对作业的任何澄清和修改都将发布在课程网站上

页。

分发说明

> 特定于站点:在此处插入一段内容,解释讲师如何将 shlab-handout.tar 文件分发给学生。以下是我们在卡耐基梅隆大学使用的指示。

首先将文件 shlab-handout.tar 复制到您计划在其中进行工作的受保护目录（实验室目录）。然后执行以下操作:

- 键入命令tar xvf shlab-handout.tar 以展开tar 文件。
- 键入命令make 来编译和链接一些测试例程。
- 在tsh.c 顶部的标题注释中输入您的团队成员姓名和Andrew ID。

查看 tsh.c （微型 shell)文件,您将看到它包含简单 Unix shell 的功能框架。为了帮助您入门,我们已经实现了一些不太有趣的功能。你的任务

就是完成下面列出的剩余空函数。为了对您进行完整性检查,我们在参考解决方案（其中包括大量注释)中列出了每个函数的大致代码行数。

- eval:解析和解释命令行的主例程。 [70行]
- 内置cmd:识别并解释内置命令:quit、fg、bg 和jobs。 [25]
  线]
- do bgfg:实现bg 和fg 内置命令。 [50行]
- waitfg:等待前台作业完成。 [20行]
- sigchld 处理程序:捕获SIGCHILD 信号。 80行]
- sigint 处理程序:捕获SIGINT (ctrl-c) 信号。 [15行]
- sigtstp 处理程序:捕获SIGTSTP (ctrl-z) 信号。 [15行]

每次修改 tsh.c 文件时,请键入 make 重新编译它。要运行 shell,请在命令行中输入 tsh:

```
unix> ./tsh tsh> [在
此处向 shell 键入命令]
```

Unix Shell 概述

shell是一种交互式命令行解释器,它代表用户运行程序。 shell 会重复打印提示符,等待stdin 上的命令行,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 ASCII 文本单词序列。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,则 shell 立即在当前进程中执行该命令。否则,该单词被假定为可执行程序的路径名。在这种情况下,shell 会分叉一个子进程,然后在子进程的上下文中加载并运行该程序。由于解释单个命令行而创建的子进程统称为作业。一般来说,一个作业可以由多个通过 Unix 管道连接的子进程组成。

如果命令行以“&”结尾,则作业在后台运行,这意味着 shell 不会等待作业终止,而是打印提示符并等待下一个命令行。否则,作业将在前台运行,这意味着 shell 会等待作业终止,然后再等待下一个命令行。因此,在任何时间点,最多有一个作业可以在前台运行。但是,可以在后台运行任意数量的作业。

例如,输入命令行

```
tsh> 工作
```

使 shell 执行内置的 jobs 命令。键入命令行

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

在前台运行 ls 程序。按照惯例，shell 确保当程序开始执行其主例程时

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

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 &
```

在后台运行 ls 程序。

Unix shell 支持作业控制的概念，它允许用户在后台和前台之间来回移动作业，并更改作业中进程的进程状态（运行、停止或终止）。键入 ctrl-c 会将 SIGINT 信号传递到前台作业中的每个进程。 SIGINT 的默认操作是终止进程。同样，键入 ctrl-z 会导致 SIGTSTP 信号传递到前台作业中的每个进程。 SIGTSTP 的默认操作是将进程置于停止状态，该状态将一直保持到收到 SIGCONT 信号而被唤醒为止。 Unix shell 还提供各种支持作业控制的内置命令。例如：

- 作业：列出正在运行和已停止的后台作业。
- bg <job>：将已停止的后台作业更改为正在运行的后台作业。
- fg <job>：将已停止或正在运行的后台作业更改为在前台运行。
- Kill <job>：终止作业。

tsh规范_

您的 tsh shell 应具有以下功能：

- 提示符应该是字符串“tsh> ”。

- 用户键入的命令行应包含名称和零个或多个参数,所有参数均由一个或多个空格分隔。如果 name 是内置命令,那么 tsh 应立即处理它并等待下一个命令行。否则,tsh 应假定 name 是可执行文件的路径,它在初始子进程的上下文中加载并运行该文件（在此上下文中,术语“作业”指的是该初始子进程）。

- tsh 不需要支持管道(|) 或I/O 重定向（< 和>）。

- 键入ctrl-c (ctrl-z) 应导致将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,它派生的任何子进程）。如果没有前台作业,那么信号应该没有效果。

- 如果命令行以& 符号结尾,则tsh 应在后台运行作业。
  否则,它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。 JID 应在命令行上用前缀“%”表示。例如，“%5”表示 JID 5，“5”表示 PID 5。（我们已为您提供了操作作业列表所需的所有例程。）

- tsh 应支持以下内置命令：
  - quit 命令终止 shell。
  - jobs 命令列出所有后台作业。
  - bg <job> 命令通过向 <job> 发送 SIGCONT 信号来重新启动它,然后在
    的背景。 <job> 参数可以是 PID 或 JID。
  - fg <job> 命令通过向其发送 SIGCONT 信号来重新启动 <job>,然后在
    前景。 <job> 参数可以是 PID 或 JID。

- tsh 应该收割它的所有僵尸子进程。如果任何作业因为收到未捕获的信号而终止,则 tsh 应识别此事件并打印一条消息,其中包含作业的 PID 和违规信号的描述。

检查你的工作

我们提供了一些工具来帮助您检查您的工作。

参考溶液。 Linux 可执行文件 tshref 是 shell 的参考解决方案。运行此程序可以解决有关 shell 应如何运行的任何问题。您的 shell 应该发出与参考解决方案相同的输出（当然,PID 除外,它在运行过程中会发生变化）。

外壳驱动程序。 sdriver.pl 程序将 shell 作为子进程执行,按照跟踪文件的指示向其发送命令和信号,并捕获并显示 shell 的输出。

使用 -h 参数来找出 sdriver.pl 的用法：

```
unix> ./sdriver.pl -h 用法:sdriver.pl [-hv] -t
<trace> -s <shellprog> -a <args> 选项：-h

                        打印此消息
    -在                 更详细一些
    -t <跟踪>           跟踪文件
    -s <外壳>           待测试的shell程序
    -a <参数>           外壳参数
    -G                  为自动评分器生成输出
```

我们还提供了 16 个跟踪文件 (trace{01-16}.txt),您可以将它们与 shell 驱动程序结合使用来测试 shell 的正确性。编号较低的跟踪文件执行非常简单的测试,编号较高的测试执行更复杂的测试。

您可以使用跟踪文件trace01.txt（例如)通过键入以下内容在 shell 上运行 shell 驱动程序：

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

（-a “-p”参数告诉您的 shell 不要发出提示）,或者

```
unix> 进行测试01
```

同样,要将结果与参考 shell 进行比较,您可以通过键入以下内容在参考 shell 上运行跟踪驱动程序：

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

或者

```
unix> make rtest01
```

tshref.out 给出了所有种族的参考解的输出,供您参考。这对您来说可能比在所有跟踪文件上手动运行 shell 驱动程序更方便。

跟踪文件的巧妙之处在于,它们生成的输出与您以交互方式运行 shell 时获得的输出相同（标识跟踪的初始注释除外）。例如：

```
bass> make test15 ./sdriver.pl -t
 trace15.txt -s ./tsh -a   -p   # # trace15.txt - 将其全部放在一起 # tsh> ./bogus ./bogus: 未找

到命令。 tsh> ./myspin 10 作业 (9721) 由信号 2 终止 tsh> ./myspin 3 & [1] (9723) ./

myspin 3 & tsh> ./myspin
 4 &
```

```
 [2] (9725) ./myspin 4 & tsh> 作业 [1]
(9723) 正在运
 行 [2] (9725) 正在运行 tsh> fg          ./myspin 3 & ./
 %1 作业 [1] (9723) 由信号 20            myspin 4 &
停止 tsh> 作业
[1] (9723) 已停止 [2] (9725) 正在运行 tsh> bg %3 %3:没有这
样的作业 tsh>
 bg %1 [1] (9723) ./myspin 3             ./myspin 3 & ./
 & tsh> 作业 [1] (9723) 正在运           myspin 4 &
行 [2 ] (9725) 运
行 tsh> fg %1 tsh> 退出
bass>


                                         ./myspin 3 & ./
                                         myspin 4 &
```

提示

- 阅读课本中第8 章（异常控制流)的每一个字。
- 使用跟踪文件来指导 shell 的开发。从trace01.txt 开始,确保您的shell 生成与参考shell相同的输出。然后继续跟踪文件trace02.txt,依此类推。
- waitpid、kill、fork、execve、setpgid 和 sigprocmask 函数将非常有用。
  便利。 waitpid 的 WUNTRACED 和 WNOHANG 选项也很有用。
- 当您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,并在kill 函数的参数中使用“-pid”而不是“pid”。
  sdriver.pl 程序测试此错误。
- 分配的棘手部分之一是决定 waitfg 之间的工作分配
  和 sigchld 处理函数。我们建议采用以下方法：
  - –在 waitfg 中,在 sleep 函数周围使用繁忙循环。
  - –在 sigchld 处理程序中,仅使用一次对 waitpid 的调用。

  虽然其他解决方案也是可能的,例如在 waitfg 和 sigchld 处理程序中调用 waitpid,但这些解决方案可能会非常混乱。在处理程序中完成所有收割会更简单。
- 在eval 中,父进程必须在分叉子进程之前使用sigprocmask 阻止SIGCHLD 信号,然后解除阻止这些信号,在通过调用addjob 将子进程添加到作业列表后再次使用sigprocmask。由于子进程继承了其父进程的阻塞向量,因此子进程必须确保在执行新程序之前解除 SIGCHLD 信号的阻塞。

父级需要以这种方式阻止 SIGCHLD 信号,以避免在父级调用 addjob之前子级被 sigchld 处理程序获取（从而从作业列表中删除)的竞争条件。 –

- more、less、vi 和emacs 等程序对终端设置执行奇怪的操作。不要从 shell 运行这些程序。坚持使用简单的基于文本的程序,例如 /bin/ls、/bin/ps 和 /bin/echo。

- 当您从标准Unix shell 运行shell 时,您的shell 将在前台进程组中运行。如果您的 shell 随后创建了一个子进程,则默认情况下该子进程也将成为前台进程组的成员。由于输入 ctrl-c 会将 SIGINT 发送到前台组中的每个进程,因此输入 ctrl-c 会将 SIGINT 发送到您的 shell 以及您的 shell 创建的每个进程,这显然是不正确的。

解决方法如下:在 fork 之后但在 execve 之前,子进程应该调用 setpgid(0, 0),这会将子进程放入一个新的进程组中,其组 ID 与子进程的 PID 相同。这可以确保前台进程组中只有一个进程,即您的 shell。当您键入 ctrl-c 时,shell 应该捕获生成的 SIGINT,然后将其转发到适当的前台作业（或更准确地说,包含前台作业的进程组）。

评估

您的分数将根据以下分布从最多 90 分中计算出来:

80正确性:16 个跟踪文件,每个文件有 5 个点。

10 个风格点。我们希望您能提出好的意见（5分)并检查每个项目的返回值
系统调用（5分）。

将使用实验室目录中包含的相同 shell 驱动程序和跟踪文件在 Linux 计算机上测试您的解决方案 shell 的正确性。您的 shell 应该在这些跟踪上生成与参考 shell相同的输出,只有两个例外:

- PID 可以（并且将会)不同。

- trace11.txt、trace12.txt 和trace13.txt 中/bin/ps 命令的输出每次运行都会有所不同。但是，/bin/ps 命令输出中任何mysplit 进程的运行状态应该相同。

上交指示

特定于站点:在此处插入一段解释学生应如何提交 tsh.c 文件。以下是我们在卡耐基梅隆大学使用的指示。

- 确保您已在tsh.c 的标题注释中包含您的姓名和Andrew ID。
- 创建以下形式的团队名称：
  - “ID”,其中 ID 是您的 Andrew ID（如果您独自工作）,或者
  - “ID1+ID2”,其中 ID1是第一个团队成员的 Andrew ID,ID2是 Andrew ID 第二个团队成员的。

  我们需要您以这种方式创建您的团队名称,以便我们可以自动评分您的作业。
- 要提交 tsh.c 文件,请输入：

  make handin TEAM=团队名称

  其中 teamname 是上述团队名称。
- 提交后,如果您发现错误并希望提交修改后的副本,请键入

  make handin TEAM=团队名称 VERSION=2

  每次提交时不断增加版本号。
- 您应该通过查看来验证您的提交

  /afs/cs.cmu.edu/academic/class/15213-f01/L5/handin

  您在此目录中具有列出和插入权限,但没有读取或写入权限。

祝你好运！