CS213，2002年秋季
实验作业L5。编写你自己的Unix Shell
指派。10月24日，到期：10月31日，星期四，11:59PM

Harry Bovik（bovik@cs.cmu.edu）是这项任务的牵头人。

简介

本作业的目的是让你更熟悉过程控制和信号的概念。你将通过编写一个支持作业控制的简单的Unix shell程序来实现这一目的。

物流

你可以在一个最多两个人的小组中工作，解决这个作业的问题。唯一的"上交"将是电子的。对作业的任何澄清和修改都将在课程网页上公布。

发放指示

站点特定：在此插入一段话，解释教师将如何向学生分发`shlab-handout.tar`文件。以下是我们在CMU使用的指示。

首先，将`shlab-handout.tar`文件复制到你计划进行工作的受保护目录（*实验室目录*）中。然后进行以下操作。

- 输入`tar xvf shlab-handout.tar`命令来展开这个tar文件。
- 键入命令`make`来编译和链接一些测试例程。
- 在`tsh.c`顶部的标题注释中键入你的团队成员名字和安德鲁ID。

看一下`tsh.c`（*tiny shell*）文件，你会发现它包含一个简单的Unix
shell的功能骨架。为了帮助你入门，我们已经实现了一些不太有趣的功能。你的任务

是完成下面列出的其余空函数。作为对你的理智检查，我们在参考方案中列出了这些函数的大致代码行数（其中包括大量的注释）。

- 评价。解析和解释命令行的主要程序。[70行]
- 内置的`cmd`。识别并解释内置命令：退出、`fg`、`bg`和`jobs`。[25行]
- `do_bgfg`: 实现`bg`和`fg`的内置命令。[50行]
- `waitfg`：等待前台工作的完成。[20行]
- `sigchld_handler`:抓取SIGCHILD信号。80行]。
- `sigint_handler`。捕捉SIGINT（`ctrl-c`）信号。[15行]
- `sigtstp`处理程序。捕捉SIGTSTP（`ctrl-z`）信号。[15行]

每次你修改`tsh.c`文件时，都要输入`make`来重新编译它。要运行你的shell，在命令行中输入`tsh`。

```
unix> ./tsh
tsh> [在这里输入命令到你的shell]
```

Unix外壳的一般概述

*shell*是一个交互式的命令行解释器，代表用户运行程序。shell重复打印一个提示符，等待`stdin`上的*命令行*，然后根据命令行的内容执行一些动作。

命令行是一串以空格为界的ASCII文本字。命令行的第一个字是一个内置命令的名称或一个可执行文件的路径名。其余的字是命令行参数。如果第一个词是一个内置的命令，shell会立即在当前进程中执行该命令。否则，该词被认为是一个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和运行该程序。由于解释一个命令行而创建的子进程被统称为一个*作业*。一般来说，一个作业可以由多个由Unix管道连接的子进程组成。

如果命令行以逗号"&"结束，那么作业就在*后台*运行，这意味着shell在打印提示符和等待下一个命令行之前不会等待作业的终止。否则，作业就在*前台运行*，这意味着shell在等待下一个命令行之前会等待作业的终止。因此，在任何时间点上，最多只有一个作业在前台运行。然而，任意数量的作业可以在后台运行。

例如，输入命令行

```
tsh> 工作
```

导致 shell 执行内置的 jobs 命令。输入命令行

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

在前台运行ls程序。根据惯例，shell确保当程序开始执行其主程序时

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

argc和argv参数的值如下。

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

或者，在命令行中输入

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

在后台运行ls程序。

Unix
shells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，并改变一个作业中的进程状态（运行、停止或终止）。输入ctrl-
c会导致一个SIGINT信号传递给前台作业中的每个进程。SIGINT的默认动作是终止该进程。同样，输入ctrl-
z会向前台工作中的每个进程发送一个SIGTSTP信号。SIGTSTP的默认动作是将进程置于停止状态，直到它收到SIGCONT信号而被唤醒。Unix
shells还提供了各种支持作业控制的内置命令。例如。

- 工作。列出正在运行和停止的后台工作。
- bg <job>:将一个停止的后台作业改为运行的后台作业。
- fg <job>:将一个停止的或正在运行的后台作业改为在前台运行。
- kill <job>:终止一项工作。

tsh规格

你的tsh shell应该有以下功能。

- 提示应该是字符串 "tsh>"。

- 用户输入的命令行应该由一个名称和零个或多个参数组成，所有参数都由一个或多个空格隔开。如果name是一个内置的命令，那么tsh应该立即处理它，并等待下一个命令行的到来。否则，tsh应假定name是一个可执行文件的路径，并在一个初始子进程的上下文中加载和运行（在这里，术语*job*指的是这个初始子进程）。
- tsh不需要支持管道（|）或I/O重定向（<和>）。
- 输入ctrl-c(ctrl-z)会导致一个SIGINT(SIGTSTP)信号被发送到当前的前台工作，以及该工作的任何后代（例如，它所分叉的任何子进程）。如果没有前台工作，那么该信号应该没有任何影响。
- 如果命令行以逗号&结尾，那么tsh应该在后台运行该作业。否则，它应该在前台运行该作业。
- 每个作业可以通过进程ID（PID）或作业ID（JID）来识别，JID是由tsh分配的一个正整数。JID 应在命令行中用前缀"%"表示。例如，"%5 "表示JID 5，而 "5 "表示PID 5。(我们已经为你提供了操作工作列表所需的所有程序）。
- tsh应该支持以下内置命令。
  - quit命令可以终止shell。
  - jobs命令列出了所有的后台作业。
  - bg <job>命令通过向<job>发送SIGCONT信号来重新启动它，然后在后台运行它。<job>参数可以是一个PID或一个JID。
  - fg <job>命令通过向<job>发送一个SIGCONT信号来重新启动<job>，然后在前台运行它。<job>参数可以是一个PID或一个JID。
- tsh应该收割它所有的僵尸子程序。如果任何作业因为收到一个它没有捕捉到的信号而终止，那么tsh应该识别出这个事件，并打印出一条包含该作业的PID和违规信号描述的消息。

检查你的工作

我们提供了一些工具来帮助你检查你的工作。

参考方案。Linux的可执行程序tshref是shell的参考方案。运行这个程序来解决你对你的shell应该如何表现的任何问题。*你的shell应该发出与参考方案相同的输出*（当然，除了PID，它在运行中会改变）。

Shell驱动。sdriver.pl程序将shell作为一个子进程执行，按照*跟踪文件的*指示向其发送命令和信号，并捕捉和显示shell的输出。

使用-h参数来了解sdriver.pl的使用情况。

```
unix> ./sdriver.pl -h
用法：sdriver.pl [-hv] -t <trace> -s <shellprog> -a <args> 选项。
  -              h打印此信息
  -              v可以更详细地说明情况
  -t <           trace>Trace文件
  -s <           shell>测试的hell程序
  -a <           args>外壳参数
  -              g为自动交易器生成输出结果
```

我们还提供了16个跟踪文件（trace{01-16}.txt），你将与shell驱动一起使用，以测试你的shell是否正确。编号较低的跟踪文件做非常简单的测试，而编号较高的测试则做比较复杂的测试。

你可以使用跟踪文件trace01.txt（例如）在你的shell上运行shell驱动，通过键入。

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

（-a "-p "参数告诉你的shell不要发出提示），或者unix>

```
make test01
```

同样，为了将你的结果与参考shell进行比较，你可以在参考shell上运行跟踪驱动，输入以下内容。

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

或

```
unix> make rtest01
```

为了供你参考，tshref.out给出了所有比赛中参考方案的输出。这对你来说可能比在所有跟踪文件上手动运行shell驱动更方便。

追踪文件的好处是，它们产生的输出与你以交互方式运行shell时的输出相同（除了一个识别追踪的初始注释）。比如说。

```
贝斯> 制作test15
./sdriver.pl -t trace15.txt -s ./tsh -a "-p"
#
# trace15.txt - 把它放在一起 #
tsh> ./bogus
./bogus。tsh> ./myspin 10，未找
到命令。
工作（9721）因信号2而终止 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
工作 [1] (9723) 因信号 20 而停止 tsh>
工作
[1] (9723) Stopped    ./myspin 3 &
[2] (9725) Running    ./myspin 4 &
tsh> bg %3
%3: 没有这样的工作
tsh> bg %1
[1] (9723) ./myspin 3
& tsh> jobs
[1] (9723) 跑步         ./myspin 3 &
[2] (9725) 跑步         ./myspin 4 &
tsh> fg %1
tsh> quit
bass>
```

提示

- 阅读课本中第8章（异常控制流）的每一个字。
- 使用这些跟踪文件来指导你的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。然后转到跟踪文件trace02.txt，以此类推。
- waitpid、kill、fork、execve、setpgid和sigprocmask等函数将非常有用。waitpid的WUNTRACED和WNOHANG选项也会很有用。
- 当你实现你的信号处理程序时，要确保向前台进程组发送SIGINT和SIGTSTP信号，在kill函数的参数中使用"-pid "而不是"pid"。sdriver.pl程序测试这个错误。
- 任务的一个棘手的部分是决定各服务员之间的工作分配。
  和sigchld处理函数。我们建议采用以下方法。
  - 在waitfg中，围绕sleep函数使用一个繁忙的循环。
  - 在sigchld处理程序中，正好使用一个对waitpid的调用。

  虽然其他的解决方案也是可行的，比如在waitfg和sigchld处理程序 -
  中都调用waitpid，但这些都会让人很困惑。在处理程序中做所有的收割工作是比较简单的。
- 在评估中，父程序必须在分叉子程序之前使用sigprocmask来阻止SIGCHLD信号，然后在通过调用addjob将子程序添加到工作列表中之后再次使用sigprocmask来解除对这些信号的阻止。由于子代继承了父代的阻塞向量，子代在执行新程序前必须确保解除对SIGCHLD信号的阻塞。

父代需要以这种方式阻断SIGCHLD信号，以避免出现竞赛情况，即子代*在*父代调用addjob *之前*被sigchld处理程序收割-(从而从作业列表中删除）。

- 诸如more、less、vi和emacs等程序会对终端设置做一些奇怪的事情。不要从你的外壳中运行这些程序。坚持使用简单的基于文本的程序，如/bin/ls。
  /bin/ps，以及/bin/echo。
- 当你从标准Unix
  shell中运行你的shell时，你的shell是在前台进程组中运行的。如果你的shell随后创建了一个子进程，默认情况下这个子进程也是前台进程组的成员。由于输入ctrl-
  c会向前台进程组中的每个进程发送SIGINT，所以输入ctrl-
  c会向你的shell以及你的shell创建的每个进程发送SIGINT，这显然不正确。

  这里有一个解决方法。在fork之后，但在execve之前，子进程应该调用setpgid(0,
  0)，将子进程放入一个新的进程组，其组ID与子进程的PID相同。这样可以确保在前台进程组中只有一个进程，即你的shell。当你输入ctrl-
  c时，shell应该捕捉到所产生的SIGINT，然后将其转发给适当的前台工作（或者更准确地说，包含前台工作的进程组）。

评价

你的分数将从最高90分中根据以下分布计算出来。

80 正确性：16个跟踪文件，每个5分。

10个风格点。我们希望你有好的评论（5分），并检查每一个系统调用的返回值（5分）。

你的解决方案shell将在一台Linux机器上进行正确性测试，使用与你的实验室目录中包含的相同的shell驱动和跟踪文件。你的shell应该在这些跟踪文件中产生与参考shell**相同的**输出，只有两个例外。

- PID可以（而且将）是不同的。
- trace11.txt、trace12.txt和trace13.txt中的/bin/ps命令的输出在不同的运行中会有所不同。然而，在/bin/ps命令的输出中，任何mysplit进程的运行状态应该是相同的。

交接说明

特定网站：在这里插入一段话，解释学生应该如何交出他们的tsh.c文件。以下是我们在CMU使用的指示。

- 确保你在`tsh.c`的标题注释中包含了你的名字和Andrew ID。
- 创建一个表格的团队名称。
  - "*ID*"，其中*ID*是你的安德鲁ID，如果你是单独工作，或
  - "ID_1 +`ID`$_2$ " 其中ID_1 是第一个团队成员的Andrew ID，ID_2 是第二个团队成员的Andrew ID。

  我们需要你以这种方式创建你的团队名称，以便我们能够自动分配你的任务。
- 在你的`tsh.c`文件中，键入。

  进行交接 `TEAM=teamname`

  其中`teamname`是上述的团队名称。
- 交稿后，如果你发现有错误，想提交一份修改稿，请输入

  ```
  make handin TEAM=teamname VERSION=2
  ```

  每次提交时都要不断递增版本号。
- 你应该通过查看以下内容来验证你的交接

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

  你在这个目录中有列表和插入的权限，但没有读或写的权限。

好运!