实验室作业：写你自己的外壳

介绍

此任务的目的是为了更加熟悉过程控制和信令传递的概念。您将通过编写一个支持作业控制的简单的Unix shell程序来实现这一点。

传出指令

从复制文件shlab-讲义开始。tar到您计划在其中进行工作的受保护的目录（实验室目录）。然后执行以下操作：

- 输入命令tarxvfshlab-讲义。tarto展开标记文件。
- 键入命令make来编译和链接一些测试例程。
- 在tsh顶部的标题注释中输入您的团队成员名称和Andrewid。c.

看着tsh。c（小shell）文件，您将看到它包含一个简单的Unix shell的功能骨架。为了帮助您开始，我们已经实现了不那么有趣的功能。您的任务

是为了完成下面列出的其余空函数。为了进行您的完整性检查，我们在参考解决方案中列出了每个函数的近似代码行数（其中包括许多注释）。

- 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：退出、fg、fg、bg和作业。[25行]
- _do bgfg：实现bg和fg内置命令。[50行]
- 等待者：等待一个前台工作的完成。[20行]
- _签名处理程序：捕获签名子信号。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]

每次你修改你的tsh时。c文件，类型为make来重新编译它。要运行shell，请在命令行中键入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，它代表用户运行程序。shell重复打印提示符，在stdin上等待命令行，然后按照命令行内容的指示执行一些操作。

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。其余的单词都是命令行参数。如果第一个单词是内置命令，则shell会立即在当前进程中执行该命令。否则，该单词将被假定为一个可执行程序的路径名。在这种情况下，shell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ajob。通常，一个作业可以由由Unix管道连接的多个子进程组成。

如果命令行以&号和“&”结束，那么作业将在后台运行，这意味着shell在打印提示符并等待下一个命令之前不会等待作业终止

线条*否则，作业将在前景中运行，这意味着shell在等待下一个命令行之前就会等待作业终止*。因此，在任何时间点，最多可以在前景中运行。但是，任意数量的作业都可以在后台运行。

例如，键入命令行

```
tsh>作业
```

使shell执行内置的作业命令。键入命令行

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

在前景中运行ls程序。按照惯例，shell确保当程序开始执行其主例程时

```
int主(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，键入命令行

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

在后台运行ls程序。

Unixshell支持作业控制的概念，它允许用户在背景和前景之间来回移动作业，并更改作业中进程的进程状态（正在运行、停止或终止）。输入ctrl-c会导致将SIGINT信号传递到前景作业中的每个进程。SIGINT的默认操作是终止该进程。类似地，输入ctrl-z会导致将签名信号传递到前景作业中的每个进程。SIGTSTP的默认操作是将进程处于停止状态，直到被接收到签名信号唤醒。Unixshell还提供了各种支持作业控制的内置命令。例如：

- 作业：列出正在运行的和已停止的后台作业。
- bg<作业>：将已停止的后台作业更改为正在运行的后台作业。
- fg<作业>：将已停止或正在运行的后台作业更改为前台正在运行的作业。
- 杀死<作业>：终止作业。

tsh规范

您的tsh shell应该有以下特性：

- 提示符应该是字符串“tsh>”。

- 用户输入的命令行应该由一个名称和0个或多个参数组成，全部由一个或多个空格分隔。如果name是内置命令，那么tsh应该立即处理它并等待下一个命令行。否则，tsh应该假定名称是一个可执行文件的路径，它在一个初始子进程的上下文中加载和运行（在这个上下文中，termjob引用了这个初始子进程）。
- tsh不需要支持管道（|）或I/O重定向（<和>）。
- 输入ctrl-c（ctrl-z）应该会导致一个SIGINT（SIGTSTP）信号被发送到当前的前景作业，以及该作业的任何后代(e。g.，它分叉的任何子进程)。如果没有前景作业，则信号应该没有影响。
- 如果命令行以&和&结束，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。
- 每个作业都可以通过进程ID（PID）或作业ID（JID）进行标识，该作业ID是由tsh分配的正整数。jid应该在命令行上用前缀“%”表示。例如，“%5”表示JID 5，而“5”表示PID 5。（我们已经为您提供了操作作业列表所需的所有例程。）
- tsh应该支持以下内置命令：
  - -quit命令终止shell。
  - -jobs命令将列出所有后台作业。
  - -bg<job>命令通过发送一个签名签名信号来重新启动<job>，然后在后台运行它。<job>参数可以是PID或JID。
  - -fg<作业>命令通过发送一个符号确认信号来重新启动<作业>，然后在前景中运行它。<job>参数可以是PID或JID。
- tsh应该收获它所有的僵尸孩子。如果任何作业因为接收到没有捕获的信号而终止，那么tsh应该识别此事件，并打印带有作业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。

参考解决方案。Linux可执行文件tshreff是shell的参考解决方案。运行此程序来解决有关shell应该如何行为的任何问题。*shell应该发出与引用解决方案相同的输出（当然，pid除外，它从运行变化到运行）。*

壳牌驱动程序。驱动程序。pl程序作为子进程执行shell，按照跟踪文件的指示发送命令和信号，并捕获并显示shell的输出。

使用-h参数来查找数据驱动程序的使用情况。pl:

```
unix> ./sdriver.pl -h
用法：s驱动程序。t<跟踪>-s<服务器>-a<args>
选项：
  -h                打印此消息
  -v                更详细
  -t <trace>        跟踪文件
  -s <shell>        要测试的外壳程序
  -一个<args>       命令行管理程序参数
  -g                生成自动平地机的输出
```

我们还提供了16个跟踪文件。您将与shell驱动程序一起使用来测试shell的正确性。低编号的跟踪文件做非常简单的测试，而高编号的测试做更复杂的测试。
您可以使用跟踪文件跟踪01在shell上运行shell驱动程序。文本文本（例如）通过输入：

unix> ./sdriver.pl -t trace01 .一个“p”

（“p”参数告诉shell不要发出提示符），或者

unix>制作测试01

类似地，要将结果与引用外壳进行比较，您可以输入以下内容在引用壳上运行跟踪驱动程序：

unix> ./sdriver.pl -t trace01 .一个“p”

或

unix>制作rtest01

请参考，tsprif。out给出了关于所有种族的参考解的输出。这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，它们生成的输出与交互式运行shell时所得到的输出相同（除了标识跟踪的初始注释）。例如：

```
低音>进行测试15
./sdriver.pl -t trace15 .一个“p”
#
# trace15 .把它们放在一起
#
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]
tsh>作业
[1]
[2]
tsh> fg %1
作业[1]（9723）被信号20停止
tsh>作业
[1]（9723）停止了。/myspin3&
[2]（9725）正在运行。/myspin4&
tsh> bg %3
%3：没有这样的工作
tsh> bg %1
[1]
tsh>作业
[1]
[2]
tsh> fg %1
tsh>退出
bass>
```

提示

- 请先阅读第8章（特殊控制流程）中的每一个字。
- 使用跟踪文件来指导shell的开发。从跟踪01开始。*确保您的shell产生与参考shell相同的输出。*然后继续进行跟踪文件跟踪02。txt，等等。
- 等待、杀戮、叉子、咒骂、设置和签名功能都会派上用场。等待的提示和等待选项也很有用。
- 当你实现你的信号处理程序时，确保向整个前景进程组发送SIGINT和SIGTSTP信号，使用“-pid”而不是“pid”在参数中的杀死函数。驱动程序。程序程序测试这个错误。
- 分配的一个棘手的部分是决定在等待fg和sigchld处理程序函数之间分配工作。我们推荐采用以下方法：

  -在等待中，使用睡眠功能的繁忙循环。
  -在签名处理程序中，只使用一个调用来等待。

  虽然其他的解决方案也是可能的，比如调用等待程序和签名处理程序，但这些可能会非常令人困惑。在处理程序中完成所有的收获更简单。
- 在eval中，父节点必须在分支子项之前使用签名屏蔽来阻止签名信号，然后解锁这些信号，在通过调用添加作业将子项添加到作业列表之后再次使用签名掩码。由于子代继承了他们父母的阻塞向量，因此子代必须确保在执行新程序之前解除签名信号。

父节点需要以这种方式阻止SIGCHLD信号，以避免竞争条件，即在父节点调用add作业之前，子节点被SIGCHLD处理程序获取（从而从作业列表中删除）。

- 诸如更多、更少、vi和emacs等程序对终端设置做了一些奇怪的事情。不要从shell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和/bin/echo。
- 从标准Unix shell时，shell在前景流程组中运行。如果shell创建了一个子进程，默认情况下该子进程也将是前景进程组的成员。因为输入ctrl-c会向前景组中的每个进程发送一个SIGINT，所以输入ctrl-c会将一个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。

  这里的解决方法是：在分叉之后，但在执行之前，子进程应该调用setpgid（0,0），这将把子进程放在一个新的进程组中，其组ID与子进程的PID相同。这确保在前景流程组中只有一个流程，即shell。当您键入ctrl-c时，shell应该捕获结果的SIGINT，然后将其转发到适当的前景作业（或者更准确地说，是包含前景作业的进程组）。

祝您好运