CS 213, Fall 2002

Lab Assignment L5: Writing Your Own Unix Shell

Assigned: Oct. 24, Due: Thu., Oct. 31, 11:59PM

Harry Bovik (bovik@cs.cmu.edu) is the lead person for this assignment.

哈利·博维克(bovik@cs.cmu.edu)是这项任务的负责人。

**Introduction**

The purpose of this assignment is to become more familiar with the concepts of process control and sig-nalling. You'll do this by writing a simple Unix shell program that supports job control.

这项作业的目的是为了更熟悉过程控制和信号的概念。您可以通过编写一个支持作业控制的简单unixshell程序来实现这一点。

**Logistics**

You may work in a group of up to two people in solving the problems for this assignment. The only “hand-in” will be electronic. Any clarifications and revisions to t he assignment will be posted on the course Web page.

你可以在一个最多两个人的小组里解决这个任务的问题。唯一的“交手”将是电子的。对作业的任何澄清和修改都将张贴在课程网页上。

**Hand Out Instructions**

**SITE-SPECIFIC: Insert a paragraph here that explains how the instructor will hand out the shlab-handout.tar file to the students. Here are the directions we use at CMU.**

**特定地点：在此处插入一段，解释讲师将如何分发shlab-讲义.tar给学生归档。以下是我们在CMU使用的指南。**

Start by copying the file shlab-handout.tar to the protected directory (the *lab directory*) in which you plan to do your work. Then do the following:

从复制文件shlab开始-讲义.tar到您计划在其中执行工作的受保护目录（lab目录）。然后执行以下操作：

* Type the command tar xvf shlab-handout.tar to expand the tarfile.
* Type the command make to compile and link some test routines.
* Type your team member names and Andrew IDs in the header comment at the top of tsh.c.

•键入命令tar xvf shlab-handout.tar展开tarfile。

•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团队成员的姓名和Andrew ID。

Looking at the tsh.c (*tiny shell*) file, you will see that it contains a functional skeleton of a simple Unix shell. To help you get started, we have already implemented the less interesting functions. Your assignment

1

is to complete the remaining empty functions listed below. As a sanity check for you, we've listed the approximate number of lines of code for each of these functions in our reference solution (which includes lots of comments).

查看tsh.c（微型shell）文件，您将看到它包含一个简单的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。 您的作业

是为了完成下面列出的其余空白功能。 为了您的健康，我们在参考解决方案中列出了每个函数的大概代码行数（其中包含很多注释）。

* eval: Main routine that parses and interprets the command line. [70 lines]
* builtin cmd: Recognizes and interprets the built-in commands: quit, fg, bg, and jobs. [25 lines]
* do bgfg: Implements the bg and fg built-in commands. [50 lines]
* waitfg: Waits for a foreground job to complete. [20 lines]
* sigchld handler: Catches SIGCHILD signals. 80 lines]
* sigint handler: Catches SIGINT (ctrl-c) signals. [15 lines]
* sigtstp handler: Catches SIGTSTP (ctrl-z) signals. [15 lines]

•eval：解析和解释命令行的主例程。 [70行]

•builtin cmd：识别并解释内置命令：quit，fg，bg和job。 [25行]

•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]

Each time you modify your tsh.c file, type make to recompile it. To run your shell, type tsh to the command line:

113/5000

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

unix> ./tsh

tsh> *[type commands to your shell here]*

**General Overview of Unix Shells**

A *shell* is an interactive command-line interpreter that runs programs on behalf of the user. A shell repeat-edly prints a prompt, waits for a *command line* on stdin, and then carries out some action, as directed by the contents of the command line.

Shell是一个交互式命令行解释器，它代表用户运行程序。 Shell重复打印提示，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。

The command line is a sequence of ASCII text words delimited by whitespace. The first word in the command line is either the name of a built-in command or the pathname of an executable file. The remaining words are command-line arguments. If the first word is a built -in command, the shell immediately executes the command in the current process. Otherwise, the word is assumed to be the pathname of an executable program. In this case, the shell forks a child process, then loads and runs the program in the context of the child. The child processes created as a result of interpreting a single command line are known collectively as a *job*. In general, a job can consist of multiple child processes connected by Unix pipes.

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，则外壳程序将在当前进程中立即执行该命令。 否则，假定该单词为可执行程序的路径名。 在这种情况下，shell会派生一个子进程，然后在该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。 通常，一个作业可以包含多个通过Unix管道连接的子进程。

If the command line ends with an ampersand ” &”, then the job runs in the *background*, which means that the shell does not wait for the job to terminate before printing the prompt and awaiting the next command line. Otherwise, the job runs in the *foreground*, which means that the shell waits for the job to terminate before awaiting the next command line. Thus, at any point in time, at most one job can be running in the foreground. However, an arbitrary number of jobs can run in the background.

如果命令行以“＆”号结尾，则作业在后台运行，这意味着Shell在打印提示并等待下一个命令行之前，不会等待作业终止。 否则，作业将在前台运行，这意味着外壳程序在等待下一个命令行之前等待作业终止。 因此，在任何时间点，前台最多只能运行一项作业。 但是，可以在后台运行任意数量的作业。

For example, typing the command line

例如，输入命令行

tsh> *jobs*

2

causes the shell to execute the built-in jobs command. Typing the command line

78/5000

使外壳执行内置作业命令。 键入命令行

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

runs the ls program in the foreground. By convention, the shell ensures that when the program begins executing its main routine

127/5000

在前台运行ls程序。 按照约定，shell确保程序开始执行其主例程时

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

the argc and argv arguments have the following values:

argc和argv参数具有以下值：

* argc == 3,
* argv[0] == ‘‘/bin/ls’’,
* argv[1]== ‘‘-l’’,
* argv[2]== ‘‘-d’’. Alternatively, typing the command line

38/5000

或者，输入命令行

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

runs the ls program in the background.

在后台运行ls程序。

Unix shells support the notion of *job control*, which allows users to move jobs back and forth between back-ground and foreground, and to change the process state (running, stopped, or terminated) of the processes in a job. Typing ctrl-c causes a SIGINT signal to be delivered to each process in the foreground job. The default action for SIGINT is to terminate the process. Similarly, typing ctrl-z causes a SIGTSTP signal to be delivered to each process in the foreground job. The default action for SIGTSTP is to place a process in the stopped state, where it remains until it is awakened by the receipt of a SIGCONT signal. Unix shells also provide various built-in commands that support job control. For example:

Unix Shell支持作业控制的概念，该概念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会使SIGINT信号传递到前台作业中的每个进程。 SIGINT的默认操作是终止该过程。 同样，键入ctrl-z会使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，该进程一直保持到收到SIGCONT信号将其唤醒为止。 Unix Shell还提供了各种支持作业控制的内置命令。 例如：

* jobs: List the running and stopped background jobs.
* bg <job>: Change a stopped background job to a running background job.
* fg <job>: Change a stopped or running background job to a running in the foreground.
* kill <job>: Terminate a job.

243/5000

•jobs：列出正在运行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在前台运行。  
  
•kill <作业>：终止作业。

**The** tsh **Specification**

Your tsh shell should have the following features:

您的tsh shell应该具有以下功能：

• The prompt should be the string “ tsh> ”.

43/5000

•提示应为字符串“ tsh>”。

3

* The command line typed by the user should consist of a name and zero or more arguments, all sepa-rated by one or more spaces. If name is a built-in command, then tsh should handle it immediately and wait for the next command line. Otherwise, tsh should assume that name is the path of an executable file, which it loads and runs in the context of an in itial child process (In this context, the term *job* refers to this initial child process).

•用户键入的命令行应由名称和零个或多个参数组成，所有参数均由一个或多个空格分隔。 如果name是内置命令，则tsh应该立即处理它并等待下一个命令行。 否则，tsh应该假定名称是可执行文件的路径，该文件在init子进程中加载并运行（在此上下文中，术语job指的是该初始子进程）。

* tsh need not support pipes (|) or I/O redirection (< and >).

61/5000

•tsh不需要支持管道（|）或I / O重定向（<和>）。

* Typing ctrl-c (ctrl-z) should cause a SIGINT (SIGTSTP) signal to be sent to the current fore-ground job, as well as any descendents of that job (e.g., any child processes that it forked). If there is no foreground job, then the signal should have no effect.

258/5000

•键入ctrl-c（ctrl-z）会导致SIGINT（SIGTSTP）信号发送到当前的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，则该信号应该无效。

* If the command line ends with an ampersand &, then tsh should run the job in the background. Otherwise, it should run the job in the foreground.
* 146/5000
* •如果命令行以＆结束，则tsh应该在后台运行作业。 否则，它将在前台运行该作业。
* Each job can be identified by either a process ID (PID) or a job I D (JID), which is a positive integer assigned by tsh. JIDs should be denoted on the command line by the prefix ' %'. For example, “ %5” denotes JID 5, and “ 5” denotes PID 5. (We have provided you with all of the routines you need for manipulating the job list.)

•每个作业都可以由进程ID（PID）或作业ID（JID）标识，该ID是由tsh分配的正整数。 JID应该在命令行上以前缀“％”表示。 例如，“％5”表示JID 5，“ 5”表示PID5。（我们为您提供了处理作业列表所需的所有例程。）

* tsh should support the following built-in commands:

**–** Thequitcommand terminates the shell.

**–** Thejobscommand lists all background jobs.

**–** Thebg <job>command restarts<job>by sending it a SIGCONT signal, and then runs it inthe background. The <job> argument can be either a PID or a JID.

**–** Thefg <job>command restarts<job>by sending it a SIGCONT signal, and then runs it inthe foreground. The <job> argument can be either a PID or a JID.

•tsh应该支持以下内置命令：

– quit命令终止外壳。

– jobs命令列出所有后台作业。

– bg <job>命令通过向其发送SIGCONT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 <job>命令通过向其发送SIGCONT信号来重新启动<job>，然后在前台运行它。 <job>参数可以是PID或JID。

* tsh should reap all of its zombie children. If any job terminates because it receives a signal that it didn't catch, then tsh should recognize this event and print a message with the job's PID and a description of the offending signal.

•tsh应该收割所有僵尸孩子。 如果任何作业由于接收到未捕获到的信号而终止，则tsh应该识别此事件，并打印一条带有该作业的PID的消息以及对该问题的信号的描述。

**Checking Your Work**

We have provided some tools to help you check your work.

**Reference solution.** The Linux executabletshrefis the reference solution for the shell. Run this programto resolve any questions you have about how your shell should behave. *Your shell should emit output that is* *identical to the reference solution* (except for PIDs, of course, which change from run to run).

367/5000

我们提供了一些工具来帮助您检查工作。  
  
参考解决方案。 Linux可执行文件tshref是Shell的参考解决方案。 运行该程序以解决有关外壳应如何操作的任何问题。 您的外壳程序应发出与参考解决方案相同的输出（当然，PID除外，其运行之间会发生变化）。

**Shell driver.** Thesdriver.plprogram executes a shell as a child process, sends it commands and signalsas directed by a *trace file* , and captures and displays the output from the shell.

Use the -h argument to find out the usage of sdriver.pl:

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参数找出sdriver.pl的用法：

4

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

We have also provided 16 trace files ( trace{01-16}.txt) that you will use in conjunction with the shell driver to test the correctness of your shell. The lower-numbered trace files do very simple tests, and the higher-numbered tests do more complicated tests.

You can run the shell driver on your shell using trace file trace01.txt (for instance) by typing:

360/5000

我们还提供了16个跟踪文件（trace {01-16} .txt），您将这些文件与shell驱动程序结合使用以测试shell的正确性。 编号较低的跟踪文件执行非常简单的测试，编号较高的测试文件进行更复杂的测试。  
  
您可以使用跟踪文件trace01.txt（例如）在外壳程序上运行外壳程序驱动程序，方法是键入：

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

(the -a "-p" argument tells your shell not to emit a prompt), or

（-a“ -p”参数告诉您的外壳不要发出提示），或者

unix> *make test01*

Similarly, to compare your result with the reference shell, you can run the trace driver on the reference shell by typing:

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

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

or

unix> *make rtest01*

For your reference, tshref.out gives the output of the reference solution on all races. This might be more convenient for you than manually running the shell driver on all trace files.

The neat thing about the trace files is that they generate the s ame output you would have gotten had you run your shell interactively (except for an initial comment that identifies the trace). For example:

作为参考，tshref.out提供了所有比赛中参考溶液的输出。 与手动在所有跟踪文件上运行Shell驱动程序相比，这可能更方便。

关于跟踪文件的整洁之处在于，它们生成的same输出，如果您以交互方式运行shell，将会获得该输出（除了标识跟踪的初始注释之外）。 例如：

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 &

5

1. (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>

**Hints**

* Read every word of Chapter 8 (Exceptional Control Flow) in your textbook.
* Use the trace files to guide the development of your shell. Sta rting with trace01.txt, make sure that your shell produces the *identical* output as the reference shell. Then move on to trace file trace02.txt, and so on.
* The waitpid, kill, fork, execve, setpgid, and sigprocmask functions will come in very handy. The WUNTRACED and WNOHANG options to waitpid will also be useful.
* When you implement your signal handlers, be sure to send SIGINT and SIGTSTP signals to the en-tire foreground process group, using ” -pid” instead of ” pid” in the argument to the kill function. The sdriver.pl program tests for this error.

•阅读教科书第8章（异常控制流）的每个单词。

•使用跟踪文件来指导外壳程序的开发。 使用trace01.txt进行填充，请确保您的shell产生与参考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函数将非常方便。 waitpid的WUNTRACED和WNOHANG选项也将很有用。

•实现信号处理程序时，请确保在kill函数的参数中使用“ -pid”而不是“ pid”将SIGINT和SIGTSTP信号发送到整个前台过程组。 sdriver.pl程序测试此错误。

* One of the tricky parts of the assignment is deciding on the allocation of work between the waitfg and sigchld handler functions. We recommend the following approach:

**–** Inwaitfg, use a busy loop around thesleepfunction.

**–** Insigchld handler, use exactly one call towaitpid.

While other solutions are possible, such as calling waitpid in both waitfg and sigchld handler, these can be very confusing. It is simpler to do all reaping in the handler.

* In eval, the parent must use sigprocmask to block SIGCHLD signals before it forks the child, and then unblock these signals, again using sigprocmask after it adds the child to the job list by calling addjob. Since children inherit the blocked vectors of their parents, the child must be sure to then unblock SIGCHLD signals before it execs the new program.

•分配的棘手部分之一是确定waitfg和sigchld处理函数之间的工作分配。 我们建议采用以下方法：

–在waitfg中，在睡眠功能周围使用繁忙循环。

–在sigchld处理程序中，仅使用一个调用来等待。

尽管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。 在处理程序中进行所有收获比较简单。

•在eval中，父级必须在派生子级之前使用sigprocmask阻止SIGCHLD信号，然后通过调用addjob将子级添加到作业列表之后，再次使用sigprocmask取消阻止这些信号。 由于孩子继承了父母的阻止向量，因此孩子必须确保在执行新程序之前先解除对SIGCHLD信号的阻止。

6

The parent needs to block the SIGCHLD signals in this way in order to avoid the race condition where the child is reaped by sigchld handler (and thus removed from the job list) *before* the parent calls addjob.

* Programs such as more, less, vi, and emacs do strange things with the terminal settings. Don't run these programs from your shell. Stick with simple text-based programs such as /bin/ls,

/bin/ps, and /bin/echo.

* When you run your shell from the standard Unix shell, your shell is running in the foreground process group. If your shell then creates a child process, by default that child will also be a member of the

foreground process group. Since typing ctrl-c sends a SIGINT to every process in the foreground group, typing ctrl-c will send a SIGINT to your shell, as well as to every process that your shell created, which obviously isn't correct.

Here is the workaround: After the fork, but before the execve, the child process should call setpgid(0, 0), which puts the child in a new process group whose group ID is identical to the child's PID. This ensures that there will be only one process, your shell, in the foreground process group. When you type ctrl-c, the shell should catch the resulting SIGINT and then forward it to the appropriate foreground job (or more precisely, the process group that contains the foreground job).

父级需要以这种方式阻塞SIGCHLD信号，以避免在父级调用addjob之前由sigchld处理程序（从作业列表中将其删除）夺取子级的竞争情况。

•诸如更多，更少，vi和emacs之类的程序会对终端设置产生奇怪的影响。不要从您的外壳运行这些程序。坚持使用基于文本的简单程序，例如/ bin / ls，

/ bin / ps和/ bin / echo。

•当您从标准Unix shell运行shell时，shell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，该子进程也将成为该子进程的成员。

前台进程组。由于键入ctrl-c会将SIGINT发送到前台组中的每个进程，因此键入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。

解决方法是：在派生之后但在execve之前，子进程应调用setpgid（0，0），这会将子进程置于组ID与该子进程的PID相同的新进程组中。这样可以确保在前台进程组中只有一个进程，即Shell。当您键入ctrl-c时，shell应该捕获结果SIGINT，然后将其转发到适当的前台作业（或更准确地说，是包含前台作业的进程组）。

**Evaluation**

Your score will be computed out of a maximum of 90 points based on the following distribution:

1. Correctness: 16 trace files at 5 points each.
2. Style points. We expect you to have good comments (5 pts) and to check the return value of EVERY system call (5 pts).

Your solution shell will be tested for correctness on a Linux machine, using the same shell driver and trace files that were included in your lab directory. Your shell sho uld produce **identical** output on these traces as the reference shell, with only two exceptions:

* The PIDs can (and will) be different.
* The output of the /bin/ps commands in trace11.txt, trace12.txt, and trace13.txt will be different from run to run. However, the running states of any mysplit processes in the output of the /bin/ps command should be identical.

您的分数将根据以下分布在90分之内计算得出：

80正确性：16个跟踪文件，每个5点。

10样式点。 我们希望您有很好的评论（5分），并检查每个系统调用的返回值（5分）。

将使用实验室目录中包含的相同外壳程序驱动程序和跟踪文件，在Linux机器上测试您的解决方案外壳程序的正确性。 您的外壳程序应在这些跟踪上产生与参考外壳程序相同的输出，只有两个例外：

•PID可以（并且将）不同。

•每次运行时，trace11.txt，trace12.txt和trace13.txt中/ bin / ps命令的输出将有所不同。 但是，/ bin / ps命令输出中任何mysplit进程的运行状态都应该相同。

**Hand In Instructions**

**SITE-SPECIFIC: Insert a paragraph here that explains how the students should hand in their tsh.c files. Here are the directions we use at CMU.**

143/5000

现场说明：在此处插入一段说明学生应如何提交其tsh.c文件的段落。 这是我们在CMU使用的说明。

7

* Make sure you have included your names and Andrew IDs in the header comment of tsh.c.
* Create a team name of the form:

**–** “ID” whereIDis your Andrew ID, if you are working alone, or

**–** “ID1+ID2” whereID1is the Andrew ID of the first team member andID2is the Andrew IDof the second team member.

We need you to create your team names in this way so that we can autograde your assignments.

* To hand in your tsh.c file, type: make handin TEAM=teamname

where teamname is the team name described above.

* After the handin, if you discover a mistake and want to submit a revised copy, type make handin TEAM=teamname VERSION=2

Keep incrementing the version number with each submission.

* You should verify your handin by looking in

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

You have list and insert permissions in this directory, but no read or write permissions.

Good luck!

•确保在tsh.c的标题注释中包含了姓名和Andrew ID。

•创建表单的团队名称：

–“ ID”，其中ID是您的Andrew ID（如果您独自工作），或者

–“ ID 1 + ID2”，其中ID1是第一个团队成员的Andrew ID，ID2是第二个团队成员的Andrew ID。

我们需要您以这种方式创建团队名称，以便我们为您的作业自动评分。

•要提交您的tsh.c文件，请输入：make handin TEAM = teamname

其中teamname是上述团队名称。

•上交后，如果发现错误并想要提交修订的副本，请输入make handin TEAM = teamname VERSION = 2

每次提交时都要增加版本号。

•您应该通过查看来验证您的交接

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

您在此目录中具有列表和插入权限，但没有读取或写入权限。

祝好运！

8