哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验设计报告

开课学期：2024 年春季

课程名称：计算机系统

实验名称：Lab2 TinyShell

实验性质：课内实验

实验时间：2024 年 5 月 24 日 地点：T2 507

学生班级：智能强基-计算机

学生学号：210010101

学生姓名：房煊梓

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2024 年 4 月

1. 需求分析

（列出 TinyShell 应支持的基本功能（如执行外部命令、内置命令、命令行解析等））

TinyShell 应支持的基本功能有：
• 命令行解析：将用户输入的命令行字符串分割成独立的命令和参数。
• 内置命令：TinyShell 通常支持一些简单的内置命令，如 quit（退出 Shell）、jobs（列出所有后台任务）、bg（后台运行）、fg（前台运行）等。
• 进程创建：对于非内置命令，需要使用 fork()和 exec()系列函数来创建新的进程并执行用户指定的命令。这是 Shell 执行大多数命令的方式。
• 信号处理：处理来自子进程或操作系统的信号，如 SIGINT（中断信号）等。

2. 设计思路

（设计 TinyShell 的整体架构，包括主函数、命令行解析函数、内置命令处理函数等。确定函数之间的调用关系和参数传递方式。）

(1)**主函数 main：**
main 函数是 shell 的主要例程，接收按值传递的 argc 和按指针传递的 argv 参数。
其主要功能与相关的函数调用关系如下：
• 解析命令行：利用 while 循环解析命令行。
• 包装信号处理函数：对于 SIGCHLD，SIGINT，SIGTSTP，SIGQUIT 信号以及相应的 handler，调用 Signal 函数，传入 signum 和 handler 对信号处理函数进行包装。
• 作业初始化：调用 initjobs 函数，传入 jobs 对作业进行初始化。
• 循环读取命令和执行：在 while(1)循环体内，首先读取命令行，然后再调用 eval 函数，传入 cmdline 来执行命令。

(2)**命令行解析函数 parseline：**
该函数对命令行 cmdline 进行解析并构建 argv 数组，接收按指针传递的 cmdline 和 argv 参数。

(3)**解析并执行命令行函数 eval：**
该函数负责解析并执行命令行，接收按指针传递的 cmdline 参数。
其主要功能与相关的函数调用关系如下：
• 解析命令行：调用 parseline 函数，传入命令行的副本 buf 和 argv 数组。
• 判断是否为内置命令：调用 builtin_cmd 函数，传入 argv 数组。
• 处理内置命令：调用 builtin_cmd 函数，传入 argv 数组。
• 处理非内置命令：在处理逻辑中需要调用 waitfg 函数，传入进程 pid。

(4)**内置命令处理函数 builtin_cmd：**
该函数负责处理内置命令，接收按指针传递的 argv 参数。
其主要功能与相关的函数调用关系如下：
• 处理 quit 命令：直接调用 exit 函数。
• 处理 jobs 命令：需要调用 listjobs 函数，传入 jobs。
• 处理 bg 和 fg 命令：需要调用 do_bgfg 函数，传入 argv 数组。

(5)**执行内置 bg 和 fg 命令的函数 do_bgfg：**
该函数负责执行内置 bg 和 fg 命令，接收按指针传入的 argv 参数。
在其处理逻辑中需要调用 waitfg 函数，传入进程 pid。

(6)**等待前台任务完成的函数 waitfg：**
该函数负责等待前台任务完成，接收按值传入的 pid 参数。

(7)**SIGCHLD 信号捕获与处理函数 sigchld_handler：**
该函数负责捕获与处理 SIGCHLD 信号，接收按值传入的 sig 参数。

(8)**SIGINT 信号捕获与处理函数 sigint_handler：**
该函数负责捕获与处理 SIGINT 信号，接收按值传入的 sig 参数。

(9)**SIGTSTP 信号捕获与处理函数 sigtstp_handler：**
该函数负责捕获与处理 SIGINT 信号，接收按值传入的 sig 参数。

3. 核心函数实现

（深入分析核心函数的实现）

3.1 void eval(char *cmdline)函数

函数功能：负责解析并执行命令行。
参　　数：用户输入的命令行 cmdline。
处理流程：首先调用 parseline 函数解析命令行，若为空命令行则不做任何操作。接下来调用 builtin_cmd 函数判断命令是否为内置命令，若是，则在 builtin_cmd 函数内分类进行处理；否则调用 fork 函数来创建子进程，并调用 execve 函数执行该命令。
要点分析：
• 父进程要在创建子进程之前用 sigprocmask 阻塞 SIGCHLD 信号，避免竞态条件。
• 子进程在执行新程序之前需要用 sigprocmask 解除阻塞。
• 在 execve 之前需要用 setpgid 显式设置进程组 ID，这样可以确保发送信号时信号被正确地发送到整个进程组。
• 在创建新进程之前阻塞 SIGCHLD 信号，在 addjob 执行结束后用 sigprocmask 解除阻塞，以保证控制流的先后顺序。

3.2 int builtin_cmd(char **argv)函数

函数功能：用于识别并执行内置的命令。
参　　数：调用 parseline 函数解析命令行输入所得到的参数。
处理流程：判断命令行参数，分类进行处理：
- “quit”：直接退出程序。
- “&”：直接返回 1。
- “jobs”：调用 listjobs 函数打印当前的作业列表，再返回 1。
- “bg”或“fg”：调用 do_bgfg 函数执行相应操作，再返回 1。

要点分析：
- 如果处理逻辑不需要直接退出程序，则在处理完之后需要返回 1。

3.3 void do_bgfg(char **argv) 函数

函数功能：专门负责实现“bg”和“fg”这两个内置命令的功能。
参　　数：调用 parseline 函数解析命令行输入所得到的参数。
处理流程：首先对参数进行检查，对于没有参数、进程不存在、job 不存在、参数不正确的情况分别提示不同的错误信息并直接返回；若没有出现以上错误，则判断该命令是“fg”命令还是“bg”命令，分别进行处理：
- “fg”：调用 kill 函数发送 SIGCONT 信号来唤醒相应的进程组，再将 job 的状态设置为 FG，最后调用 waitfg 函数，等待前台任务的完成。
- “bg”：调用 kill 函数发送 SIGCONT 信号来唤醒相应的进程组，再将 job 的状态设置为 FG，最后打印相关信息。

要点分析：
- 在调用 kill 函数时，应使用“-pid”而非“pid”作为参数，以避免潜在错误。

3.4 void waitfg(pid_t pid) 函数

函数功能：负责等待前台任务的完成。
参　　数：等待的进程 ID pid。
处理流程：通过 pid 获取对于的 job 结构体的地址，然后使用基于 sleep 函数的忙等待循环，等待前台任务的完成。
要点分析：
- 由于 while 循环的判断条件需要用到 job->state 即根据状态判断前台任务是否完成，而前台任务完成之后会调用 deletejob 函数，它会将 pid 和 jid 都置为 0，则无法根据 pid 或 jid 找到该 job 的地址，因此需要事先保存。

3.5 void sigchld_handler(int sig) 函数

函数功能：用于捕获和处理 SIGCHLD 信号。
参　　数：收到的信号 sig。
处理流程：利用 while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0) 获取终止的子进程 pid，在循环体内首先阻塞信号，然后对信号类型进行判断并进行相应的处理，最后解除阻塞。
- 子进程正常终止：直接调用 deletejob 删除任务。
- 子进程被信号终止：先打印相关信息再调用 deletejob 删除任务。
- 子进程被信号暂停：先打印相关信息再将任务的状态修改为 ST。
- 其他：打印异常终止信息。

要点分析：
- 使用 sigprocmask 在处理每个子进程前临时阻塞所有信号，处理完后再解除阻塞，避免在处理过程中被其他信号中断所可能导致的竞态条件。
- 在进行处理的前后分别保存和恢复 errno，避免错误状态受到意外更改。

4. 自测试结果

（请提供测试用例 trace02 至 trace16 的自测试输出截图，并与 tshref 的测试结果进行对比分析，确保结果的一致性；若存在差异，请明确指出不同之处。）

4.1 测试用例 trace01.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：与 tshref 测试结果相同。

4.2 测试用例 trace02.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：与 tshref 测试结果相同。

4.3 测试用例 trace03.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：与 tshref 测试结果相同。

4.4 测试用例 trace04.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (4055184) ./myspin 1 &
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：除了进程 ID 不同以外，其他部分与 tshref 测试结果相同。

4.5 测试用例 trace05.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (4055203) ./myspin 2 &
tsh> ./myspin 3 &
[2] (4055205) ./myspin 3 &
tsh> jobs
[1] (4055203) Running ./myspin 2 &
[2] (4055205) Running ./myspin 3 &
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：除了进程 ID 不同以外，其他部分与 tshref 测试结果相同。

4.6 测试用例 trace06.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (4055217) terminated by signal 2
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：除了进程 ID 不同以外，其他部分与 tshref 测试结果相同。

4.7 测试用例 trace07.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (4055226) ./myspin 4 &
tsh> ./myspin 5
Job [2] (4055228) terminated by signal 2
tsh> jobs
[1] (4055226) Running ./myspin 4 &
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：除了进程 ID 不同以外，其他部分与 tshref 测试结果相同。

4.8 测试用例 trace08.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (4055247) ./myspin 4 &
tsh> ./myspin 5
Job [2] (4055249) stopped by signal 20
tsh> jobs
[1] (4055247) Running ./myspin 4 &
[2] (4055249) Stopped ./myspin 5
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：除了进程 ID 不同以外，其他部分与 tshref 测试结果相同。

4.9 测试用例 trace09.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (4055260) ./myspin 4 &
tsh> ./myspin 5
Job [2] (4055262) stopped by signal 20
tsh> jobs
[1] (4055260) Running ./myspin 4 &
[2] (4055262) Stopped ./myspin 5
tsh> bg %2
[2] (4055262) ./myspin 5
tsh> jobs
[1] (4055260) Running ./myspin 4 &
[2] (4055262) Running ./myspin 5
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：除了进程 ID 不同以外，其他部分与 tshref 测试结果相同。

4.10 测试用例 trace10.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (4055281) ./myspin 4 &
tsh> fg %1
Job [1] (4055281) stopped by signal 20
tsh> jobs
[1] (4055281) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：除了进程 ID 不同以外，其他部分与 tshref 测试结果相同。

4.11 测试用例 trace11.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test11
./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (4055298) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1202 hvc0     Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,57600,38400,9600 hvc0 vt220
   1206 tty1     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
4047429 pts/0    Ss     0:00 -bash
4055293 pts/0    S+     0:00 make test11
4055294 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
4055295 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p
4055296 pts/0    S+     0:00 ./tsh -p
4055301 pts/0    R      0:00 /bin/ps a
4083346 pts/1    Ss+    0:00 -bash
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：所有 mysplit 进程的运行状态与 tshref 保持一致。

4.12 测试用例 trace12.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test12
./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (4055311) stopped by signal 20
tsh> jobs
[1] (4055311) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1202 hvc0     Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,57600,38400,9600 hvc0 vt220
   1206 tty1     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
4047429 pts/0    Ss     0:00 -bash
4055306 pts/0    S+     0:00 make test12
4055307 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
4055308 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
4055309 pts/0    S+     0:00 ./tsh -p
4055311 pts/0    T      0:00 ./mysplit 4
4055312 pts/0    T      0:00 ./mysplit 4
4055316 pts/0    R      0:00 /bin/ps a
4083346 pts/1    Ss+    0:00 -bash
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：所有 mysplit 进程的运行状态与 tshref 保持一致。

4.13 测试用例 trace13.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test13
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (4055325) stopped by signal 20
tsh> jobs
[1] (4055325) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1202 hvc0     Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,57600,38400,9600 hvc0 vt220
   1206 tty1     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
4047429 pts/0    Ss     0:00 -bash
4055320 pts/0    S+     0:00 make test13
4055321 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
4055322 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
4055323 pts/0    S+     0:00 ./tsh -p
4055325 pts/0    T      0:00 ./mysplit 4
4055326 pts/0    T      0:00 ./mysplit 4
4055329 pts/0    R      0:00 /bin/ps a
4083346 pts/1    Ss+    0:00 -bash
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1202 hvc0     Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,57600,38400,9600 hvc0 vt220
   1206 tty1     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
4047429 pts/0    Ss     0:00 -bash
4055320 pts/0    S+     0:00 make test13
4055321 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
4055322 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
4055323 pts/0    S+     0:00 ./tsh -p
4055332 pts/0    R      0:00 /bin/ps a
4083346 pts/1    Ss+    0:00 -bash
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：所有 mysplit 进程的运行状态与 tshref 保持一致。

4.14 测试用例 trace14.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test14
./sdriver.pl -t trace14.txt -s ./tsh -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus:Command not found
tsh> ./myspin 4 &
[1] (4055344) ./myspin 4 &
tsh> fg
fg command requires PID or %jobid argument
tsh> bg
bg command requires PID or %jobid argument
tsh> fg a
fg: argument must be a PID or %jobid
tsh> bg a
bg: argument must be a PID or %jobid
tsh> fg 9999999
(9999999): No such process
tsh> bg 9999999
(9999999): No such process
tsh> fg %2
%2: No such job
tsh> fg %1
Job [1] (4055344) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (4055344) ./myspin 4 &
tsh> jobs
[1] (4055344) Running ./myspin 4 &
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：除了进程 ID 不同以外，其他部分与 tshref 测试结果相同。

4.15 测试用例 trace15.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ 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 [1] (4055370) terminated by signal 2
tsh> ./myspin 3 &
[1] (4055372) ./myspin 3 &
tsh> ./myspin 4 &
[2] (4055374) ./myspin 4 &
tsh> jobs
[1] (4055372) Running ./myspin 3 &
[2] (4055374) Running ./myspin 4 &
tsh> fg %1
Job [1] (4055372) stopped by signal 20
tsh> jobs
[1] (4055372) Stopped ./myspin 3 &
[2] (4055374) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (4055372) ./myspin 3 &
tsh> jobs
[1] (4055372) Running ./myspin 3 &
[2] (4055374) Running ./myspin 4 &
tsh> fg %1
tsh> quit
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：除了进程 ID 不同以外，其他部分与 tshref 测试结果相同。

4.16 测试用例 trace16.txt 的输出截图

```
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$ make test16
./sdriver.pl -t trace16.txt -s ./tsh -a "-p"
#
# trace16.txt - Tests whether the shell can handle SIGTSTP and SIGINT
#     signals that come from other processes instead of the terminal.
#
tsh> ./mystop 2
Job [1] (4055396) stopped by signal 20
tsh> jobs
[1] (4055396) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (4055400) terminated by signal 2
210010101@comp0:~/comp3052tinyshelllab/shlab-handout$
```

测试结论：除了进程 ID 不同以外，其他部分与 tshref 测试结果相同。

5. 实验中遇到的问题及解决方法

（包括设计过程中的错误及测试过程中遇到的问题）

在实验中，我遇到的问题和解决方法主要有：

首先，对于实验指导书的阅读不够仔细而漏掉了一些函数的调用，导致无法通过测试。解决方法是通过在被测试的函数内使用 printf 语句打印信息来进行 debug，从而发现自己缺少相关的函数调用，添加后解决了问题。

其次，误以为需要逐一通过每个 trace 才能进行下一个 trace，于是卡住，无法输出正确结果。解决方法是在查看接下来的几个 trace 的提示之后发现可能需要综合几个 trace 的提示来编写函数才能获得最终的通过，在结合多个提示编写函数之后解决了问题。

6. 请总结实验的收获，并给出对实验内容的建议

注：本章为酌情加分项。

通过本次实验，我的收获有：通过对实验内容的分析，我进一步加深了对进程与并发、linux 异常控制流与信号机制、shell 的基本原理以及并发冲突的解决方法等知识的理解；通过一步步完善实验代码，我的代码能力得到了增强；通过不断地对程序进行测试，我的 linux 下的软件系统开发与测试能力得到了增强。

我对实验内容的建议是：建议在最后添加需要完成的函数的全部功能和实现的关键点说明，方便查漏补缺。