中国科学技术大学

实验报告

计算机系统详解

Shell Lab

学生姓名: 朱云沁

学生学号: PB20061372

完成时间: 二〇二二年六月二十六日

目录

一、 简介

1. 实验目的

- 熟悉类 Unix 系统的异常控制流, 理解进程控制, 信号量等概念.
- 学习 shell 程序进行任务控制的方式, 理解应用级并发.

2. 实验要求

基于给定的 C 语言框架, 实现一个简单的类 Unix shell 程序 `tsh`, 支持 `bg`, `fg`, `jobs` 等命令, 用于进程控制; 支持 `ctrl-c`, `ctrl-z` 等快捷键产生信号量.

`tsh.c` 中, 待补全的函数及其功能如表 1 所示.

Function	Description
`void eval(char *cmdline)`	Main routine that parses and interprets the command line.
`int builtin_cmd(char **argv)`	Recognizes and interprets the built-in commands.
`void do_bgfg(char **argv)`	Implements the `bg` and `fg` built-in commands.
`void waitfg(pid_t pid)`	Waits for a foreground job to complete.
`void sigchld_handler(int sig)`	Catches `SIGCHILD` signals.
`void sigint_handler(int sig)`	Catches `SIGINT` (`ctrl-c`) signals.
`void sigtstp_handler(int sig)`	Catches `SIGTSTP` (`ctrl-z`) signals.

表 1: Shell Lab 待实现的 C 函数

对于给定的跟踪文件 `trace01.txt` ~ `trace16.txt`, 所实现程序的输出结果应当与参考程序 `tshref` 一致[1].

3. 实验环境

本实验所有程序和命令均在以下环境执行:

Machine	MacBook Pro 13”
SoC	Apple M1, 基于 ARM, 含 8 核 CPU、8 核 GPU 及 16GB RAM
OS	macOS Monterey 12.4
IDE	Visual Studio Code 1.68.1
Docker	Docker 20.10.13
Image	Ubuntu 22.04, amd64
Packages	GCC 11.2.0, Make 4.3, Perl 5.34.0

容器配置参见附录 B `Dockerfile`

[1] 参考程序输出见附录 A `tshref.out`

二、 实验成果

在终端中使用 `make` 工具生成必要的目标，并将测试结果重定向写入文件中，输入输出信息如图 1 所示. 其中涉及的部分自定义命令，参见附录 B `Makefile`.

在 Visual Studio Code 中比较 `tsh.out` 与 `tshref.out`，发现若除去 PID 与文件名字符串，两程序输出结果完全一致，证明所实现 `tsh` 程序功能正确.

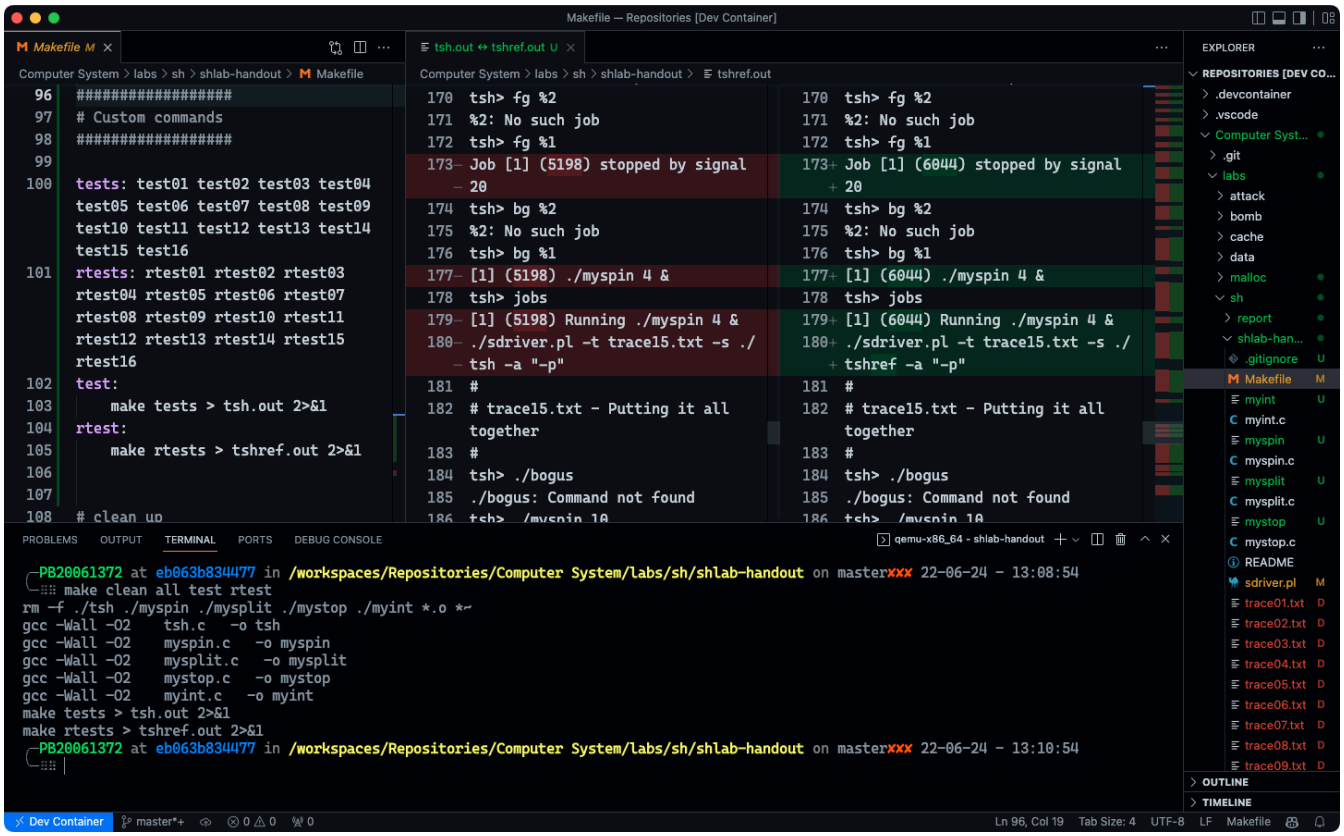

图 1: 在 Visual Studio Code 中编译运行程序，并比较输出文件

完整源文件，参见附录 B `tsh.c`; 完整输出文件，参见附录 A `tsh.out`.

三、 实验过程

1. 准备工作

通读教材第八章 Exceptional Control Flow 与实验材料，掌握进程控制的基本方式，了解 `waitpid`, `kill`, `fork`, `execve`, `setpgid` 和 `sigprocmask` 等函数的用法. 进一步，分析给定的框架代码 `tsh.c`，将已经实现的功能总结如下:

- `main` 函数: 程序初始化，包括解析参数选项，注册信号量处理事件等; 程序主循环，包括读入命令行，调用 `eval` 函数等.
- `parseline` 函数: 解析命令行，包括构建命令参数，前台 (foreground) 或后台 (background) 任务的判断等.
- 操控任务列表的函数，包括初始化，添加任务，删除任务，查找任务，打印任务列表等.

- 其他函数, 包括输出错误信息, sigaction 函数的封装, SIGQUIT 信号量的处理事件等.

根据 sdriver.pl 所描述的跟踪文件格式, 对 trace01.txt ~ trace16.txt 以及参考程序输出 tshref.out 进行分析, 将要求实现的功能总结如下:

- trace01: 读入EOF, shell 程序正常退出. 该功能已在 main 函数主循环中实现.
- trace02: 处理 quit 命令, shell 程序正常退出. 应当在 builtin_cmd 函数中实现.
- trace03: 运行一个前台任务. 相关逻辑应当在 eval 函数实现.
- trace04: 运行一个后台任务. (命令行以‘&’字符结束.) 相关逻辑应当在 eval 函数实现.
- trace05: 处理 jobs 命令, 打印后台任务列表. 应当在 builtin_cmd 函数中实现.
- trace06 & trace07: 向前台任务发送 SIGINT 信号量. 应当实现 sigint_handler 函数.
- trace08: 向前台任务发送 SIGTSTP 信号量. 应当实现 sigtstp_handler 函数.
- trace09: 处理 bg 命令, 将某个停止的前台任务转移至后台运行. 应当实现 do_bgfg 函数.
- trace10: 处理 fg 命令, 将某个后台任务转移至前台运行. 应当实现 do_bgfg 函数.
- trace11 ~ trace16: 用于验证上述功能的完整性, 包括将信号量发送至所有前台进程, 重启所有停止的进程, 错误处理, 由其他进程发送信号量等.

遵循给定框架的编码及注释风格, 补全代码, 在 Ubuntu 容器中完成所有调试. [2] 下面, 依次解释各函数的实现方法.

2. void eval(char *cmdline)

该函数在主循环中调用, 输入参数为命令行字符串. 参考教材 P755 以及实验要求和提示, 该函数应当有如下流程:

1. 调用 parseline 函数, 解析当前命令行;
2. 若参数列表为空, 直接返回;
3. 调用 builtin_cmd 函数, 若当前命令为内建命令 (quit, jobs, fg, bg), 直接处理; 否则进入下一步骤;
4. 调用 sigprocmask 等函数, 屏蔽 SIGCHILD 信号量, 防止任务进程在更新任务列表前被回收 (可参考教材 P779);
5. 调用 fork 及 setpgid 函数, 创建子进程 (即任务进程), 并设置唯一的 GID (等于其 PID);
6. 由于信号量屏蔽存在继承, 首先在子进程中将 SIGCHILD 解除屏蔽; 随后调用 execve 函数, 在子进程中运行输入的可执行目标文件 (此处应有错误处理);

[2] 出于简单考虑, 下文代码未实现所有错误处理的封装.

7. 更新任务列表后, 在父进程 (即 shell 进程) 中将 SIGCHILD 解除屏蔽.

8. 若为前台任务, 调用 waitfg 等待终止; 若为后台任务, 输出信息.

功能完整的代码如下:

```
void eval(char *cmdline) {
  char *argv[MAXARGS]; /* argument list execve() */
  char buf[MAXLINE];   /* holds modified command line */
  int bg;              /* should the job run in bg or fg */
  pid_t pid;           /* process id */
  sigset_t mask, prev; /* signal set to block certain signals */

  strcpy(buf, cmdline);
  bg = parseline(buf, argv);
  if (argv[0] == NULL) /* ignore empty lines */
    return;

  if (!builtin_cmd(argv)) {
    sigemptyset(&mask);
    sigaddset(&mask, SIGCHLD);
    sigprocmask(SIG_BLOCK, &mask, &prev); /* block SIGCHLD */

    if ((pid = fork()) == 0) {               /* child runs user job*/
      setpgid(0, 0);                         /* set process group id */
      sigprocmask(SIG_SETMASK, &prev, NULL); /* unblock SIGCHLD in child*/
      if (execve(argv[0], argv, environ) < 0) {
        fprintf(stdout, "%s: Command not found\n", argv[0]);
        exit(1);
      }
    }

    if (!bg) {
      addjob(jobs, pid, FG, cmdline);
      sigprocmask(SIG_SETMASK, &prev, NULL); /* unblock SIGCHLD */
      waitfg(pid); /* parent waits for fg job to terminate */
    } else {
      addjob(jobs, pid, BG, cmdline);
      sigprocmask(SIG_SETMASK, &prev, NULL);
      printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
    }
  }
}
```

3. int builtin_cmd(char **argv)

该函数在 eval 中被调用, 要求实现对 quit, jobs, fg, bg 等命令的判断和直接处理, 可用简单的分支结构实现. 参考教材 P755, 不难得到代码如下:

```
int builtin_cmd(char **argv) {
  if (!strcmp(argv[0], "quit")) { /* quit command */
    exit(0);
  }
  if (!strcmp(argv[0], "jobs")) { /* jobs command */
    listjobs(jobs);
    return 1;
  }
  if (!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg")) { /* bf & fg command */
    do_bgfg(argv);
    return 1;
  }
  return 0; /* not a builtin command */
}
```

若为 quit 命令, shell 程序正常退出; 若为 jobs 命令, 调用给定的 listjobs 函数即可; 若为 fg 或 bg 命令, 需在 do_bgfg 函数中处理.

4. void do_bgfg(char **argv)

该函数实现 fg 或 bg 命令的处理. 根据实验材料的描述, fg 和 bg 将某个任务在前台或后台重新启动, 该任务可用 JID 或 PID 指定. 此外, 为了通过跟踪文件 trace14.txt 的测试, 需考虑若干种错误. 因此, 实现难点在于参数字符串的处理.

该程序应有如下流程:

1. 若未输入参数, 报错并返回;
2. 若参数字符串以'%' 开头, 说明输入的是 JID, 否则输入的是 PID;
3. 判断输入的 JID 或 PID 是否合法 (是否为数字), 若不合法, 报错并返回;
4. 调用 getjobjid 或 getjobpid 函数获取目标任务的信息, 若任务不存在, 报错并返回;
5. 调用 kill 函数, 向目标进程组发送 SIGCONT 信号;
6. 修改任务状态信息. 若为前台任务, 调用 waitfg 等待终止; 若为后台任务, 输出信息.

完整代码如下:

```
void do_bgfg(char **argv) {
  struct job_t *job;
  int jid;
```

```
pid_t pid;
char *ptr;

if (argv[1] == NULL) {
  fprintf(stdout, "%s command requires PID or %%jobid argument\n", argv[0]);
  return;
}

if (argv[1][0] == '%') { /* identify by JID */
  for (ptr = argv[1] + 1; *ptr; ptr++) {
    if (!isdigit(*ptr)) {
      fprintf(stdout, "%s: argument must be a PID or %%jobid\n", argv[0]);
      return;
    }
  }
  jid = atoi(argv[1] + 1);
  job = getjobjid(jobs, jid);
  if (job == NULL) {
    fprintf(stdout, "%%%d: No such job\n", jid);
    return;
  }
} else { /* identify by PID */
  for (ptr = argv[1]; *ptr; ptr++) {
    if (!isdigit(*ptr)) {
      fprintf(stdout, "%s: argument must be a PID or %%jobid\n", argv[0]);
      return;
    }
  }
  pid = atoi(argv[1]);
  job = getjobpid(jobs, pid);
  if (job == NULL) {
    fprintf(stdout, "(%d): No such process\n", pid);
    return;
  }
}

kill(-(job->pid), SIGCONT); /* send SIGCONT to process group */

if (!strcmp(argv[0], "fg")) {
  job->state = FG;
  waitfg(job->pid); /* wait for fg job to terminate */
} else {
  job->state = BG;
```

```
    printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);
  }
  return;
}
```

5. void waitfg(pid_t pid)

该函数需在 eval 及 do_bgfg 中调用. 实现的功能为循环等待, 直至目标进程不再是前台进程 (即 pid != fgpid(jobs)). 参考程序输出显示, 等待时进程状态为 Sl+, 故使用 sleep 函数. 代码如下:

```
void waitfg(pid_t pid) {
  while (pid == fgpid(jobs))
    sleep(1);
  return;
}
```

6. void sigchld_handler(int sig)

该函数为 SIGCHLD 的处理事件, 该信号量用于回收僵尸进程. 对于不同情形, 应输出不同提示信息, 并做相应处理. 参考教材 P744 ~ P749 以及实验材料的提示, 首先应循环调用 waitpid(-1, &status, WUNTRACED | WNOHANG), 逐个回收所有停止或终止的子进程, 进而分类讨论:

- 若进程正常终止 (WIFEXITED(status)), 输出相应信息, 将该进程从任务列表中删除;
- 若进程接收到信号量而终止 (WIFSIGNALED(status)), 输出相应信息, 将该进程从任务列表中删除;
- 若进程停止 (WIFSTOPPED(status)), 输出相应信息, 将对应任务的状态修改为 ST.

代码如下:

```
void sigchld_handler(int sig) {
  pid_t pid;
  int status;

  while ((pid = waitpid(-1, &status, WUNTRACED | WNOHANG)) > 0) {
    if (WIFEXITED(status)) /* terminated normaly */
      deletejob(jobs, pid);
    else if (WIFSIGNALED(status)) { /* terminated by signal */
      printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid,
             WTERMSIG(status));
      deletejob(jobs, pid);
```

```
    } else if (WIFSTOPPED(status)) { /* stopped by signals */
      printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid), pid,
             WSTOPSIG(status));
      getjobpid(jobs, pid)->state = ST;
    }
  }
  return;
}
```

7. void sigint_handler(int sig) 及 void sigtstp_handler(int sig)

即 SIGINT 和 SIGTSTP 的处理事件, 分别由 ctrl-c, ctrl-z 快捷键触发, 用于终止或停止前台进程. 函数体内, 应当调用 kill 函数, 向前台任务的所有子进程发送 SIGINT 或 SIGTSTP. 僵尸进程的回收已经在 sigchld_handler 函数中实现. 代码如下:

```
void sigint_handler(int sig) {
  pid_t pid = fgpid(jobs);

  if (pid != 0)
    kill(-pid, SIGINT); /* send SIGINT to foreground process group */
  return;
}
void sigtstp_handler(int sig) {
  pid_t pid = fgpid(jobs);

  if (pid != 0)
    kill(-pid, SIGTSTP); /* send SIGTSTP to foreground process group */
  return;
}
```

四、 总结

完成 Shell Lab, 主要有以下收获:

- 掌握了系统调用, 进程控制, 信号量处理等重要的操作系统概念;
- 初次接触了系统级编程, 理解了通过操作系统实现应用级并发的基本方式;
- 实现了常用的进程控制命令, 如`jobs`, `bg`, `fg`, `quit`等, 深入理解了 shell 程序的原理;
- 熟悉了 Unix 系统函数库中 `fork`, `execve`, `kill`, `setpgid`, `sigprocmask` 等函数的用法;
- 学会了封装函数用于错误处理的编程方法;
- 练习了 Makefile 的编写, 正则表达式的使用等.

本实验的所有材料已上传至 GitHub:

https://github.com/HasiNed/Computer-System

附录 A　部分输出结果

1. tsh.out

```
make[1]: Entering directory '/workspaces/Repositories/Computer System/labs/sh/shlab-handout'
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (1115) ./myspin 1 &
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (1172) ./myspin 2 &
tsh> ./myspin 3 &
[2] (1178) ./myspin 3 &
tsh> jobs
[1] (1172) Running ./myspin 2 &
[2] (1178) Running ./myspin 3 &
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (1295) terminated by signal 2
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (1351) ./myspin 4 &
tsh> ./myspin 5
Job [2] (1357) terminated by signal 2
tsh> jobs
```

```
[1] (1351) Running ./myspin 4 &
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (1405) ./myspin 4 &
tsh> ./myspin 5
Job [2] (1411) stopped by signal 20
tsh> jobs
[1] (1405) Running ./myspin 4 &
[2] (1411) Stopped ./myspin 5
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (1459) ./myspin 4 &
tsh> ./myspin 5
Job [2] (1465) stopped by signal 20
tsh> jobs
[1] (1459) Running ./myspin 4 &
[2] (1465) Stopped ./myspin 5
tsh> bg %2
[2] (1465) ./myspin 5
tsh> jobs
[1] (1459) Running ./myspin 4 &
[2] (1465) Running ./myspin 5
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (1531) ./myspin 4 &
tsh> fg %1
Job [1] (1531) stopped by signal 20
tsh> jobs
[1] (1531) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
./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] (1570) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  546 pts/1    Ssl    0:00 /usr/bin/qemu-x86_64 /usr/bin/zsh /usr/bin/zsh
  744 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make all test rtest
```

```
97    935 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c make tests > tsh.out 2>&1
98    948 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make tests
99   1558 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c ./sdriver.pl -t trace11.txt -s
     ↪  ./tsh -a "-p"
100  1561 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/perl /usr/bin/perl ./sdriver.pl -t
     ↪  trace11.txt -s ./tsh -a -p
101  1564 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 ./tsh ./tsh -p
102 ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
103 #
104 # trace12.txt - Forward SIGTSTP to every process in foreground process group
105 #
106 tsh> ./mysplit 4
107 Job [1] (1617) stopped by signal 20
108 tsh> jobs
109 [1] (1617) Stopped ./mysplit 4
110 tsh> /bin/ps a
111   PID TTY      STAT   TIME COMMAND
112   546 pts/1    Ssl    0:00 /usr/bin/qemu-x86_64 /usr/bin/zsh /usr/bin/zsh
113   744 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make all test rtest
114   935 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c make tests > tsh.out 2>&1
115   948 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make tests
116  1605 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c ./sdriver.pl -t trace12.txt -s
     ↪  ./tsh -a "-p"
117  1608 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/perl /usr/bin/perl ./sdriver.pl -t
     ↪  trace12.txt -s ./tsh -a -p
118  1611 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 ./tsh ./tsh -p
119  1617 pts/1    Tl     0:00 /usr/bin/qemu-x86_64 ./mysplit ./mysplit 4
120  1620 pts/1    Tl     0:00 /usr/bin/qemu-x86_64 ./mysplit ./mysplit 4
121 ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
122 #
123 # trace13.txt - Restart every stopped process in process group
124 #
125 tsh> ./mysplit 4
126 Job [1] (1652) stopped by signal 20
127 tsh> jobs
128 [1] (1652) Stopped ./mysplit 4
129 tsh> /bin/ps a
130   PID TTY      STAT   TIME COMMAND
131   546 pts/1    Ssl    0:00 /usr/bin/qemu-x86_64 /usr/bin/zsh /usr/bin/zsh
132   744 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make all test rtest
133   935 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c make tests > tsh.out 2>&1
134   948 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make tests
135  1637 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c ./sdriver.pl -t trace13.txt -s
     ↪  ./tsh -a "-p"
136  1640 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/perl /usr/bin/perl ./sdriver.pl -t
     ↪  trace13.txt -s ./tsh -a -p
137  1643 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 ./tsh ./tsh -p
138  1652 pts/1    Tl     0:00 /usr/bin/qemu-x86_64 ./mysplit ./mysplit 4
139  1655 pts/1    Tl     0:00 /usr/bin/qemu-x86_64 ./mysplit ./mysplit 4
140 tsh> fg %1
```

```
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  546 pts/1    Ssl    0:00 /usr/bin/qemu-x86_64 /usr/bin/zsh /usr/bin/zsh
  744 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make all test rtest
  935 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c make tests > tsh.out 2>&1
  948 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make tests
 1637 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c ./sdriver.pl -t trace13.txt -s
 ↪  ./tsh -a "-p"
 1640 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/perl /usr/bin/perl ./sdriver.pl -t
 ↪  trace13.txt -s ./tsh -a -p
 1643 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 ./tsh ./tsh -p
./sdriver.pl -t trace14.txt -s ./tsh -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (1731) ./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] (1731) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (1731) ./myspin 4 &
tsh> jobs
[1] (1731) Running ./myspin 4 &
./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] (1823) terminated by signal 2
tsh> ./myspin 3 &
```

```
[1] (1835) ./myspin 3 &
tsh> ./myspin 4 &
[2] (1841) ./myspin 4 &
tsh> jobs
[1] (1835) Running ./myspin 3 &
[2] (1841) Running ./myspin 4 &
tsh> fg %1
Job [1] (1835) stopped by signal 20
tsh> jobs
[1] (1835) Stopped ./myspin 3 &
[2] (1841) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (1835) ./myspin 3 &
tsh> jobs
[1] (1835) Running ./myspin 3 &
[2] (1841) Running ./myspin 4 &
tsh> fg %1
tsh> quit
./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] (1910) stopped by signal 20
tsh> jobs
[1] (1910) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (1928) terminated by signal 2
make[1]: Leaving directory '/workspaces/Repositories/Computer System/labs/sh/shlab-handout'
```

2. tshref.out

```
make[1]: Entering directory '/workspaces/Repositories/Computer System/labs/sh/shlab-handout'
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
./sdriver.pl -t trace03.txt -s ./tshref -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (2006) ./myspin 1 &
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (2069) ./myspin 2 &
tsh> ./myspin 3 &
[2] (2075) ./myspin 3 &
tsh> jobs
[1] (2069) Running ./myspin 2 &
[2] (2075) Running ./myspin 3 &
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (2154) terminated by signal 2
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (2175) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2181) terminated by signal 2
tsh> jobs
[1] (2175) Running ./myspin 4 &
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
```

```
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (2232) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2238) stopped by signal 20
tsh> jobs
[1] (2232) Running ./myspin 4 &
[2] (2238) Stopped ./myspin 5
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (2271) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2277) stopped by signal 20
tsh> jobs
[1] (2271) Running ./myspin 4 &
[2] (2277) Stopped ./myspin 5
tsh> bg %2
[2] (2277) ./myspin 5
tsh> jobs
[1] (2271) Running ./myspin 4 &
[2] (2277) Running ./myspin 5
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (2331) ./myspin 4 &
tsh> fg %1
Job [1] (2331) stopped by signal 20
tsh> jobs
[1] (2331) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (2388) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  546 pts/1    Ssl    0:00 /usr/bin/qemu-x86_64 /usr/bin/zsh /usr/bin/zsh
  744 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make all test rtest
 1955 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c make rtests > tshref.out 2>&1
 1958 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make rtests
```

```
 2376 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
 2379 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/perl /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tshref -a -p
 2382 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 ./tshref ./tshref -p
./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (2417) stopped by signal 20
tsh> jobs
[1] (2417) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  546 pts/1    Ssl    0:00 /usr/bin/qemu-x86_64 /usr/bin/zsh /usr/bin/zsh
  744 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make all test rtest
 1955 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c make rtests > tshref.out 2>&1
 1958 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make rtests
 2405 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
 2408 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/perl /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a -p
 2411 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 ./tshref ./tshref -p
 2417 pts/1    Tl     0:00 /usr/bin/qemu-x86_64 ./mysplit ./mysplit 4
 2420 pts/1    Tl     0:00 /usr/bin/qemu-x86_64 ./mysplit ./mysplit 4
./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (2467) stopped by signal 20
tsh> jobs
[1] (2467) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  546 pts/1    Ssl    0:00 /usr/bin/qemu-x86_64 /usr/bin/zsh /usr/bin/zsh
  744 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make all test rtest
 1955 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c make rtests > tshref.out 2>&1
 1958 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make rtests
 2455 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
 2458 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/perl /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
 2461 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 ./tshref ./tshref -p
 2467 pts/1    Tl     0:00 /usr/bin/qemu-x86_64 ./mysplit ./mysplit 4
 2470 pts/1    Tl     0:00 /usr/bin/qemu-x86_64 ./mysplit ./mysplit 4
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
```

```
  546 pts/1    Ssl    0:00 /usr/bin/qemu-x86_64 /usr/bin/zsh /usr/bin/zsh
  744 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make all test rtest
 1955 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c make rtests > tshref.out 2>&1
 1958 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/make make rtests
 2455 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /bin/sh /bin/sh -c ./sdriver.pl -t trace13.txt -s
 ↪  ./tshref -a "-p"
 2458 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 /usr/bin/perl /usr/bin/perl ./sdriver.pl -t
 ↪  trace13.txt -s ./tshref -a -p
 2461 pts/1    Sl+    0:00 /usr/bin/qemu-x86_64 ./tshref ./tshref -p
./sdriver.pl -t trace14.txt -s ./tshref -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (2540) ./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] (2540) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (2540) ./myspin 4 &
tsh> jobs
[1] (2540) Running ./myspin 4 &
./sdriver.pl -t trace15.txt -s ./tshref -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [1] (2605) terminated by signal 2
tsh> ./myspin 3 &
[1] (2635) ./myspin 3 &
tsh> ./myspin 4 &
```

```
[2] (2641) ./myspin 4 &
tsh> jobs
[1] (2635) Running ./myspin 3 &
[2] (2641) Running ./myspin 4 &
tsh> fg %1
Job [1] (2635) stopped by signal 20
tsh> jobs
[1] (2635) Stopped ./myspin 3 &
[2] (2641) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (2635) ./myspin 3 &
tsh> jobs
[1] (2635) Running ./myspin 3 &
[2] (2641) Running ./myspin 4 &
tsh> fg %1
tsh> quit
./sdriver.pl -t trace16.txt -s ./tshref -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] (2692) stopped by signal 20
tsh> jobs
[1] (2692) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (2728) terminated by signal 2
make[1]: Leaving directory '/workspaces/Repositories/Computer System/labs/sh/shlab-handout'
```

附录 B　代码清单

1. tsh.c

```c
/*
 * tsh - A tiny shell program with job control
 *
 * Yunqin Zhu, PB20061372
 */
#include <ctype.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

/* Misc manifest constants */
#define MAXLINE 1024   /* max line size */
#define MAXARGS 128    /* max args on a command line */
#define MAXJOBS 16     /* max jobs at any point in time */
#define MAXJID 1 << 16 /* max job ID */

/* Job states */
#define UNDEF 0 /* undefined */
#define FG 1    /* running in foreground */
#define BG 2    /* running in background */
#define ST 3    /* stopped */

/*
 * Jobs states: FG (foreground), BG (background), ST (stopped)
 * Job state transitions and enabling actions:
 *     FG -> ST  : ctrl-z
 *     ST -> FG  : fg command
 *     ST -> BG  : bg command
 *     BG -> FG  : fg command
 * At most 1 job can be in the FG state.
 */

/* Global variables */
extern char **environ;   /* defined in libc */
char prompt[] = "tsh> "; /* command line prompt (DO NOT CHANGE) */
int verbose = 0;         /* if true, print additional output */
int nextjid = 1;         /* next job ID to allocate */
char sbuf[MAXLINE];      /* for composing sprintf messages */

struct job_t {           /* The job struct */
  pid_t pid;             /* job PID */
```

```
47    int jid;               /* job ID [1, 2, ...] */
48    int state;             /* UNDEF, BG, FG, or ST */
49    char cmdline[MAXLINE]; /* command line */
50  };
51  struct job_t jobs[MAXJOBS]; /* The job list */
52  /* End global variables */
53
54  /* Function prototypes */
55
56  /* Here are the functions that you will implement */
57  void eval(char *cmdline);
58  int builtin_cmd(char **argv);
59  void do_bgfg(char **argv);
60  void waitfg(pid_t pid);
61
62  void sigchld_handler(int sig);
63  void sigtstp_handler(int sig);
64  void sigint_handler(int sig);
65
66  /* Here are helper routines that we've provided for you */
67  int parseline(const char *cmdline, char **argv);
68  void sigquit_handler(int sig);
69
70  void clearjob(struct job_t *job);
71  void initjobs(struct job_t *jobs);
72  int maxjid(struct job_t *jobs);
73  int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline);
74  int deletejob(struct job_t *jobs, pid_t pid);
75  pid_t fgpid(struct job_t *jobs);
76  struct job_t *getjobpid(struct job_t *jobs, pid_t pid);
77  struct job_t *getjobjid(struct job_t *jobs, int jid);
78  int pid2jid(pid_t pid);
79  void listjobs(struct job_t *jobs);
80
81  void usage(void);
82  void unix_error(char *msg);
83  void app_error(char *msg);
84  typedef void handler_t(int);
85  handler_t *Signal(int signum, handler_t *handler);
86
87  /*
88   * main - The shell's main routine
89   */
90  int main(int argc, char **argv) {
91    char c;
92    char cmdline[MAXLINE];
93    int emit_prompt = 1; /* emit prompt (default) */
94
95    /* Redirect stderr to stdout (so that driver will get all output
96     * on the pipe connected to stdout) */
```

```
97    dup2(1, 2);
98
99    /* Parse the command line */
100   while ((c = getopt(argc, argv, "hvp")) != EOF) {
101     switch (c) {
102     case 'h': /* print help message */
103       usage();
104       break;
105     case 'v': /* emit additional diagnostic info */
106       verbose = 1;
107       break;
108     case 'p':          /* don't print a prompt */
109       emit_prompt = 0; /* handy for automatic testing */
110       break;
111     default:
112       usage();
113     }
114   }
115
116   /* Install the signal handlers */
117
118   /* These are the ones you will need to implement */
119   Signal(SIGINT, sigint_handler);   /* ctrl-c */
120   Signal(SIGTSTP, sigtstp_handler); /* ctrl-z */
121   Signal(SIGCHLD, sigchld_handler); /* Terminated or stopped child */
122
123   /* This one provides a clean way to kill the shell */
124   Signal(SIGQUIT, sigquit_handler);
125
126   /* Initialize the job list */
127   initjobs(jobs);
128
129   /* Execute the shell's read/eval loop */
130   while (1) {
131
132     /* Read command line */
133     if (emit_prompt) {
134       printf("%s", prompt);
135       fflush(stdout);
136     }
137     if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
138       app_error("fgets error");
139     if (feof(stdin)) { /* End of file (ctrl-d) */
140       fflush(stdout);
141       exit(0);
142     }
143
144     /* Evaluate the command line */
145     eval(cmdline);
146     fflush(stdout);
```

```
147       fflush(stdout);
148     }
149
150     exit(0); /* control never reaches here */
151   }
152
153   /*
154    * eval - Evaluate the command line that the user has just typed in
155    *
156    * If the user has requested a built-in command (quit, jobs, bg or fg)
157    * then execute it immediately. Otherwise, fork a child process and
158    * run the job in the context of the child. If the job is running in
159    * the foreground, wait for it to terminate and then return.  Note:
160    * each child process must have a unique process group ID so that our
161    * background children don't receive SIGINT (SIGTSTP) from the kernel
162    * when we type ctrl-c (ctrl-z) at the keyboard.
163    */
164   void eval(char *cmdline) {
165     char *argv[MAXARGS]; /* argument list execve() */
166     char buf[MAXLINE];   /* holds modified command line */
167     int bg;              /* should the job run in bg or fg */
168     pid_t pid;           /* process id */
169     sigset_t mask, prev; /* signal set to block certain signals */
170
171     strcpy(buf, cmdline);
172     bg = parseline(buf, argv);
173     if (argv[0] == NULL) /* ignore empty lines */
174       return;
175
176     if (!builtin_cmd(argv)) {
177       sigemptyset(&mask);
178       sigaddset(&mask, SIGCHLD);
179       sigprocmask(SIG_BLOCK, &mask, &prev); /* block SIGCHLD */
180
181       if ((pid = fork()) == 0) {              /* child runs user job*/
182         setpgid(0, 0);                        /* set process group id */
183         sigprocmask(SIG_SETMASK, &prev, NULL); /* unblock SIGCHLD in child*/
184         if (execve(argv[0], argv, environ) < 0) {
185           fprintf(stdout, "%s: Command not found\n", argv[0]);
186           exit(1);
187         }
188       }
189
190       if (!bg) {
191         addjob(jobs, pid, FG, cmdline);
192         sigprocmask(SIG_SETMASK, &prev, NULL); /* unblock SIGCHLD */
193         waitfg(pid); /* parent waits for fg job to terminate */
194       } else {
195         addjob(jobs, pid, BG, cmdline);
196         sigprocmask(SIG_SETMASK, &prev, NULL);
```

```
197         printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
198       }
199     }
200   }
201
202   /*
203    * parseline - Parse the command line and build the argv array.
204    *
205    * Characters enclosed in single quotes are treated as a single
206    * argument.  Return true if the user has requested a BG job, false if
207    * the user has requested a FG job.
208    */
209   int parseline(const char *cmdline, char **argv) {
210     static char array[MAXLINE]; /* holds local copy of command line */
211     char *buf = array;          /* ptr that traverses command line */
212     char *delim;                /* points to first space delimiter */
213     int argc;                   /* number of args */
214     int bg;                     /* background job? */
215
216     strcpy(buf, cmdline);
217     buf[strlen(buf) - 1] = ' ';   /* replace trailing '\n' with space */
218     while (*buf && (*buf == ' ')) /* ignore leading spaces */
219       buf++;
220
221     /* Build the argv list */
222     argc = 0;
223     if (*buf == '\'') {
224       buf++;
225       delim = strchr(buf, '\'');
226     } else {
227       delim = strchr(buf, ' ');
228     }
229
230     while (delim) {
231       argv[argc++] = buf;
232       *delim = '\0';
233       buf = delim + 1;
234       while (*buf && (*buf == ' ')) /* ignore spaces */
235         buf++;
236
237       if (*buf == '\'') {
238         buf++;
239         delim = strchr(buf, '\'');
240       } else {
241         delim = strchr(buf, ' ');
242       }
243     }
244     argv[argc] = NULL;
245
246     if (argc == 0) /* ignore blank line */
```

```
247       return 1;
248
249     /* should the job run in the background? */
250     if ((bg = (*argv[argc - 1] == '&')) != 0) {
251       argv[--argc] = NULL;
252     }
253     return bg;
254   }
255
256   /*
257    * builtin_cmd - If the user has typed a built-in command then execute
258    *    it immediately.
259    */
260   int builtin_cmd(char **argv) {
261     if (!strcmp(argv[0], "quit")) { /* quit command */
262       exit(0);
263     }
264     if (!strcmp(argv[0], "jobs")) { /* jobs command */
265       listjobs(jobs);
266       return 1;
267     }
268     if (!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg")) { /* bf & fg command */
269       do_bgfg(argv);
270       return 1;
271     }
272     return 0; /* not a builtin command */
273   }
274
275   /*
276    * do_bgfg - Execute the builtin bg and fg commands
277    */
278   void do_bgfg(char **argv) {
279     struct job_t *job;
280     int jid;
281     pid_t pid;
282     char *ptr;
283
284     if (argv[1] == NULL) {
285       fprintf(stdout, "%s command requires PID or %%jobid argument\n", argv[0]);
286       return;
287     }
288
289     if (argv[1][0] == '%') { /* identify by JID */
290       for (ptr = argv[1] + 1; *ptr; ptr++) {
291         if (!isdigit(*ptr)) {
292           fprintf(stdout, "%s: argument must be a PID or %%jobid\n", argv[0]);
293           return;
294         }
295       }
296       jid = atoi(argv[1] + 1);
```

```
297     job = getjobjid(jobs, jid);
298     if (job == NULL) {
299       fprintf(stdout, "%%%d: No such job\n", jid);
300       return;
301     }
302   } else { /* identify by PID */
303     for (ptr = argv[1]; *ptr; ptr++) {
304       if (!isdigit(*ptr)) {
305         fprintf(stdout, "%s: argument must be a PID or %%jobid\n", argv[0]);
306         return;
307       }
308     }
309     pid = atoi(argv[1]);
310     job = getjobpid(jobs, pid);
311     if (job == NULL) {
312       fprintf(stdout, "(%d): No such process\n", pid);
313       return;
314     }
315   }
316
317   kill(-(job->pid), SIGCONT); /* send SIGCONT to process group */
318
319   if (!strcmp(argv[0], "fg")) {
320     job->state = FG;
321     waitfg(job->pid); /* wait for fg job to terminate */
322   } else {
323     job->state = BG;
324     printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);
325   }
326   return;
327 }
328
329 /*
330  * waitfg - Block until process pid is no longer the foreground process
331  */
332 void waitfg(pid_t pid) {
333   while (pid == fgpid(jobs))
334     sleep(1);
335   return;
336 }
337
338 /*****************
339  * Signal handlers
340  *****************/
341
342 /*
343  * sigchld_handler - The kernel sends a SIGCHLD to the shell whenever
344  *     a child job terminates (becomes a zombie), or stops because it
345  *     received a SIGSTOP or SIGTSTP signal. The handler reaps all
346  *     available zombie children, but doesn't wait for any other
```

```
347  *     currently running children to terminate.
348  */
349 void sigchld_handler(int sig) {
350   pid_t pid;
351   int status;
352
353   while ((pid = waitpid(-1, &status, WUNTRACED | WNOHANG)) > 0) {
354     if (WIFEXITED(status)) /* terminated normaly */
355       deletejob(jobs, pid);
356     else if (WIFSIGNALED(status)) { /* terminated by signal */
357       printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid,
358              WTERMSIG(status));
359       deletejob(jobs, pid);
360     } else if (WIFSTOPPED(status)) { /* stopped by signals */
361       printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid), pid,
362              WSTOPSIG(status));
363       getjobpid(jobs, pid)->state = ST;
364     }
365   }
366   return;
367 }
368
369 /*
370  * sigint_handler - The kernel sends a SIGINT to the shell whenver the
371  *    user types ctrl-c at the keyboard.  Catch it and send it along
372  *    to the foreground job.
373  */
374 void sigint_handler(int sig) {
375   pid_t pid = fgpid(jobs);
376
377   if (pid != 0)
378     kill(-pid, SIGINT); /* send SIGINT to foreground process group */
379   return;
380 }
381
382 /*
383  * sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
384  *     the user types ctrl-z at the keyboard. Catch it and suspend the
385  *     foreground job by sending it a SIGTSTP.
386  */
387 void sigtstp_handler(int sig) {
388   pid_t pid = fgpid(jobs);
389
390   if (pid != 0)
391     kill(-pid, SIGTSTP); /* send SIGTSTP to foreground process group */
392   return;
393 }
394
395 /*********************
396  * End signal handlers
```

```
 *********************/

/*********************************************
 * Helper routines that manipulate the job list
 *********************************************/

/* clearjob - Clear the entries in a job struct */
void clearjob(struct job_t *job) {
  job->pid = 0;
  job->jid = 0;
  job->state = UNDEF;
  job->cmdline[0] = '\0';
}

/* initjobs - Initialize the job list */
void initjobs(struct job_t *jobs) {
  int i;

  for (i = 0; i < MAXJOBS; i++)
    clearjob(&jobs[i]);
}

/* maxjid - Returns largest allocated job ID */
int maxjid(struct job_t *jobs) {
  int i, max = 0;

  for (i = 0; i < MAXJOBS; i++)
    if (jobs[i].jid > max)
      max = jobs[i].jid;
  return max;
}

/* addjob - Add a job to the job list */
int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline) {
  int i;

  if (pid < 1)
    return 0;

  for (i = 0; i < MAXJOBS; i++) {
    if (jobs[i].pid == 0) {
      jobs[i].pid = pid;
      jobs[i].state = state;
      jobs[i].jid = nextjid++;
      if (nextjid > MAXJOBS)
        nextjid = 1;
      strcpy(jobs[i].cmdline, cmdline);
      if (verbose) {
        printf("Added job [%d] %d %s\n", jobs[i].jid, jobs[i].pid,
               jobs[i].cmdline);
```

```
447         }
448         return 1;
449       }
450     }
451     printf("Tried to create too many jobs\n");
452     return 0;
453   }
454
455   /* deletejob - Delete a job whose PID=pid from the job list */
456   int deletejob(struct job_t *jobs, pid_t pid) {
457     int i;
458
459     if (pid < 1)
460       return 0;
461
462     for (i = 0; i < MAXJOBS; i++) {
463       if (jobs[i].pid == pid) {
464         clearjob(&jobs[i]);
465         nextjid = maxjid(jobs) + 1;
466         return 1;
467       }
468     }
469     return 0;
470   }
471
472   /* fgpid - Return PID of current foreground job, 0 if no such job */
473   pid_t fgpid(struct job_t *jobs) {
474     int i;
475
476     for (i = 0; i < MAXJOBS; i++)
477       if (jobs[i].state == FG)
478         return jobs[i].pid;
479     return 0;
480   }
481
482   /* getjobpid  - Find a job (by PID) on the job list */
483   struct job_t *getjobpid(struct job_t *jobs, pid_t pid) {
484     int i;
485
486     if (pid < 1)
487       return NULL;
488     for (i = 0; i < MAXJOBS; i++)
489       if (jobs[i].pid == pid)
490         return &jobs[i];
491     return NULL;
492   }
493
494   /* getjobjid  - Find a job (by JID) on the job list */
495   struct job_t *getjobjid(struct job_t *jobs, int jid) {
496     int i;
```

```
497
498   if (jid < 1)
499     return NULL;
500   for (i = 0; i < MAXJOBS; i++)
501     if (jobs[i].jid == jid)
502       return &jobs[i];
503   return NULL;
504 }
505
506 /* pid2jid - Map process ID to job ID */
507 int pid2jid(pid_t pid) {
508   int i;
509
510   if (pid < 1)
511     return 0;
512   for (i = 0; i < MAXJOBS; i++)
513     if (jobs[i].pid == pid) {
514       return jobs[i].jid;
515     }
516   return 0;
517 }
518
519 /* listjobs - Print the job list */
520 void listjobs(struct job_t *jobs) {
521   int i;
522
523   for (i = 0; i < MAXJOBS; i++) {
524     if (jobs[i].pid != 0) {
525       printf("[%d] (%d) ", jobs[i].jid, jobs[i].pid);
526       switch (jobs[i].state) {
527       case BG:
528         printf("Running ");
529         break;
530       case FG:
531         printf("Foreground ");
532         break;
533       case ST:
534         printf("Stopped ");
535         break;
536       default:
537         printf("listjobs: Internal error: job[%d].state=%d ", i, jobs[i].state);
538       }
539       printf("%s", jobs[i].cmdline);
540     }
541   }
542 }
543 /******************************
544  * end job list helper routines
545  ******************************/
546
```

```
547 /***********************
548  * Other helper routines
549  ***********************/
550
551 /*
552  * usage - print a help message
553  */
554 void usage(void) {
555   printf("Usage: shell [-hvp]\n");
556   printf("   -h   print this message\n");
557   printf("   -v   print additional diagnostic information\n");
558   printf("   -p   do not emit a command prompt\n");
559   exit(1);
560 }
561
562 /*
563  * unix_error - unix-style error routine
564  */
565 void unix_error(char *msg) {
566   fprintf(stdout, "%s: %s\n", msg, strerror(errno));
567   exit(1);
568 }
569
570 /*
571  * app_error - application-style error routine
572  */
573 void app_error(char *msg) {
574   fprintf(stdout, "%s\n", msg);
575   exit(1);
576 }
577
578 /*
579  * Signal - wrapper for the sigaction function
580  */
581 handler_t *Signal(int signum, handler_t *handler) {
582   struct sigaction action, old_action;
583
584   action.sa_handler = handler;
585   sigemptyset(&action.sa_mask); /* block sigs of type being handled */
586   action.sa_flags = SA_RESTART; /* restart syscalls if possible */
587
588   if (sigaction(signum, &action, &old_action) < 0)
589     unix_error("Signal error");
590   return (old_action.sa_handler);
591 }
592
593 /*
594  * sigquit_handler - The driver program can gracefully terminate the
595  *    child shell by sending it a SIGQUIT signal.
596  */
```

```c
void sigquit_handler(int sig) {
  printf("Terminating after receipt of SIGQUIT signal\n");
  exit(1);
}
```

2. Makefile (节选)

```
##################
# Custom commands
##################

tests: test01 test02 test03 test04 test05 test06 test07 test08 test09 test10 test11 test12 test13 test14 test15 test16
rtests: rtest01 rtest02 rtest03 rtest04 rtest05 rtest06 rtest07 rtest08 rtest09 rtest10 rtest11 rtest12 rtest13 rtest14 rtest15 rtest16
test:
	make tests > tsh.out 2>&1
rtest:
	make rtests > tshref.out 2>&1
```

3. Dockerfile

```
# Emulate x86 architecture
FROM --platform=linux/x86_64 ubuntu:latest

# Switch apt source to ustc mirror
RUN sed -i "s/archive.ubuntu.com/mirrors.ustc.edu.cn/g" /etc/apt/sources.list
RUN sed -i "s/security.ubuntu.com/mirrors.ustc.edu.cn/g" /etc/apt/sources.list

# Install packages
RUN apt-get update \
    && DEBIAN_FRONTEND=noninteractive apt-get install -y build-essential sudo git locales zsh vim perl curl gdb\
    && apt-get clean -y

# Generate locale
RUN locale-gen --no-purge en_US.UTF-8

# Create user to show student ID
ARG USERNAME="PB20061372"
RUN useradd $USERNAME -m \
    && echo "$USERNAME ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/$USERNAME \
    && chmod 0440 /etc/sudoers.d/$USERNAME
USER $USERNAME

# Install oh-my-zsh and set theme to fino-time (my favorite)
```

```
24 RUN sh -c "$(curl -fsSL https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
   ↪  ""
25 RUN sed -i "s/robbyrussell/fino-time/g" ~/.zshrc
```