Tshlab

Analysis Report

Purpose

To become more familiar with process control and signalling by finishing a tiny linux shell.

Work--complete functions below:

- eval() : main routine that parses and interprets the cmdline
- sigchld_handler, sigint_handler, sigtstp_handler

Requirements:

- redirection of input & output
- pipes not needed
- built-commands & job control
  - quit
  - jobs
  - bg/fg job
- *reap all of its zombie chidren*

File Hierachy

tshlab/src/	Main source codes, some of which will be copied to handout and sent to students
tshlab/writeup/	Writeup of Lab sent to students
tshlab/patches/	Patches added by PKU // to fix bugs ?
tshlab/test/	Test
tshlab/test-autograder/	Test
tshlab/online-solutions/	Online solutions // may be used for cheat checker ?
tshlab/handin/	Directory of handins from students
tshlab/handout/	Directory sent to student
tshlab-rubric.txt	Other code rules

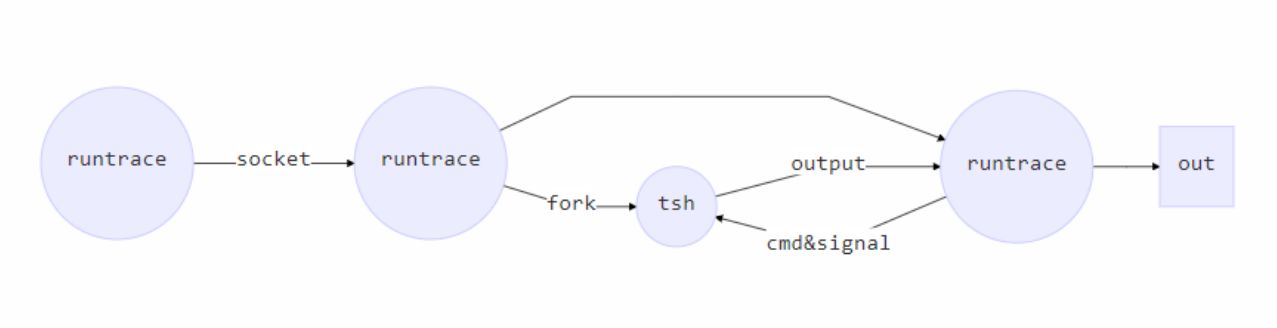
tshlab/Makefiles and other config files for autolab

Main Part -- tshlab/src

This directory contains the source for the driver program to test the tiny shell.

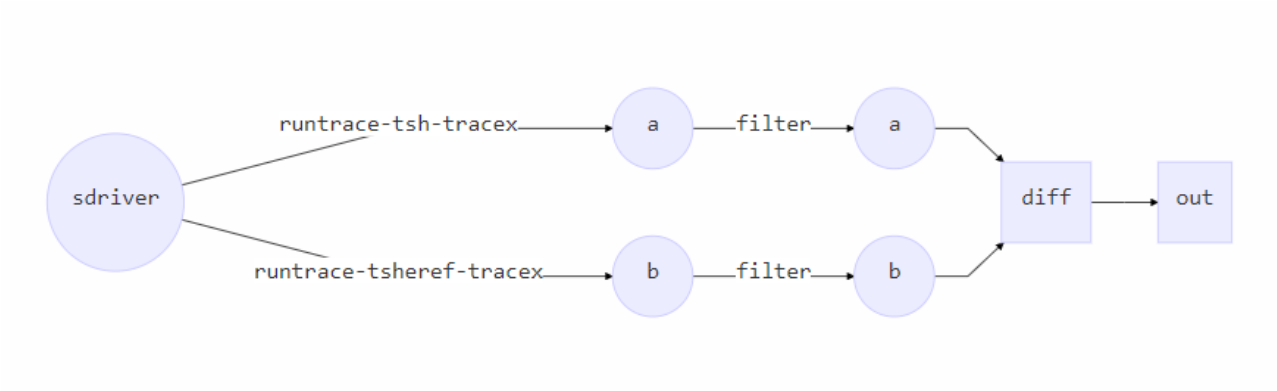
- *runtrace.c* is the program used to run and test *tsh*. It forks a child to execve *tsh*, and the parent process uses socket pairs to send signal and data to child process*(tsh)*. The *datafd* socket pair is used to send and receive datas, in other words, the input and output of *tsh*. The *syncfd* socket pair is used to send and receive synchronize signals(i.e. WAIT SIGNAL).

  This synchronization can prevent from race condition. For example, parent process sends "/bin/myspin1 10" to *tsh* through *datafd* and then hope to send a SIGINT to it. But when the SIGINT arrives, *tsh* may not have started *myspin1*, then the *myspin1* never stops and time out. So we send a SIGNAL from *myspin1* and call WAIT in parent process before we send SIGINT.



- sdriver.c is the main routine that runs the driver.

  The driver tests the functionality of a shell by calling *runtrace* for multiple times. It uses trace files as input, and compare the output of *tsh* with that from the *tshref*, ignoring blank spaces and PID. We consider the *tsh* right on some trace if and only if the output matches.



- trace*.c are trace files used to test the tiny shell. The details of them are discussed in writeup.
- my*.c are helper programs used by traces.
- *fork.c is the wrapper for fork that introduces non-determinism in the order that parent and child are executed.*
- elide.pl is used to elide code between "$begin <handout>" and "$end <handout>" to generate tsh-handout.c for students.
- tsh-broken.c is a broken version of tsh which uses 'tcsetpgrp'.

Upgrade

fix bug

- In function builtin_cmd(), opened file descriptor isn't closed.

tshlab/src/tsh.c

```
    if (tok->outfile) {
        out_fd = open(tok->outfile, O_WRONLY | O_CREAT | O_TRUNC, 0644);
        if(out_fd == -1) {
            (void) fprintf(stderr, "Error: %s Couldn't open file\n",
                           tok->outfile);
            return 1;
        }
    }

    switch (tok->builtins) {
    case BUILTIN_QUIT:
        fflush(stdout);
        fflush(stderr);
        exit(0);
    case BUILTIN_JOBS:
        listjobs(job_list, out_fd);
        return 1;
    case BUILTIN_FG:
    case BUILTIN_BG:
        do_bgfg(tok->argv, out_fd);
        return 1;
    default:
        /* not a builtin command */
        (*input_fd) = in_fd;
        (*output_fd) = out_fd;
        return 0;
    }

    return 0;    /* not a builtin command */
```

```
    if (tok->outfile) {
        out_fd = open(tok->outfile, O_WRONLY | O_CREAT | O_TRUNC, 0644);
        if(out_fd == -1) {
            (void) fprintf(stderr, "Error: %s Couldn't open file\n",
                           tok->outfile);
            return 1;
        }
    }

    (*input_fd) = in_fd;
    (*output_fd) = out_fd;

    switch (tok->builtins) {
    case BUILTIN_QUIT:
        fflush(stdout);
        fflush(stderr);
        exit(0);
    case BUILTIN_JOBS:
        listjobs(job_list, out_fd);
        return 1;
    case BUILTIN_FG:
    case BUILTIN_BG:
        do_bgfg(tok->argv, out_fd);
        return 1;
    default:
        /* not a builtin command */
        return 0;
    }
```

- Gcc error -- sprintf buffer isn't big enough.

```
lzzz@DESKTOP-5DIRHK6:/mnt/c/Users/Joe/OneDrive - pku.edu.cn/Grade_3/OSP/OSP/Project/tshlab-handout$ make
/usr/bin/gcc -Wall -g -Werror   -Wl,--wrap,fork -o tsh tsh.c fork.c
tsh.c: In function ‘ listjobs’ :
tsh.c:713:31: error: ‘ sprintf’  may write a terminating nul past the end of the destination [-Werror=format-overflow=]
             sprintf(buf, "%s\n", job_list[i].cmdline);
                                 ^
tsh.c:713:13: note: ‘ sprintf’  output between 2 and 1025 bytes into a destination of size 1024
             sprintf(buf, "%s\n", job_list[i].cmdline);
             ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cc1: all warnings being treated as errors
Makefile:19: recipe for target 'tsh' failed
make: *** [tsh] Error 1
```

- When job numbers exceed MAXJOBS, segmentation fault happens.

tshlab/src/tsh.c MODIFIED Side-by-side diff View file Comment •••

```
326  326            signals handlers can run again. Notice that the global job
327  327            list is protected because the signals for the three
328  328            handlers that manipulate it are blocked. */
329       -        addjob(job_list, pipe_counts + 1, pids, status, cmdline);
330       -
     329  +        if (!addjob(job_list, pipe_counts + 1, pids, status, cmdline)) {
     330  +            return;
     331  +        }
331  332           char buf[MAXLINE];
332  333           struct job_t *j = getjobpid(job_list, pids[0]);
333  334           if (!bg) {
```

- make "pkill" only kill its process.

tshlab/src/trace10.txt MODIFIED

```diff
 WAIT

-/bin/echo -e tsh\076 /usr/bin/pkill myspin1
+/bin/echo -e tsh\076 /usr/bin/pkill -u $USER myspin1
 NEXT
-/usr/bin/pkill myspin1
+/usr/bin/pkill -u $USER myspin1
 NEXT

 SIGNAL
```

pipe

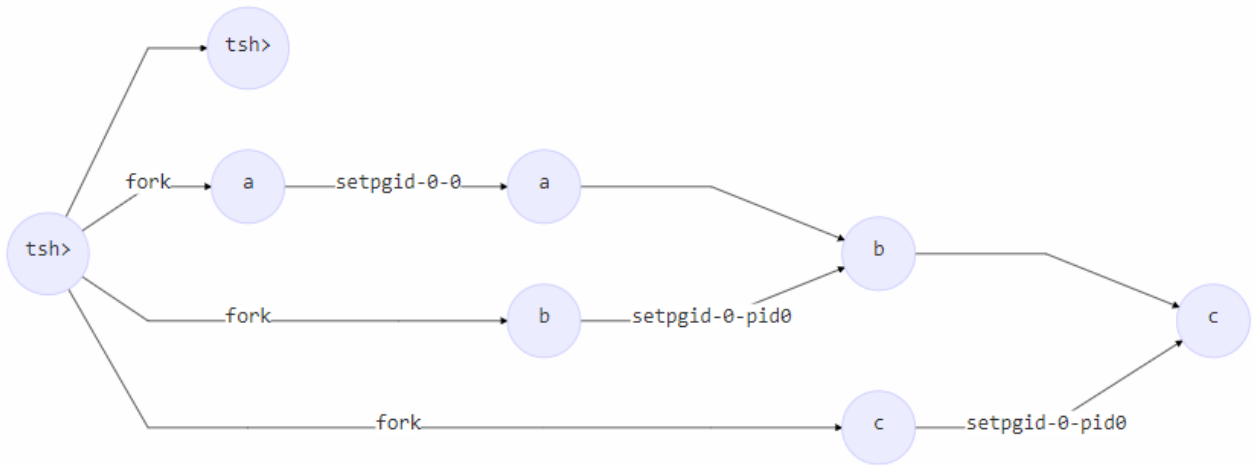
Make the *tsh* support pipes.

If *tsh* support pipes, then one job may consists of several processes(connected by pipes). So we modify the job_t struct, add PGID, change pid to pids[].

To monitor the real *shell* in linux, we add these requirements:

- Child processes should be forked by *tsh*, not the child of *tsh*.
- Child processes should share the same PGID, which is different of the PGID of *tsh*.
- A job is considered completed only if all of its processes have finished.

Attention: There will be race condition between setpgid(0, 0) and setpgid(0, pid0); process b and c should call setpgid after process a.



Additionally, we use SIGUSR1 to synchronize and prevent from race condition.

```
                }
                Signal(SIGTTIN, SIG_DFL);
                Signal(SIGTTOU, SIG_DFL);
-               if (i == 0)
+               if (i == 0) {
                    setpgid(0, 0);
+                   /* send a SIGUSR1 to parent */
+                   kill(getppid(), SIGUSR1);
+               }
                else {
                    setpgid(0, childgid);
                }
```

For simplicity, *tsh* need not to support built-in commands with pipes.

Job struct & perl

Modify job struct and related operations(e.g. printf).

```
61  struct job_t {              /* The job struct */
62 -    pid_t pid;              /* job PID */
63      int jid;                /* job ID [1, 2, ...] */
64      int state;              /* UNDEF, BG, FG, or ST */
65      char cmdline[MAXLINE];  /* command line */
66  };
67  struct job_t job_list[MAXJOBS]; /* The job list */
```

```
61  struct job_t {              /* The job struct */
62 +    int pgid;
63 +    int procnum;            /* The number of processes in a job */
64      int jid;                /* job ID [1, 2, ...] */
65      int state;              /* UNDEF, BG, FG, or ST */
66 +    pid_t pids[MAXPIPES + 1];           /* job PID */
67      char cmdline[MAXLINE];  /* command line */
68  };
69  struct job_t job_list[MAXJOBS]; /* The job list */
```

Modify sdriver to filter "(pid, pid, ..., pid)".

```
 * These transformations allow us to do diffs on the outputs of
 * different runs of different shells.
 */
-#define PERLPROG "while(<>){chomp; s/\\s+//g; s/\\(\\d+\\)/\\(PID\\)/g; print \"$_\"}"
+#define PERLPROG "while(<>){chomp; s/\\s+//g; s/\\(.*\\)/[PIDS]/g; print \"$_\"}"

/********************
 * Global variables
```

trace

fix trace10:

```
SIGINT
NEXT
```

```
-/bin/echo -e tsh\076 /bin/sh -c \047/bin/ps h | /bin/fgrep -v grep | /bin/fgrep mysplit\047
+/bin/echo -e tsh\076 /bin/sh -c \047/bin/ps h \0174 /bin/fgrep -v grep \0174 /bin/fgrep mysplit\047
NEXT
/bin/sh -c '/bin/ps h | /bin/fgrep -v grep | /bin/fgrep mysplit'
NEXT
```

Add more traces to test for pipes:

- *trace25* Pipe - use commands with a pipe to xargs
- *trace26* Pipe - test if children are forked by tsh
- *trace27* Pipe - background jobs with pipe
- *trace28* Pipe - use commands with multiple pipes

There are four traces which test for pipes at 4 pts each. Only one trace tests for multiple pipes.

Source Code & Git Repositroy & Writeup

See git repository.

Author	Commit	Message
Lzzz	`b24eb1e`	modify writeup -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`cfeea76`	add comment by Cheng Sheng <1600012909@pku.edu.cn>
Lzzz	`ff33525`	modify config.h, tsh head -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`1c98b8e`	add environment -- implemented by Cheng Sheng <1600012909@pku.edu.cn>
Lzzz	`75c92dd`	modify PERLPROG to ignore anything between "()" -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`36f6982`	fix fgpid bug, add fgpgid in job struct -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`bc6fc85`	fix race bug -- add SIGUSR1 to synchronize -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`302b7e9`	fix "loop initial declaration" bug, robust code fix -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`7a11f67`	fix pkill bug -- should implement ENV $USER -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`0f39e50`	add traces to test pipes -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`0530e85`	fix "too many jobs" segmentation fault bug -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`da5722d`	fix code to more robust -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`6898cc3`	fully implement pipes & add helper functions -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`6f7d300`	implement pipes -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`192e465`	fix traces: substitute | with \0174 -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`21679d5`	fix bug: tsh.c -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`164313b`	rewrite tsh.c and then add deal_pipe
Lzzz	`2484bcf`	fix bug: builtin_cmd may not close file descriptors -- SoB Li Zhuo <lizhmq@pku.edu.cn>
Lzzz	`294f118`	init

To-do List

- Builtin-cmd & pipes

  For simplicity, current tsh doesn't support builtin-cmd with pipes.

- Multi-process control

  For foreground job, tsh will wait for all processes to terminate. In other words, a job is considered terminated if all of its processes has terminated.

  But what's the status of a job if some processes in it are stopped while others are running? This is undefined in tsh.

- Background job read/write

  Jobs in tsh is actually back ground job in bash(tsh is the foreground job), so they cann't read/write from/to stdin/stdout. Otherwise SIGTTIN/SIGTTOUT will be raised.

- Evaluation

  We judge the tsh by its output which may be improved by other methods.