日志 2.8

【20%】学会编译 tsh.c 调用 tsh 文件 trace.txt 的功能验证方法。

调用系统命令，查看文件。

```
zhangjiwei@zhangjiwei-virtual-machine:~$ cd shlab-handout
zhangjiwei@zhangjiwei-virtual-machine:~/shlab-handout$ ls
Makefile  mysplit.c    trace02.txt  trace08.txt  trace14.txt  tshref.out
myint     mystop       trace03.txt  trace09.txt  trace15.txt
myint.c   mystop.c     trace04.txt  trace10.txt  trace16.txt
myspin    README       trace05.txt  trace11.txt  tsh
myspin.c  sdriver.pl   trace06.txt  trace12.txt  tsh.c
mysplit   trace01.txt  trace07.txt  trace13.txt  tshref
zhangjiwei@zhangjiwei-virtual-machine:~/shlab-handout$
```

根据 shlab.pdf 的说明，编译 tsh.c 只需键入指令： unix > make

验证 tracexx.txt 的功能可以通过对比 tsh 的与 tshref.out 来验证，

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

或者输入： unix > make test01

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

或者输入 unix > make rtest01

通过对比这两个指令得到的结果即可验证自己是否实现了对应功能

【20%】用 trace01 和 trace02 比较 tsh 和 tshref 执行结果并分析。

验证 trace01

输入指令 make test01

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

输入指令 make rtest01

```
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
```

它正确读出文件中数据，然后在读取 close 的时候，关闭文件，并等待（wait 感觉测不出来，貌似直接关闭了，我后来加了很多 wait 也是一样的

比较：发现效果相同，第一个测试不能进行有效的判别，所以也不能获得任何信息

验证 trace02

输入指令 make test02

```
zhangjiwei@zhangjiwei-virtual-machine:~/shlab-handout$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
^CMakefile:31: recipe for target 'test02' failed
make: *** [test02] 中断
```

输入指令 make rtest02

```
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

在 tsh 文件中 需要输入 ctrl+c 退出
原文件中执行 quit 指令 目前我们还没有填充函数，tsh 不能实现该功能

【30%】编程实现 quit 内置命令，补齐文件 tsh.c 中的函数 eval()和函数 builtin_cmd()与 quit 相关的部分。

（1）eval()中填充 quit 部分 首先调用 parseline 函数,得到具体函数,传参给 builtin_cmd()。

```
void eval(char *cmdline)
{
        char *argv[MAXARGS];
        parseline(cmdline,argv);
        builtin_cmd(argv);
        return;
}
```

（2）builtin_cmd()中填充 quit 部分 使用 strcmp 函数，判断是否为 quit 指令，是则退出。

```
int builtin_cmd(char **argv)
{//如果是quit命令，则退出。
        if(!strcmp(argv[0],"quit"))
        exit(0);
        return 0;     /* not a builtin command */
}
```

（3）完成 trace02 功能 直接退出

```
zhangjiwei@zhangjiwei-virtual-machine:~/shlab-handout$ make
gcc -Wall -O2    tsh.c   -o tsh
zhangjiwei@zhangjiwei-virtual-machine:~/shlab-handout$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

```
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

【20%】使用 trace03 验证 quit 命令。
输入指令 make test03

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

输入指令 make rtest03

```
./sdriver.pl -t trace03.txt -s ./tshref -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```

发现：正确的结果多输出了一句 tsh> quit 原因是 trace03 中使用了 echo 命令

【10%】了解 eval()与 execve 执行流程和 fork()多进程运行方式。

eval

eval 函数的作用是评估并解读用户刚刚输入的命令行。如果用户请求了内置 命令（quit, jobs, bg or fg）就立刻执行这个操作。否则 fork 开辟一个子进 程。在子进程中加载并运行该程序。如果一个进程在前台执行，就会在前台进程 执行前一直等待。

execve

execve 函数在当前进程的上下文中加载并运行一个新程序。而且带参数列表 argv 和环境变量列表 envp。当出现错误时，execve 会返回到调用程序。

fork

fork 函数被调用之后会创建一个子进程，同时会有两次返回，一个是子进程 的 pid 一个是 0.两个进程有相同的用户栈，相同的本地变量值，相同的堆，相同 的全局变量值和相同的代码。而且父子进程相互独立，是并发运行的。