System Programming Lab #2

2021-03-31

SP-TAs

Lab Assignment #2 – Shell Lab

- Download skeleton code from eTL

  `shlab.tar`

- Hand In
  - Upload your files **eTL**
  - A zip file should include your implementation (tsh.c) and a report

- PLEASE, **READ** the Hand-out!!!
  - Hints section provides helpful information to implement your shell

- Assigned: Mar. 31st
- Deadline: Apr. 14th, 23:59:59 PM

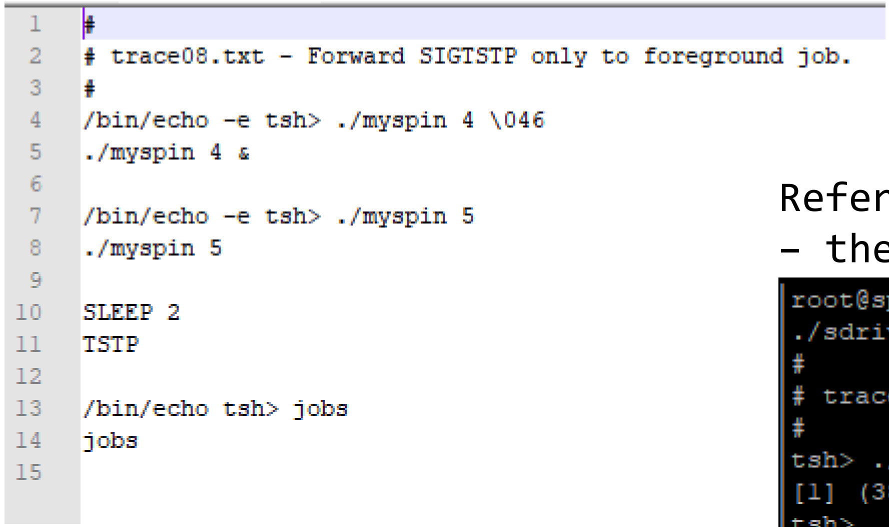
- Next class(4/7) will be zoom Q&A session

Shell Lab

- To become more familiar with the concepts of process control and signaling.
- Writing a simple Unix shell program that supports job control.

Let’s start the fun part!

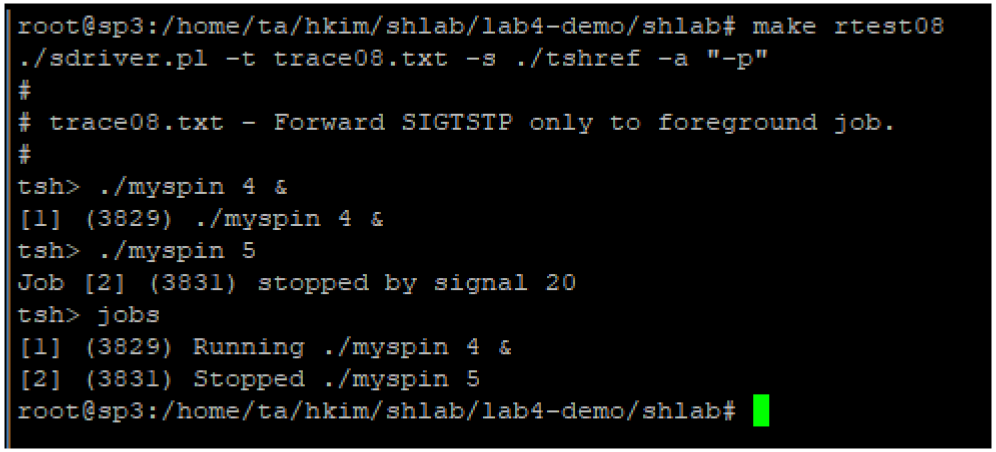
trace08.txt



```
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo -e tsh> ./myspin 5
./myspin 5

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs

```

Reference Output
- the solution

# make rtest{NN}
# make rtest08



```
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab# make rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (3829) ./myspin 4 &
tsh> ./myspin 5
Job [2] (3831) stopped by signal 20
tsh> jobs
[1] (3829) Running ./myspin 4 &
[2] (3831) Stopped ./myspin 5
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab#
```

Shell Lab

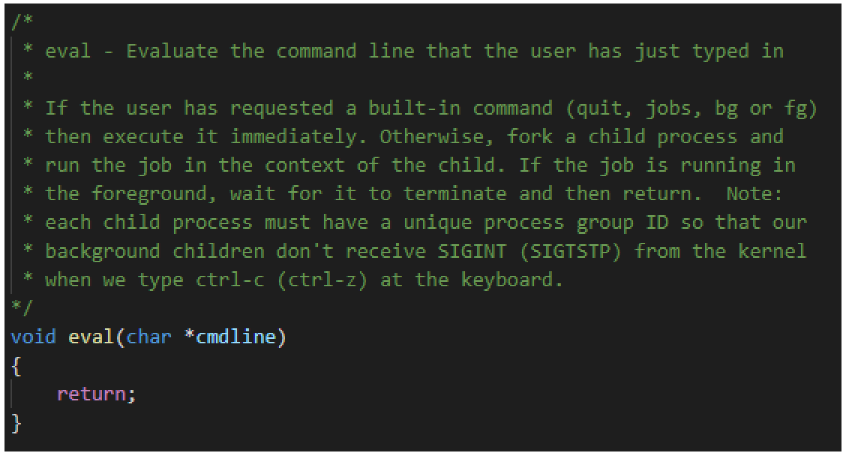
- There's a lot of starter code
  - Look over it so you don't needlessly repeat work
- Don't be afraid to write your own helper functions; this is not a simple assignment
- SIGCHLD handler may have to reap multiple children per call
- Try actually using your shell and seeing if/where it fails
  - Can be easier than looking at the driver output

You will implement…

```
/* Here are the functions that you will implement */
void eval(char *cmdline);
int builtin_cmd(char **argv);
void do_bgfg(char **argv);
void waitfg(pid_t pid);

void sigchld_handler(int sig);
void sigtstp_handler(int sig);
void sigint_handler(int sig);
```

- `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]

Guide to start your implementation



```c
/*
 * eval - Evaluate the command line that the user has just typed in
 *
 * If the user has requested a built-in command (quit, jobs, bg or fg)
 * then execute it immediately. Otherwise, fork a child process and
 * run the job in the context of the child. If the job is running in
 * the foreground, wait for it to terminate and then return.  Note:
 * each child process must have a unique process group ID so that our
 * background children don't receive SIGINT (SIGTSTP) from the kernel
 * when we type ctrl-c (ctrl-z) at the keyboard.
*/
void eval(char *cmdline)
{
    return;
}
```

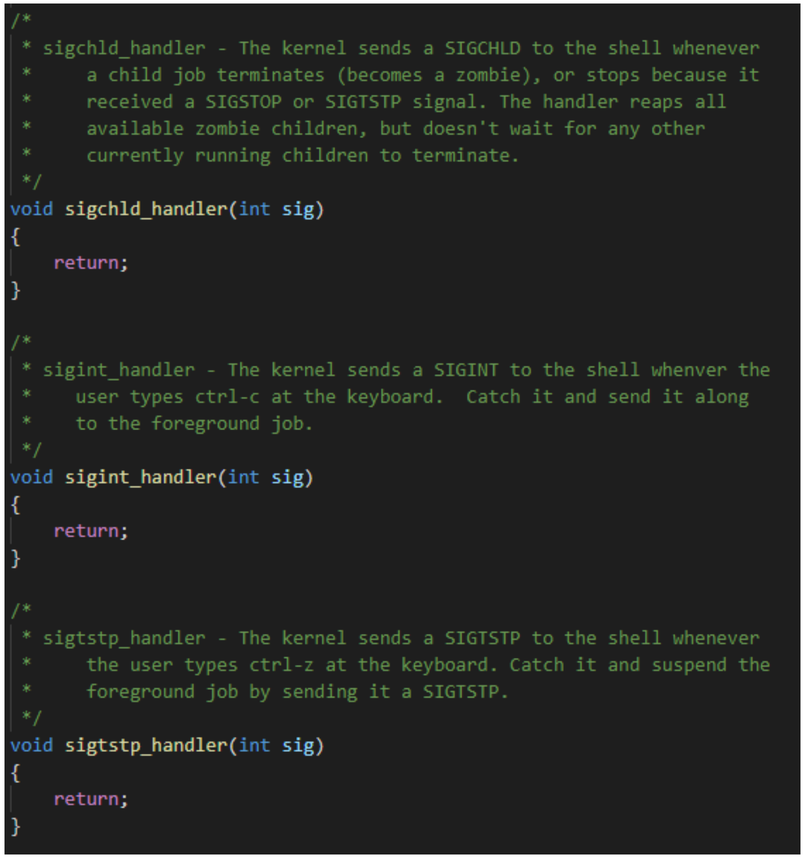
0. *parse & check cmd*
1. block signals
2. create child process
3. do the job

3.1 <child process>

1) Unblock signal
2) Get new process group ID
3) Load & run new program

3.2 <parent process>

1) Addjob
2) Unblock signal
3) (if bg) print log message



```c
/*
 * sigchld_handler - The kernel sends a SIGCHLD to the shell whenever
 *     a child job terminates (becomes a zombie), or stops because it
 *     received a SIGSTOP or SIGTSTP signal. The handler reaps all
 *     available zombie children, but doesn't wait for any other
 *     currently running children to terminate.
 */
void sigchld_handler(int sig)
{
    return;
}

/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenver the
 *    user types ctrl-c at the keyboard.  Catch it and send it along
 *    to the foreground job.
 */
void sigint_handler(int sig)
{
    return;
}

/*
 * sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
 *     the user types ctrl-z at the keyboard. Catch it and suspend the
 *     foreground job by sending it a SIGTSTP.
 */
void sigtstp_handler(int sig)
{
    return;
}
```

4. Implement signal handler

Shell Lab

- Read man pages. You may find the following functions helpful:
  - `sigemptyset()`
  - `sigaddset()`
  - **`sigprocmask()`**
  - `sigsuspend()`
  - `waitpid()`
  - `open()`
  - `dup2()`
  - `setpgid()`
  - `kill()`

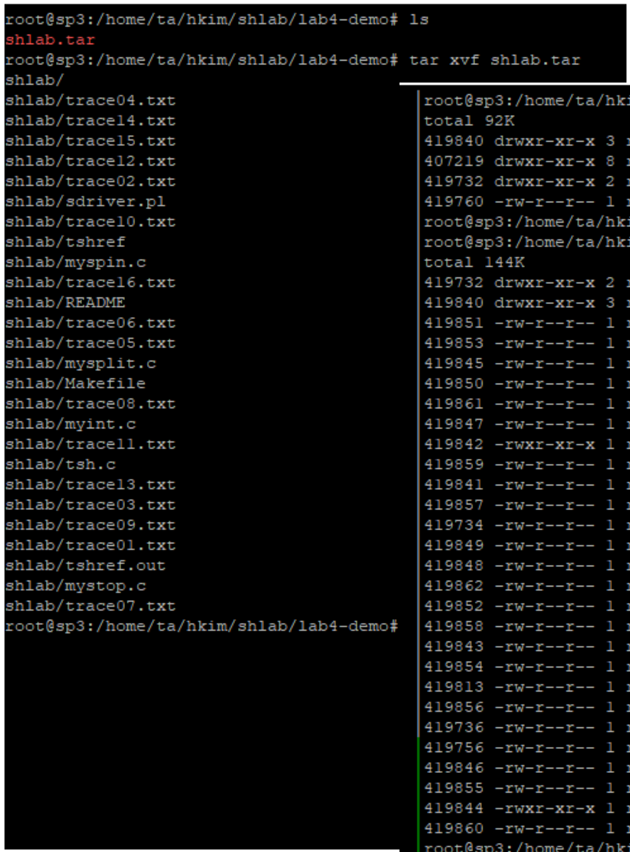
In `eval`, the parent must use `sigprocmask` to block `SIGCHLD` signals before it forks the child, and then unblock these signals, again using `sigprocmask` after it adds the child to the job list by calling `addjob`

Shell Lab Testing

- Run your shell
    - This is the *fun* part!
- tshref
    - How should the shell behave?
- runtrace
    - Each trace tests one feature.

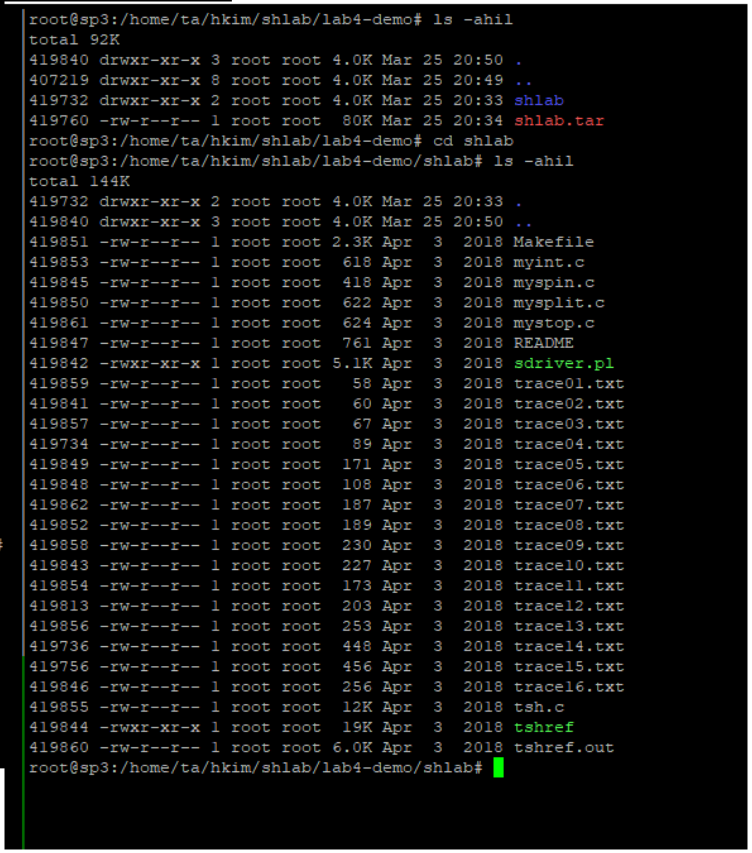
Let's start the fun part!

`# tar xvf shlab.tar`



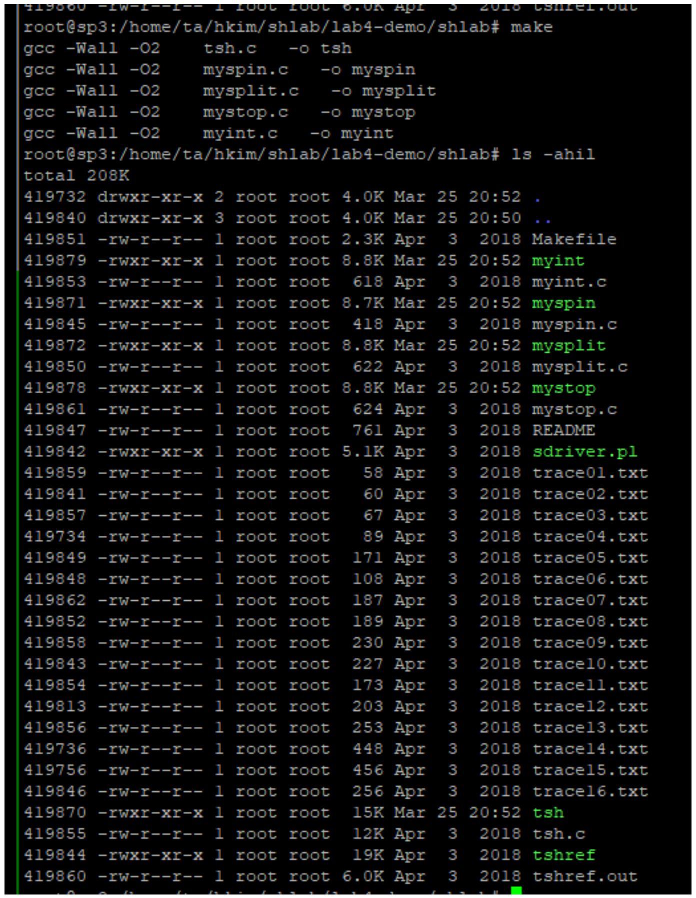
```
root@sp3:/home/ta/hkim/shlab/lab4-demo# ls
shlab.tar
root@sp3:/home/ta/hkim/shlab/lab4-demo# tar xvf shlab.tar
shlab/
shlab/trace04.txt
shlab/trace14.txt
shlab/trace15.txt
shlab/trace12.txt
shlab/trace02.txt
shlab/sdriver.pl
shlab/trace10.txt
shlab/tshref
shlab/myspin.c
shlab/trace16.txt
shlab/README
shlab/trace06.txt
shlab/trace05.txt
shlab/mysplit.c
shlab/Makefile
shlab/trace08.txt
shlab/myint.c
shlab/trace11.txt
shlab/tsh.c
shlab/trace13.txt
shlab/trace03.txt
shlab/trace09.txt
shlab/trace01.txt
shlab/tshref.out
shlab/mystop.c
shlab/trace07.txt
root@sp3:/home/ta/hkim/shlab/lab4-demo#
```

After decompression



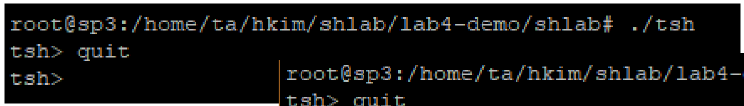
```
root@sp3:/home/ta/hkim/shlab/lab4-demo# ls -ahil
total 92K
419840 drwxr-xr-x 3 root root 4.0K Mar 25 20:50 .
407219 drwxr-xr-x 8 root root 4.0K Mar 25 20:49 ..
419732 drwxr-xr-x 2 root root 4.0K Mar 25 20:33 shlab
419760 -rw-r--r-- 1 root root  80K Mar 25 20:34 shlab.tar
root@sp3:/home/ta/hkim/shlab/lab4-demo# cd shlab
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab# ls -ahil
total 144K
419732 drwxr-xr-x 2 root root 4.0K Mar 25 20:33 .
419840 drwxr-xr-x 3 root root 4.0K Mar 25 20:50 ..
419851 -rw-r--r-- 1 root root 2.3K Apr  3  2018 Makefile
419853 -rw-r--r-- 1 root root  618 Apr  3  2018 myint.c
419845 -rw-r--r-- 1 root root  418 Apr  3  2018 myspin.c
419850 -rw-r--r-- 1 root root  622 Apr  3  2018 mysplit.c
419861 -rw-r--r-- 1 root root  624 Apr  3  2018 mystop.c
419847 -rw-r--r-- 1 root root  761 Apr  3  2018 README
419842 -rwxr-xr-x 1 root root 5.1K Apr  3  2018 sdriver.pl
419859 -rw-r--r-- 1 root root   58 Apr  3  2018 trace01.txt
419841 -rw-r--r-- 1 root root   60 Apr  3  2018 trace02.txt
419857 -rw-r--r-- 1 root root   67 Apr  3  2018 trace03.txt
419734 -rw-r--r-- 1 root root   89 Apr  3  2018 trace04.txt
419849 -rw-r--r-- 1 root root  171 Apr  3  2018 trace05.txt
419848 -rw-r--r-- 1 root root  108 Apr  3  2018 trace06.txt
419862 -rw-r--r-- 1 root root  187 Apr  3  2018 trace07.txt
419852 -rw-r--r-- 1 root root  189 Apr  3  2018 trace08.txt
419858 -rw-r--r-- 1 root root  230 Apr  3  2018 trace09.txt
419843 -rw-r--r-- 1 root root  227 Apr  3  2018 trace10.txt
419854 -rw-r--r-- 1 root root  173 Apr  3  2018 trace11.txt
419813 -rw-r--r-- 1 root root  203 Apr  3  2018 trace12.txt
419856 -rw-r--r-- 1 root root  253 Apr  3  2018 trace13.txt
419736 -rw-r--r-- 1 root root  448 Apr  3  2018 trace14.txt
419756 -rw-r--r-- 1 root root  456 Apr  3  2018 trace15.txt
419846 -rw-r--r-- 1 root root  256 Apr  3  2018 trace16.txt
419855 -rw-r--r-- 1 root root  12K Apr  3  2018 tsh.c
419844 -rwxr-xr-x 1 root root  19K Apr  3  2018 tshref
419860 -rw-r--r-- 1 root root 6.0K Apr  3  2018 tshref.out
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab#
```

`# make`

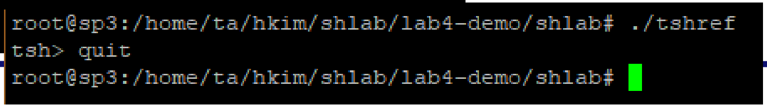


```
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab# make
gcc -Wall -O2    tsh.c   -o tsh
gcc -Wall -O2    myspin.c   -o myspin
gcc -Wall -O2    mysplit.c   -o mysplit
gcc -Wall -O2    mystop.c   -o mystop
gcc -Wall -O2    myint.c   -o myint
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab# ls -ahil
total 208K
419732 drwxr-xr-x 2 root root 4.0K Mar 25 20:52 .
419840 drwxr-xr-x 3 root root 4.0K Mar 25 20:50 ..
419851 -rw-r--r-- 1 root root 2.3K Apr  3  2018 Makefile
419879 -rwxr-xr-x 1 root root 8.8K Mar 25 20:52 myint
419853 -rw-r--r-- 1 root root  618 Apr  3  2018 myint.c
419871 -rwxr-xr-x 1 root root 8.7K Mar 25 20:52 myspin
419845 -rw-r--r-- 1 root root  418 Apr  3  2018 myspin.c
419872 -rwxr-xr-x 1 root root 8.8K Mar 25 20:52 mysplit
419850 -rw-r--r-- 1 root root  622 Apr  3  2018 mysplit.c
419878 -rwxr-xr-x 1 root root 8.8K Mar 25 20:52 mystop
419861 -rw-r--r-- 1 root root  624 Apr  3  2018 mystop.c
419847 -rw-r--r-- 1 root root  761 Apr  3  2018 README
419842 -rwxr-xr-x 1 root root 5.1K Apr  3  2018 sdriver.pl
419859 -rw-r--r-- 1 root root   58 Apr  3  2018 trace01.txt
419841 -rw-r--r-- 1 root root   60 Apr  3  2018 trace02.txt
419857 -rw-r--r-- 1 root root   67 Apr  3  2018 trace03.txt
419734 -rw-r--r-- 1 root root   89 Apr  3  2018 trace04.txt
419849 -rw-r--r-- 1 root root  171 Apr  3  2018 trace05.txt
419848 -rw-r--r-- 1 root root  108 Apr  3  2018 trace06.txt
419862 -rw-r--r-- 1 root root  187 Apr  3  2018 trace07.txt
419852 -rw-r--r-- 1 root root  189 Apr  3  2018 trace08.txt
419858 -rw-r--r-- 1 root root  230 Apr  3  2018 trace09.txt
419843 -rw-r--r-- 1 root root  227 Apr  3  2018 trace10.txt
419854 -rw-r--r-- 1 root root  173 Apr  3  2018 trace11.txt
419813 -rw-r--r-- 1 root root  203 Apr  3  2018 trace12.txt
419856 -rw-r--r-- 1 root root  253 Apr  3  2018 trace13.txt
419736 -rw-r--r-- 1 root root  448 Apr  3  2018 trace14.txt
419756 -rw-r--r-- 1 root root  456 Apr  3  2018 trace15.txt
419846 -rw-r--r-- 1 root root  256 Apr  3  2018 trace16.txt
419870 -rwxr-xr-x 1 root root  15K Mar 25 20:52 tsh
419855 -rw-r--r-- 1 root root  12K Apr  3  2018 tsh.c
419844 -rwxr-xr-x 1 root root  19K Apr  3  2018 tshref
419860 -rw-r--r-- 1 root root 6.0K Apr  3  2018 tshref.out
```

Compare your shell with a reference solution (tshref)



```
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab# ./tsh
tsh> quit
tsh>
```

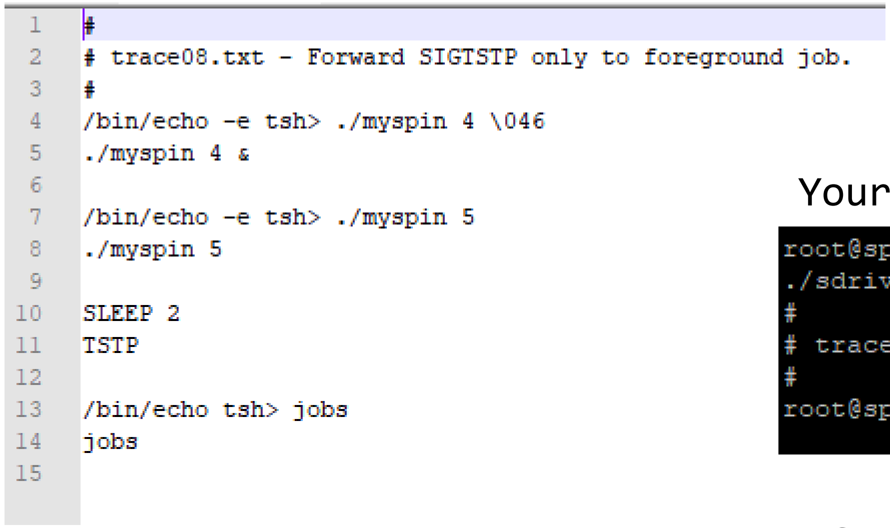


```
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab# ./tshref
tsh> quit
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab#
```

DCSLAB

Let's start the fun part!

trace08.txt

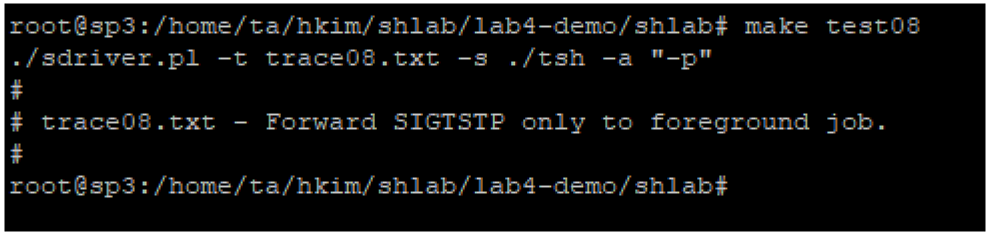


```
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo -e tsh> ./myspin 5
./myspin 5

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs

```

Your Output

# make test{NN}

# make test08

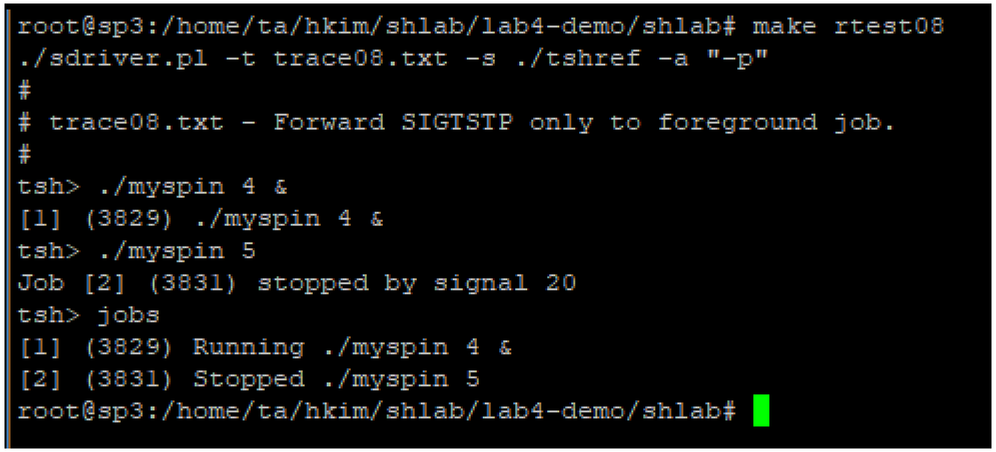


```
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab# make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab#
```

Reference Output
- the solution

# make rtest{NN}

# make rtest08

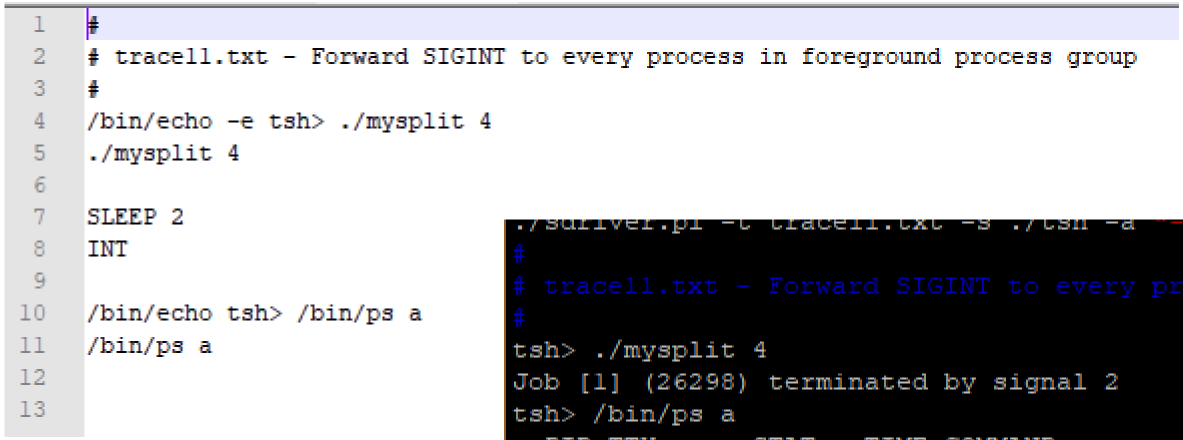


```
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab# make rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (3829) ./myspin 4 &
tsh> ./myspin 5
Job [2] (3831) stopped by signal 20
tsh> jobs
[1] (3829) Running ./myspin 4 &
[2] (3831) Stopped ./myspin 5
root@sp3:/home/ta/hkim/shlab/lab4-demo/shlab#
```

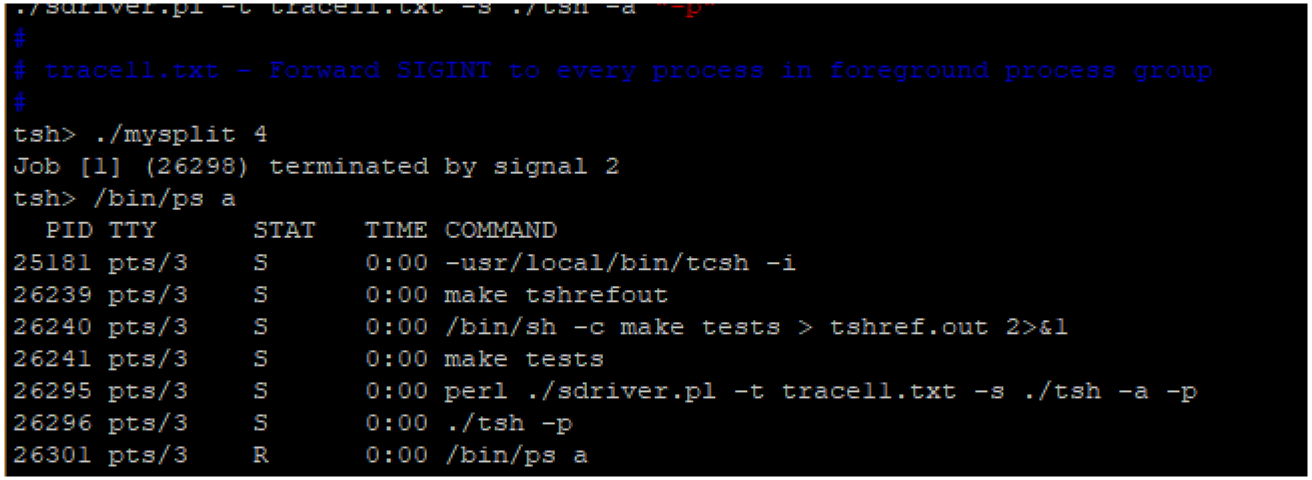
DCSLAB

Let’s start the fun part!

trace11.txt



```
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
/bin/echo -e tsh> ./mysplit 4
./mysplit 4

SLEEP 2
INT

/bin/echo tsh> /bin/ps a
/bin/ps a


```



```
./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] (26298) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
25181 pts/3    S      0:00 -usr/local/bin/tcsh -i
26239 pts/3    S      0:00 make tshrefout
26240 pts/3    S      0:00 /bin/sh -c make tests > tshref.out 2>&1
26241 pts/3    S      0:00 make tests
26295 pts/3    S      0:00 perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p
26296 pts/3    S      0:00 ./tsh -p
26301 pts/3    R      0:00 /bin/ps a
```

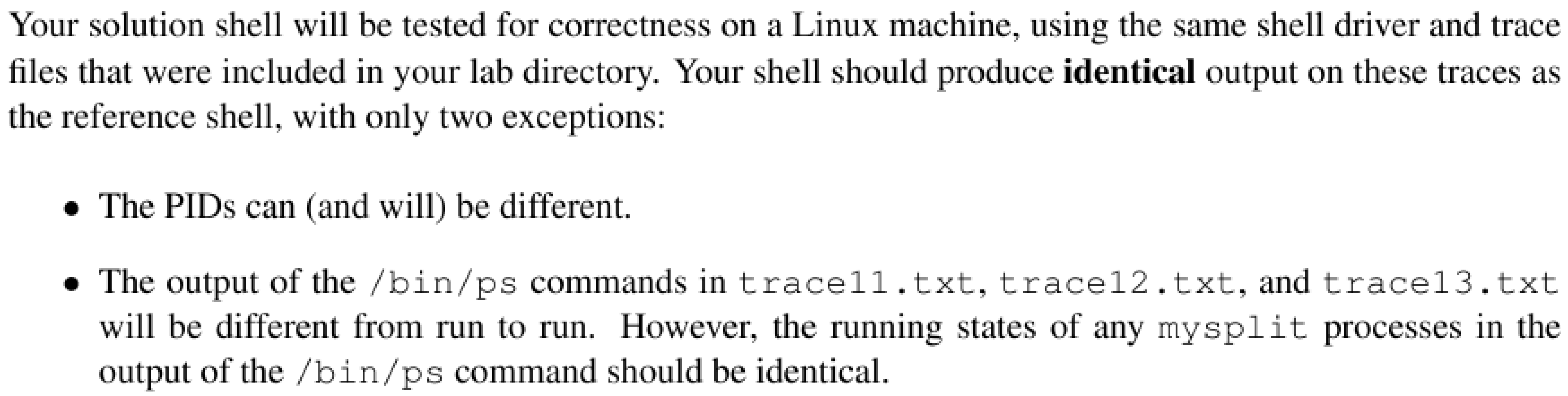
- The output of the /bin/ps commands in trace11.txt, trace12.txt, and trace13.txt will be different from run to run. However, the running states of any mysplit processes in the output of the /bin/ps command should be identical.

DCSLAB

Checking Your Work

- Reference Solution
  - The linux executable tshref is the reference solution for the shell
  - **Your shell should emit output that is *identical* to the reference solution**

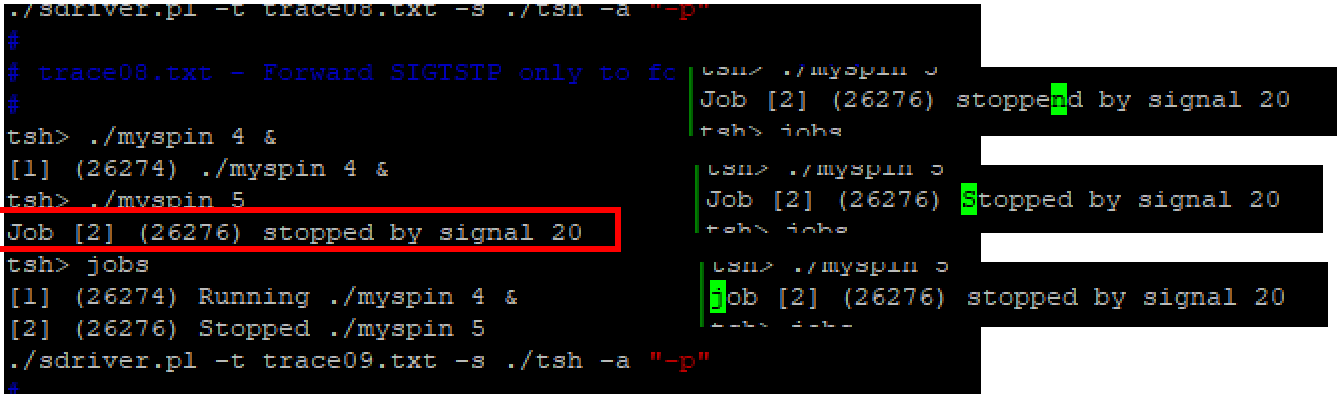
- Shell driver
  - The `sdriver.pl` program
    - executes a shell as a child process,
    - sends it commands and signals as directed by a trace file,
    - and captures and displays the output from the shell



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 should 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.

Errors from last semester

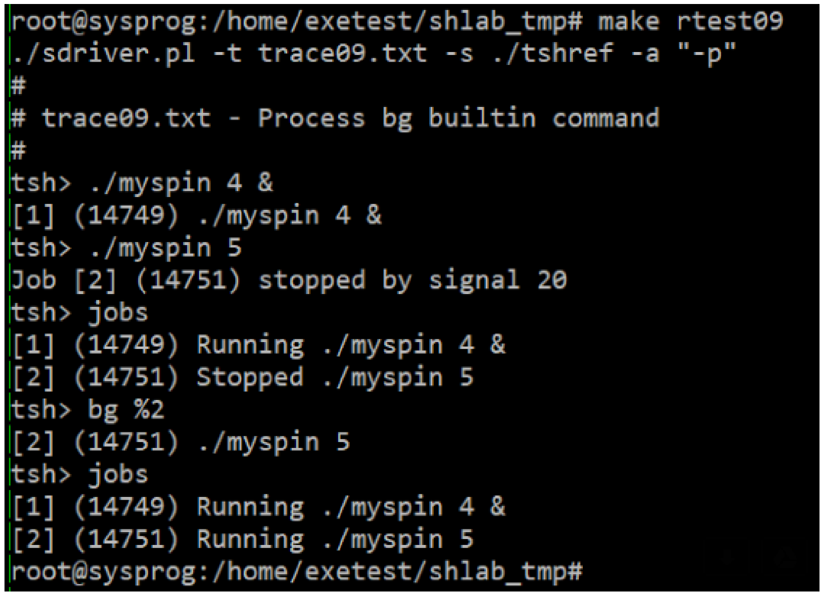


```
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to fc
#
tsh> ./myspin 4 &
[1] (26274) ./myspin 4 &
tsh> ./myspin 5
Job [2] (26276) stopped by signal 20
tsh> jobs
[1] (26274) Running ./myspin 4 &
[2] (26276) Stopped ./myspin 5
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
```

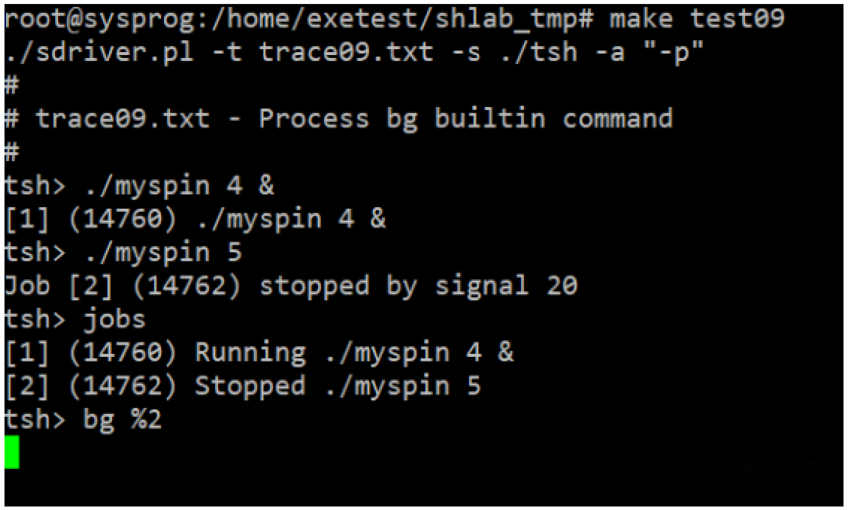
```
tsh> ./myspin 5
Job [2] (26276) stoppend by signal 20
tsh> jobs
```

```
tsh> ./myspin 5
Job [2] (26276) Stopped by signal 20
tsh> jobs
```

```
tsh> ./myspin 5
job [2] (26276) stopped by signal 20
tsh> jobs
```



```
root@sysprog:/home/exetest/shlab_tmp# make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (14749) ./myspin 4 &
tsh> ./myspin 5
Job [2] (14751) stopped by signal 20
tsh> jobs
[1] (14749) Running ./myspin 4 &
[2] (14751) Stopped ./myspin 5
tsh> bg %2
[2] (14751) ./myspin 5
tsh> jobs
[1] (14749) Running ./myspin 4 &
[2] (14751) Running ./myspin 5
root@sysprog:/home/exetest/shlab_tmp#
```



```
root@sysprog:/home/exetest/shlab_tmp# make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (14760) ./myspin 4 &
tsh> ./myspin 5
Job [2] (14762) stopped by signal 20
tsh> jobs
[1] (14760) Running ./myspin 4 &
[2] (14762) Stopped ./myspin 5
tsh> bg %2
```

Questions from previous semester

- verbose 옵션 대응까지 과제의 범위 안에 있는지 궁금합니다
  - 채점 시에 verbose 옵션 구현 여부 체크 하지 않습니다.

- Non-local jump 사용하여 구현해도 괜찮은가요?
  - Jump를 사용하지 않고 구현하시기 바랍니다.

- 채점 기준의 check the return value of every system call (5pt) 관련 sigemptyset, sigaddset, sigprocmask, kill 등의 함수들의 return value 도 모두 체크해줘야 하나요?
  - 네 체크하시기 바랍니다.

- tshref/ref 내부에서 ./myspin: Command not found 에러
  - make 실행하셔서 myspin / myint / mysplit / mystop 실행파일 생성하신 후 실행하시기 바랍니다.

Additional announcement about evaluation

- You may assume that there're no input errors that are not specified in the trace files or the assignment specifications.

➢ **Your shell should emit output that is *identical* to the reference solution**

  - Evaluation will processed with trace files included in `shlab.tar`
  - We'll not accept any objection related to evaluation

Fin.

- Due: Apr. 14th

- Questions
  - etl Q&A Board
  - sp_tas@dcslab.snu.ac.kr

- Next class(Apr 7th)…
  - Lab #2 Q&A