Shell Lab: Writing Your Own Unix Shell

20190084 권민재, CSED211

개요

이 과제는 주어진 코드를 바탕으로 다른 프로세스를 실행하고 시그널을 핸들링 하는 등의 역할을 수행하는 tsh를 작성하는 것이다.

구현

eval

이 함수는 사용자의 입력이 빌트인 커맨드가 아닌 적절한 명령일 때 해당 명령을 foreground나 background로 실행시켜주는 함수이다.

```
// parseline을 통해 argv에 아규먼트 리스트를 세팅하고, 백그라운드 여부를 isBackgroundJob에 저장
int isBackgroundJob = parseline(cmdline, argv);

// 올바르지 않은 입력일 경우 종료.
// i.e. 아무것도 입력되지 않았거나, built-in 커맨드인 경우.
if(!argv[0] || builtin_cmd(argv)) {
    return;
}
```

이 함수가 우선 실행되면, parseline을 이용하여 cmdline을 파싱한 결과를 argv에 넣는다. parseline의 결과값은 백그라운드 여부이기 때문에, 해당 함수의 반환값을 isBackgroundJob에 저장한다. 이후, 파싱한 argv에서 첫번째 인자가 존재하지 않거나, 주어진 인자가 빌트인 커맨드인 경우에는 함수를 종료한다.

```
// sigemptyset을 수행하고, 실패했을 때에는 에러를 표시한다..
if(sigemptyset(&_sigset) < 0){
    unix_error("sigemptyset error");
}

// SIGINT, SIGCHLD, SIGSTOP에 대해 sigaddset을 수행하고, 실패할 경우에는 에러를 표시한다.
for(i = 0; i < 3; ++i){
    if(sigaddset(&_sigset, signalValueList[i]) != 0){
        unix_error("sigaddset error");
```

```
    }
}

// sigprocmask를 수행하고, 실패할 경우에는 에러를 표시한다.
if(sigprocmask(SIG_BLOCK, &_sigset, 0) < 0){
    unix_error("sigprocmask error");
}
```

이후, 정상적인 입력이라면 sigset을 세팅한다. sigset을 세팅할 때에는 signalValueList에 들어있는 세가지 Signal `SIGINT`, `SIGCHLD`, `SIGSTOP`을 핸들링할 수 있도록 sigset에 추가한다. 3가지를 추가해야하는 이유는, trace16에서 이 세가지를 핸들링할 수 있나 점검하기 때문이다. 이후, sigprocmask를 통해 블럭을 수행한다.

```
// fork를 수행하고, 실패할 경우에는 에러를 표시한다.
if((pidFork = fork()) < 0){
    unix_error("fork error");
}

// child process인 경우
if(pidFork == 0){
    // 우선, 현재 sigset을 unblock한다.
    sigprocmask(SIG_UNBLOCK, &_sigset, 0);

    // 이후, setpgid를 통해 프로세스 그룹을 설정하고, 실패할 경우에는 에러를 띄운다.
    if(setpgid(0, 0) < 0){
        unix_error("setpgid error");
    }

    // execve를 통해 주어진 인자를 실행하고, 실패할 경우에는 커맨드가 존재하지 않는다는 에러를 띄운
다.
    if(execve(argv[0], argv, environ) < 0){
        printf("%s: Command not found\n", argv[0]);
        exit(1);
    }
    return;
}
```

이후, 포크를 진행하고, child process인 경우에는 우선 sigprocmask를 통해 언블럭을 진행한다. 이후, setpgid를 통해 프로세스 그룹을 지정해 준 이후 `execve`로 사용자의 입력을 실행한다. 실패했을 경우에는 커맨드를 찾지 못했다는 에러를 출력한다.

```
// addjob을 통해 Job 배열에 포크된 프로세스 아이디를 추가한다.
addjob(jobs, pidFork, isBackgroundJob ? BG : FG, cmdline);

// sigprocmask를 통해 unblock을 수행한다.
```

```
if(sigprocmask(SIG_UNBLOCK, &_sigset, 0) < 0){
    unix_error("sigprocmask error");
}

// 만약 백그라운드 Job이 아니라면, Job이 종료될 때 까지 대기한다.
if(!isBackgroundJob){
    waitfg(pidFork);
    return;
}

// background job이라면, 실행 정보를 출력한다.
if(pidFork){
    printf("[%d] (%d) %s", pid2jid(pidFork), pidFork, cmdline);
}
```

부모 프로세스, 즉 쉘에서는 addjob을 통해 job list에 background / foreground 여부와 함께 저장한다. 이후 여기서도 역시 언블록을 진행해 준 뒤, foreground job이라면 waitfg를 이용하여 job이 종료될 때 까지 대기한다. Background job이라면, 실행 정보를 출력한다.

builtin_cmd

이 함수는 주어진 커맨드가 빌트인 커맨드인지 확인하고, 빌트인 커맨드라면 그를 수행하고 1을 반환하고, 빌트인 커맨드가 아니라면 0을 반환하는 함수이다.

```
// 인자들 중 첫번째 인자를 Cmd로 설정한다.
char* cmd = argv[0];

if(strcmp(cmd, "quit") == 0){
    // cmd가 quit 이라면 쉘을 종료한다.
    exit(0);
} else if (strcmp(cmd, "fg") == 0 || strcmp(cmd, "bg") == 0) {
    // cmd가 fg나 bg라면 do_bgfg를 호출한다.
    do_bgfg(argv);
} else if (strcmp(cmd, "jobs") == 0) {
    // cmd가 jobs라면 listjobs를 이용해 출력한다.
    listjobs(jobs);
} else {
    // cmd가 built-in cmd가 아닌 경우 0을 반환한다.
    return 0;
}
// cmd가 built-in cmd인 경우이므로 1을 반환한다.
return 1;
```

사용자의 입력이 quit이라면 exit(0)을 통해 쉘을 종료하고, fg이거나 bg이면 do_bgfg를 호출한다. 만약, 사용자의 입력이 jobs라면 listjobs를 호출한다. quit, fg, bg, jobs 중에 하나였다면 1을 반환하고, 아니었다면 0을 반환한다.

do_bgfg

이 함수는 `bg`일 경우에는 멈춘 background job을 background에서 실행시키고, `fg`인 경우에는 background job을 foreground에서 실행하도록 하는 함수이다.

```
// 만약 올바르지 않은 입력인데 do_bgfg가 호출되었다면, 에러를 표시하고 종료한다.
if(!argv[0] || !(strcmp(argv[0], "fg") == 0 || strcmp(argv[0], "bg") == 0)){
    printf("do_bgfg: Internal error");
    exit(0);
}

// 만약 두번째 인자가 존재하지 않는다면, 인자가 필요함을 표시하고 종료한다.
if(!argv[1]){
    printf("%s command requires PID or %%jobid argument\n", argv[0]);
    return;
}

// 두번째 인자를 param에 저장한다.
param = argv[1];

// 첫번째 글자가 %라면, JobID 이므로 isJobId를 true로 설정하고, 아니라면 false로 설정한다.
isJobId = (param[0] == '%');

// 두번째 인자가 올바른 인자인지, 즉 숫자로 이루어져 있는 인자인지 체크하자.
// Job ID 인 경우에는 인덱스 1부터, Process ID인 경우에는 인덱스 0부터, 각 글자가 숫자인지 체크한다.
for(i = isJobId; ; i++){
    // null 문자일 경우 종료
    if(param[i] == '\0'){
        break;
    }

    // i번째 인덱스의 글자가 숫자가 아닌 경우
    if(!isdigit(param[i])){
        // isDigit을 false로 만들고 반복 종료.
        isDigit = 0;
        break;
    }
}

// param이 올바르지 않은, 즉 숫자로만 이루어져 있지 않은 경우 에러를 띄우고 종료
if(!isDigit) {
    printf("%s: argument must be a PID or %%jobid\n", cmd);
    return;
}
```

이 함수가 실행될 때에는 우선 사용자의 입력을 정제한다. 사용자의 입력이 비어있거나, fg나 bg가 아니라면 이 함수가 불리면 안되는 상황이기에 곧바로 종료시킨다. 또한, 두번째 인자가 존재하지 않는다면 두번째 인자가 필요함을 표시하고 종료한다. 이후 첫 글자를 검사하여 Job ID인지 Process ID 인지 그 여부를 저장하고, 그를 바탕으로 %를 제외한 모든 글자가 숫자인지 검증하여 숫자가 아니라면 에러를 띄운다. 이러한 입력 정제 과정을 거친 이후에 본격적으로 함수의 루틴이 시작된다.

```
if(isJobId) {
    // Job ID인 경우, ID를 파싱하고 getjobjid를 통해 job 조회
    jid = (int) strtol(param + 1, NULL, 10);
    job = getjobjid(jobs, jid);
} else {
    // Process ID인 경우, ID를 파싱하고 getjobpid를 통해 job 조회
    pid = strtol(param, NULL, 10);
    job = getjobpid(jobs, pid);
}

// 해당하는 job이 존재하지 않는 경우, Job ID 여부에 따라 적절한 에러 메시지 출력
if(!job){
    if(isJobId){
        printf("%%%d: No such job\n", jid);
    } else {
        printf("(%d): No such process\n", pid);
    }
    return;
}
```

우선, 앞서 구했던 Job ID 여부를 이용하여 적절하게 `getjobjid`나 `getjobpid`를 호출하여 사용자가 원하는 Job을 찾아내고, 만약 그에 해당하는 Job이 존재하지 않는다면 에러를 띄우고 종료한다.

```
pid = job->pid; // job의 pid 설정
isFG = (strcmp(cmd, "fg") == 0); // foreground 여부 저장

killResult = kill(-pid, SIGCONT); // 프로세스에 SIGCONT 전송

if(isFG){
    // fg인 경우

    // 우선 kill에 실패했을 경우 에러를 띄운다.
    if(killResult < 0) {
        unix_error("kill (fg) error");
    }

    // Job의 상태를 FG로 바꾸고 job이 종료되기를 기다린다.
    job->state = FG;
    waitfg(pid);
```

```
} else {
    // bg인 경우

    // 우선 kill에 실패했을 경우 에러를 띄운다.
    if(killResult < 0) {
        unix_error("kill (bg) error");
    }

    // Job의 상태를 BG로 바꾸고 상태를 출력한다.
    job->state = BG;
    printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);
}
```

이후, 앞서 찾았던 job의 pid로 SIGCONT 시그널을 보내서 실행을 요구한다. 만약 kill에 실패했다면, `fg`나 `bg`가 입력된 각 경우에 알맞게 에러 메시지를 출력한다. 에러가 나지 않은 경우에는 각 입력에 따라 나머지 행동을 수행한다. 만약, `fg`가 입력된 경우라면, job을 foreground에 올려야 하므로, job의 state를 FG로 변경하고, 해당 Job이 종료될 때 까지 대기될 수 있도록 waitfg를 호출한다. 만약, `bg`가 입력되었다면, job의 state를 BG로 변경하고, Job의 상태를 출력한다.

waitfg

이 함수는 foreground job이 종료될 때 까지 대기하는 함수이다.

```
struct job_t* job = getjobpid(jobs, pid); // pid로부터 Job을 조회
if(job){
    // 만약 job이 존재한다면,
    while(job->state == FG){
        // job의 상태가 FG인 동안 sleep.
        sleep(1);
    }
}
```

우선 주어진 pid를 이용하여 job을 조회하고, 만약 해당 job이 존재한다면 해당 Job의 상태가 FG인 동안 sleep을 수행한다.

sigchld_handler

이 함수는 SIGCHLD 시그널을 핸들링하는 함수이다.

```
// 자식 프로세스 대기 및 상태 회수
while((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0)
```

우선 모든 자식 프로세스에 대해 검사하기 위해 waitpid의 첫번째 인자에 -1을 넣어서 임의의 자식 프로세스에 대해 대기하고 그 반환값과 상태를 업데이트 한다.

```
if (WIFSTOPPED(status)) { // 자식 프로세스가 정지된 상태일 때
    // 우선 job을 조회한다.
    job = getjobpid(jobs, pid);
    if(!job){ // job이 존재하지 않는 경우, 에러를 표시하고 종료한다.
        printf("Lost track of (%d)\n", pid);
        return;
    }

    job->state = ST; // Job의 상태를 ST로 설정

    // Job 상태 출력
    // WSTOPSIG를 통해 프로세스를 정지시킨 시그널 조회
    printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid), pid,
WSTOPSIG(status));
}
```

자식 프로세스가 현재 정지되어 있는 상태(WIFSTOPPED)라면, 우선 job을 조회한다. 해당 Job이 존재하지 않는 경우에는 에러를 띄운다. 에러가 나지 않았다면, 해당 job의 상태를 ST로 설정하고, WSTOPSIG를 통해 자식 프로세스를 정지시킨 원인이 된 시그널을 출력한다.

```
else {
    if(WIFEXITED(status)){ // 자식 프로세스가 EXIT된 상태일 때
        if(WTERMSIG(status)){ // 자식 프로세스를 종료시킨 시그널이 0이 아닐 때
            // 에러 출력
            unix_error("waitpid error");
        }
    } else if (WIFSIGNALED(status)) { // 자식 프로세스가 어떤 신호를 받아서 종료되었을 때
        // Job 상태 출력
        // WTERMSIG를 통해 종료하게 만든 시그널 출력
        printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid,
WTERMSIG(status));
    }

    // Job 삭제
    deletejob(jobs, pid);
}
```

자식 프로세스가 정상적으로 종료된 상태(WIFEXITED)일 때, 종료시킨 시그널이 0이 아니라면 에러를 띄운다. 자식 프로세스가 어떤 시그널을 받아서 종료되었을 때(WIFSIGNALED)에는, WTERMSIG를 통해 종료의 원인이 된 시그널을 출력한다. 이후 deletejob을 통해 job을 삭제한다.

sigint_handler

이 함수는 SIGINT 시그널을 핸들링하는 함수이다.

```
pid_t pid = fgpid(jobs); // fgpid로 foreground job 조회
// pid가 존재할 때, kill 시도.
// 실패할 경우 에러 출력
if(pid && (kill(-pid, sig) < 0)){
    unix_error("kill (sigint) error");
}
```

우선 현재 foreground job을 조회한 이후, sig (SIGINT)로 kill을 시도한다. 만약 kill에 실패했다면, 에러를 출력한다.

sigtstp_handler

이 함수는 SIGTSTP 시그널을 핸들링하는 함수이다.

```
pid_t pid = fgpid(jobs); // fgpid로 foreground job 조회
// pid가 존재할 때, kill 시도.
// 실패할 경우 에러 출력
if(pid && (kill(-pid, sig) < 0)){
    unix_error("kill (tstp) error");
}
```

우선 현재 foreground job을 조회한 이후, SIGTSTP로 kill을 시도한다. 만약 kill에 실패했다면, 에러를 출력한다.

결과

trace01

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

trace02

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

trace03

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

trace04

```
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (330) ./myspin 1 &
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (335) ./myspin 1 &
```

trace05

```
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (341) ./myspin 2 &
tsh> ./myspin 3 &
```

```
[2] (343) ./myspin 3 &
tsh> jobs
[1] (341) Running ./myspin 2 &
[2] (343) Running ./myspin 3 &
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (349) ./myspin 2 &
tsh> ./myspin 3 &
[2] (351) ./myspin 3 &
tsh> jobs
[1] (349) Running ./myspin 2 &
[2] (351) Running ./myspin 3 &
```

trace06

```
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (358) terminated by signal 2
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (363) terminated by signal 2
```

trace07

```
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (369) ./myspin 4 &
tsh> ./myspin 5
Job [2] (371) terminated by signal 2
tsh> jobs
[1] (369) Running ./myspin 4 &
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
```

```
tsh> ./myspin 4 &
[1] (377) ./myspin 4 &
tsh> ./myspin 5
Job [2] (379) terminated by signal 2
tsh> jobs
[1] (377) Running ./myspin 4 &
```

trace08

```
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (386) ./myspin 4 &
tsh> ./myspin 5
Job [2] (388) stopped by signal 20
tsh> jobs
[1] (386) Running ./myspin 4 &
[2] (388) Stopped ./myspin 5
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (394) ./myspin 4 &
tsh> ./myspin 5
Job [2] (396) stopped by signal 20
tsh> jobs
[1] (394) Running ./myspin 4 &
[2] (396) Stopped ./myspin 5
```

trace09

```
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (403) ./myspin 4 &
tsh> ./myspin 5
Job [2] (405) stopped by signal 20
tsh> jobs
[1] (403) Running ./myspin 4 &
[2] (405) Stopped ./myspin 5
tsh> bg %2
[2] (405) ./myspin 5
```

```
tsh> jobs
[1] (403) Running ./myspin 4 &
[2] (405) Running ./myspin 5
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (413) ./myspin 4 &
tsh> ./myspin 5
Job [2] (415) stopped by signal 20
tsh> jobs
[1] (413) Running ./myspin 4 &
[2] (415) Stopped ./myspin 5
tsh> bg %2
[2] (415) ./myspin 5
tsh> jobs
[1] (413) Running ./myspin 4 &
[2] (415) Running ./myspin 5
```

trace10

```
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (424) ./myspin 4 &
tsh> fg %1
Job [1] (424) stopped by signal 20
tsh> jobs
[1] (424) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (433) ./myspin 4 &
tsh> fg %1
Job [1] (433) stopped by signal 20
tsh> jobs
[1] (433) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

trace11

```
./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] (443) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss+    0:00 bash
  287 pts/1    Ss     0:00 /bin/bash
  438 pts/1    S+     0:00 make rtest11 test11
  439 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tshref -
a "-p"
  440 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s
./tshref -a -p
  441 pts/1    S+     0:00 ./tshref -p
  446 pts/1    R      0:00 /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] (451) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss+    0:00 bash
  287 pts/1    Ss     0:00 /bin/bash
  438 pts/1    S+     0:00 make rtest11 test11
  447 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a
"-p"
  448 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -
a -p
  449 pts/1    S+     0:00 ./tsh -p
  454 pts/1    R      0:00 /bin/ps a
```

trace12

```
./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] (460) stopped by signal 20
tsh> jobs
[1] (460) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss+    0:00 bash
```

```
  287 pts/1    Ss     0:00 /bin/bash
  455 pts/1    S+     0:00 make rtest12 test12
  456 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -
a "-p"
  457 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s
./tshref -a -p
  458 pts/1    S+     0:00 ./tshref -p
  460 pts/1    T      0:00 ./mysplit 4
  461 pts/1    T      0:00 ./mysplit 4
  464 pts/1    R      0:00 /bin/ps a
./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] (469) stopped by signal 20
tsh> jobs
[1] (469) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss+    0:00 bash
  287 pts/1    Ss     0:00 /bin/bash
  455 pts/1    S+     0:00 make rtest12 test12
  465 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a
"-p"
  466 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -
a -p
  467 pts/1    S+     0:00 ./tsh -p
  469 pts/1    T      0:00 ./mysplit 4
  470 pts/1    T      0:00 ./mysplit 4
  473 pts/1    R      0:00 /bin/ps a
```

trace13

```
./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (479) stopped by signal 20
tsh> jobs
[1] (479) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss+    0:00 bash
  287 pts/1    Ss     0:00 /bin/bash
  474 pts/1    S+     0:00 make rtest13 test13
```

```
  475 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -
a "-p"
  476 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s
./tshref -a -p
  477 pts/1    S+     0:00 ./tshref -p
  479 pts/1    T      0:00 ./mysplit 4
  480 pts/1    T      0:00 ./mysplit 4
  483 pts/1    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss+    0:00 bash
  287 pts/1    Ss     0:00 /bin/bash
  474 pts/1    S+     0:00 make rtest13 test13
  475 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -
a "-p"
  476 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s
./tshref -a -p
  477 pts/1    S+     0:00 ./tshref -p
  486 pts/1    R      0:00 /bin/ps a
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (491) stopped by signal 20
tsh> jobs
[1] (491) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss+    0:00 bash
  287 pts/1    Ss     0:00 /bin/bash
  474 pts/1    S+     0:00 make rtest13 test13
  487 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a
"-p"
  488 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -
a -p
  489 pts/1    S+     0:00 ./tsh -p
  491 pts/1    T      0:00 ./mysplit 4
  492 pts/1    T      0:00 ./mysplit 4
  495 pts/1    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss+    0:00 bash
  287 pts/1    Ss     0:00 /bin/bash
  474 pts/1    S+     0:00 make rtest13 test13
  487 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a
"-p"
```

```
  488 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -
a -p
  489 pts/1    S+     0:00 ./tsh -p
  498 pts/1    R      0:00 /bin/ps a
```

trace14

```
./sdriver.pl -t trace14.txt -s ./tshref -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (506) ./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] (506) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (506) ./myspin 4 &
tsh> jobs
[1] (506) Running ./myspin 4 &
./sdriver.pl -t trace14.txt -s ./tsh -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (524) ./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] (524) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (524) ./myspin 4 &
tsh> jobs
[1] (524) Running ./myspin 4 &
```

trace15

```
./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] (543) terminated by signal 2
tsh> ./myspin 3 &
[1] (545) ./myspin 3 &
tsh> ./myspin 4 &
[2] (547) ./myspin 4 &
tsh> jobs
[1] (545) Running ./myspin 3 &
[2] (547) Running ./myspin 4 &
tsh> fg %1
Job [1] (545) stopped by signal 20
tsh> jobs
[1] (545) Stopped ./myspin 3 &
[2] (547) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (545) ./myspin 3 &
tsh> jobs
[1] (545) Running ./myspin 3 &
```

```
[2] (547) Running ./myspin 4 &
tsh> fg %1
tsh> quit
./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] (562) terminated by signal 2
tsh> ./myspin 3 &
[1] (564) ./myspin 3 &
tsh> ./myspin 4 &
[2] (566) ./myspin 4 &
tsh> jobs
[1] (564) Running ./myspin 3 &
[2] (566) Running ./myspin 4 &
tsh> fg %1
Job [1] (564) stopped by signal 20
tsh> jobs
[1] (564) Stopped ./myspin 3 &
[2] (566) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (564) ./myspin 3 &
tsh> jobs
[1] (564) Running ./myspin 3 &
[2] (566) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

trace16

```
./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] (580) stopped by signal 20
tsh> jobs
[1] (580) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (583) terminated by signal 2
./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] (588) stopped by signal 20
tsh> jobs
[1] (588) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (591) terminated by signal 2
```

토론 및 개선

- 리다이렉트나 파이프를 이용할 수 있도록 개선하면 좋을 것으로 보인다.
- 환경변수 (environment)가 제대로 전달되지 않는데, 이 점을 개선하면 좋을 것 같다.